Automatic Syntactic Processing of Modern Hebrew

Dissertation submitted in partial fulfillment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY

by

Yoav Goldberg

Submitted to the Senate of
Ben-Gurion University of the Negev

November 2011
Beer-Sheva

Automatic Syntactic Processing of Modern Hebrew

Dissertation submitted in partial fulfillment
of the requirements for the degree of
DOCTOR OF PHILOSOPHY

by

Yoav Goldberg

Submitted to the Senate of
Ben-Gurion University of the Negev

Author signature: _____
Approved by the advisor: _____
Approved by the Dean of the Kreitman School of Advanced Graduate Studies: _____

November 2011
Beer-Sheva

This work was carried out under the supervision of Prof. Michael Elhadad

In the Department of **Computer Science**
Faculty: **Natural Sciences**

To those who find it interesting
and those who don't

Contents

Abstract vi

Acknowledgements x

List of Figures xiii

List of Tables xiii

1 Introduction 1

1.1 Main themes . . . 3

1.1.1 Separation of lexical and syntactic knowledge . . . 3

1.1.2 Using morphological information to improve parsing accuracy . 4

1.1.3 Encoding input uncertainty using a lattice-based representation . 4

1.2 Contributions . . . 4

1.2.1 Parsing systems for Modern Hebrew . . . 4

1.2.2 Modern Hebrew dependency Treebank . . . 5

1.2.3 Algorithmic and methodological contributions . . . 5

1.3 Structure of this thesis . . . 6

I Background 8

2 Syntactic Representations of Language 9

2.1 Syntactic structures . . . 9

2.1.1 Chunks . . . 9

2.1.2 Dependency trees . . . 10

2.1.3 Constituency trees . . . 13

2.1.4 Constituency to dependency conversion . . . 14

2.2 Syntactic disambiguation . . . 15

2.3 Grammar-based vs. data-driven parsers . . . 17

2.4 Applications of syntactic structure 18

3 Modern Hebrew **21**
3.1 Lexical level 21
3.1.1 Unvocalized orthography 21
3.1.2 Affixation 21
3.1.3 Rich templatic morphology 22
3.1.4 The participle form 23
3.2 Syntactic level 23
3.2.1 Relatively free constituent order 23
3.2.2 Verbless constructions 24
3.2.3 NP structure and construct-state 24
3.2.4 Definiteness 24
3.2.5 Case marking 25
3.2.6 Agreement 25

4 Automatic Syntactic Disambiguation in Modern Hebrew **26**
4.1 Aspects of Modern Hebrew parsing 26
4.1.1 Small amount of annotated data 26
4.1.2 Ambiguous word segmentation 27
4.1.3 Morphological variation and high out-of-vocabulary rate 27
4.1.4 Morphological agreement 29
4.2 Existing resources for Hebrew text processing 29
4.2.1 The Hebrew constituency-Treebank 29
4.2.2 The MILA broad-coverage lexicon 31
4.2.3 Hebrew morphological disambiguator 33
4.2.4 A resource-incompatibility issue 33
4.3 Summary 34

II Dependency Parsing **36**

5 Hebrew Dependency Treebank **37**
5.1 Dependency annotation guidelines 37
5.1.1 Tagging and segmentation 37
5.1.2 Inter-word relations 38
5.1.3 Dependency labels 44
5.2 Constituency to dependency conversion 45

5.2.1 Fixes and modifications to the Hebrew constituency-Treebank . 46
5.2.2 Head-assignment rules 47
5.2.3 Treebank statistics 49
5.3 Summary 49

6 Background on Dependency Parsing **52**
6.1 Evaluation measures 52
6.2 Dependency-parsing algorithms 54
6.2.1 Transition-based (Shift Reduce) dependency parsing 54
6.2.2 Graph-based dependency parsing 57
6.2.3 Hybrid-systems and ensemble-methods 59
6.2.4 Labeled dependency parsing 59
6.3 Dep. parsing of morphologically rich languages 60

7 EASYFIRST Dependency Parsing **62**
7.1 Non-directional easy-first parsing 62
7.2 The transition system 63
7.3 Parsing algorithm 65
7.4 Learning algorithm 67
7.5 Feature representation 70
7.6 Computational complexity 72
7.7 Evaluation on English 77
7.7.1 Parse diversity 79
7.7.2 Error analysis / limitations 80
7.8 Related work 81
7.9 Chapter summary 82

8 Hebrew Dependency Parsing **83**
8.1 Architecture 83
8.1.1 Justification for a pipeline architecture 83
8.1.2 Lexicon/syntax separation 85
8.2 Data and baseline experiments 85
8.2.1 Baseline results 86
8.3 Agreement in the EASYFIRST parser 89
8.3.1 Cases where agreement can disambiguate attachments 89
8.3.2 Easy-first morphological agreement features 92
8.3.3 Evaluation and results 95
8.4 An ensemble-based parser 96

8.4.1 Evaluation and results . . . 97
8.5 Adding an edge labeler . . . 97
8.5.1 Evaluation and results . . . 99
8.6 Evaluating the final system . . . 100
8.7 Conclusions . . . 101

III Constituency Parsing 103

9 Background on Constituency Parsing 104
9.1 Evaluation measures . . . 104
9.2 Probabilistic context-free grammars . . . 105
9.3 Grammar binarization . . . 107
9.4 The CKY algorithm . . . 108
9.5 Treebank grammars and grammar refinement . . . 109
9.6 Automatic state-split grammars (PCFG-LA) . . . 110
9.7 Parsing morph. rich languages . . . 112
9.7.1 Arabic . . . 113
9.7.2 Hebrew and relational-realizational parsing . . . 113
9.8 Chapter summary . . . 115

10 A Hebrew Constituency Parser 116
10.1 Baseline experiments . . . 117
10.1.1 Analyzing the Learned PCFG-LA Grammar . . . 118
10.1.2 Limitation of PCFG-LA parsing of Modern Hebrew . . . 120
10.2 Manual state-splits . . . 121
10.3 Better lexical coverage with an external lexicon . . . 122
10.3.1 A unified lexical probability model . . . 123
10.4 Joint segmentation and parsing . . . 127
10.4.1 Lattice representation . . . 127
10.4.2 Lattice parsing . . . 128
10.5 Incorporating morphology . . . 130
10.5.1 Forcing morphologically-motivated splits . . . 130
10.5.2 Agreement as filter . . . 131
10.5.3 The Hebrew agreement filter . . . 133
10.6 Evaluation and results . . . 137
10.7 The final model . . . 139
10.8 Summary . . . 140

11 Conclusions and Future Work **142**
11.1 Open Issues 146

Bibliography **148**

Abstract

Language is composed of words, which are combined to form sentences. While single words can convey myriad meanings, it is the combination of words into sentences that allows for efficient communication and the realization of complex ideas. When words are combined to form a sentence, their combination is governed by a set of rules, called the *syntax*, or the *grammar*, of the language. Sentences must obey structural constraints posed by the syntax, and the structure of the sentence determines its meaning. The main focus of this thesis is *syntactic parsing*, the task of assigning syntactic analysis to sentences. The approach taken is data-driven: starting with a list of sentences and their known syntactic structures (a *Treebank*), a *learning algorithm* is trained to produce a *parser* which can assign syntactic structures to novel sentences unseen to the learning algorithm.

In this thesis, I focus on automatic parsing of Modern Hebrew, a semitic language with a rich and productive morphology, relatively free word order and a small Treebank. The primary goal of the thesis is the creation of automatic systems capable of parsing Modern Hebrew text with good accuracy.

A large part of modern parsing literature is devoted to automatic parsing of English, a language with a relatively simple morphology, relatively fixed word order, and a large Treebank. Data-driven English parsing is now at the state where naturally occurring text in the news domain can be automatically parsed with accuracies of around 90% (according to standard parsing evaluation measures). However, when moving from English to languages with richer morphologies and less-rigid word orders, the parsing algorithms developed for English exhibit a large drop in accuracy.

To overcome this drop, I concentrate on aspects in which Modern Hebrew differs from English. In English, word-order is relatively fixed, while in Hebrew as in many other languages word order is much more flexible (for example, the subject may appear either before or after a verb). In languages with flexible word order, the meaning of the sentence is realized using other structural elements, like word inflections or markers, which are referred to as morphology. The lexical units (words) in English are always

separated by white space. In Hebrew (as in Arabic) most words are separated by white space, but many of the function words do not stand on their own and are instead attached to the following words. The resulting combinations (function word + following word) become a single token, which, crucially, can be read ambiguously, as either a single word or a word combination.

Several natural questions arise: can the small size of the Treebank be compensated using other available resources or sources of information? Can morphological information be used effectively in order to improve parsing accuracy? How should the word segmentation issue (that function words do not appear in isolation but attach to the next word, forming ambiguous letter patterns) be handled? Based on these questions, the thesis revolves around the following themes:

Separation of lexical and syntactic knowledge: I argue that the amount of syntactic knowledge needed for a parsing system is relatively limited, and that sufficiently large parts of it can be captured also based on a relatively small Treebank. Lexical knowledge, on the other hand, is much more vast, and we should not rely on a Treebank (small or large) to provide adequate lexical coverage. Instead, we should find ways of integrating lexical knowledge, which is external to the Treebank, into the parsing process.

Using morphological information to improve parsing accuracy: Morphology provides useful hints for resolving syntactic ambiguity, and the parsing model should have a way of utilizing these hints. The morphological clues provided by the language should be taken into account when designing a parsing model for a language, and the parsing algorithm should be flexible enough to accommodate the possible morphological cues.

Delaying dealing with uncertainty: Sometimes, the language signal (the input to the parser) may be uncertain. When computationally feasible, it is best to let this uncertainty be resolved by the parser rather than in a separate pre-processing step. More generally, it is better to defer the resolution of uncertainty as much as possible, preferring to deal with certain decisions before dealing with less certain ones.

The main contribution of this thesis are systems capable of parsing naturally occurring Modern Hebrew text. Specifically, I present a constituency parser and a dependency parser (bundled as open-source software packages) which can parse Hebrew texts with accuracies of 77% F_1 (constituency) and 79.5% UAS (dependency) when starting from raw text, and 85% F_1, 85% UAS when assuming the correct word-segmentation is known. A secondary contribution of the thesis is the creation of a Modern Hebrew dependency annotation guideline and a dependency Treebank, which is created by conversion from the Modern Hebrew constituency Treebank.

The Hebrew parsing systems developed in this work are based on the following

algorithmic and methodological contributions:

For dependency parsing, I present the EASYFIRST parser – an efficient ($O(n \log n)$) dependency parsing algorithm. The algorithm is a greedy, bottom up dependency parser, based on the principle that easy decisions should be made before more difficult ones. For Hebrew, the algorithm provides state-of-the-art parsing results. For English, the algorithm rivals the accuracies of much slower parsing algorithms which are based on exhaustive search. The EASYFIRST dependency parsing algorithm can make use of a rich feature set, which I use to encode morphological agreement information. This work is one of the first to show that agreement information significantly improves the accuracy of a parsing system.

For constituency parsing, I show that parsing performance can be enhanced by utilizing a language resource external to the Treebank: specifically, a lexicon-based morphological analyzer. I present a computational model of interfacing the external lexicon and a Treebank-based parser, also in the common case where the lexicon and the Treebank follow different annotation schemes. I show that Hebrew word-segmentation and constituency-parsing can be performed jointly using CKY lattice parsing. Performing the tasks jointly is effective, and substantially outperforms a pipeline-based model. I suggest modeling grammatical agreement in a constituency-based parser as a filter mechanism that is orthogonal to the grammar, and I present a concrete implementation of the method. While the constituency parser does not make many agreement mistakes to begin with, the filter mechanism is effective in fixing the agreement mistakes that the parser does make.

These contributions extend outside of the scope of Hebrew processing, and are of general applicability to the NLP community. Hebrew is a specific case of a morphologically-rich language, and ideas presented in this thesis are useful also for processing other languages, as well as English. The lattice-based parsing methodology is useful in any case where the input is uncertain. Extending the lexical coverage of a Treebank-derived parser using an external lexicon is relevant for any language with a small Treebank.

Keywords: syntax, morphology, dependency-parsing, constituency-parsing, Modern-Hebrew, text-processing

Acknowledgements

Working on this thesis was a fun and rewarding experience, partly because of the subject matter, but mostly thanks to amazing peoples, ideas and interactions I had along the way.

First and foremost, I owe great thanks to my thesis advisor, Prof. Michael Elhadad. I had an incredible time working with him, and don't think I could have found a better match. Michael has the gift of constantly teaching you things without letting you know that he is doing so, and making you believe you are discovering them on your own. He is sharp and kind and I took something from every interaction with him, and we had quite a few! He let me form my own path, always there with great comments and questions and suggestions, many of which I ignored just to realize, months later, that I really should have listened. I am still not entirely sure how exactly he did it, but I do know for certain that his few, delicate, subtle, well placed actions enhanced me in numerous ways and shaped the way in which I do research, think about things, and even approach life.

BGU's Natural Language Processing core group is a small but spectacular group of people. They really were the best colleagues, office mates and friends one could hope to have. Each quirky in its own way, and all always welcoming, enthusiastic and stimulating, I had an amazing time at the office, every single day. Meni Adler (who thought me to think unsupervised, finds music in anything, and is a real renaissance man), Yael Netzer (who supervised my first project, is a true artist, and always flowing with great ideas for cool stuff to do), David Gabbay (who thinks so much and speaks so little, but every word counts – the perfect companion for the Haiku Generation work I did with him and Yael) and Raphael Cohen (who joined years later but now I can't imagine the group without him, with witty and clever remarks, a healthy approach to everything, countless fresh perspectives and always willing to hang out and enjoy life) – the experience would not have been nearly as rewarding with any other group of people.

The NLP group was part of BGU's Computer Science department, which is another exceptional group of people. In particular, I spent many rewarding lunches, conversations and coffee breaks with my fellow PhD students Alon Grubshtein and Yair Adato

and Guy Wiener. Moshe Zazon, Nimrod Milo and many others were also around to contribute to the family feeling. As for faculty members, I enjoyed the presence of Mayer Goldberg who supplied many diversions from my own research, constantly reminded me to think differently and find joy in unexpected things, Mike Codish who sat across the hall doing interesting stuff and got me excited (although for a short while) about a subject as seemingly dull as SAT solving, Gera Weiss with whom I had always enjoyable conversations and got numerous advices, Eyal Shimony who was great to work under in the SPLAB course and reminded me of the importance of paying attention to details, Mira Balaban who got me briefly interested in Computer Music and was always around to offer advice, Aryeh Kontorovich who formally answered my always informal questions about machine learning theory, and really, every other member of the CS department.

When taking Michal Ziv-Ukelson's Dynamic Programming course I got acquainted with the RNA folding problem, and chats with Michal and Shay Zakov led me to put natural language on hold for a while and to parse RNA structures instead. Working with Michal and Shay was truly a rewarding experience – many candies, lots of fun, many clever ideas thrown around, some hacking sessions and voila: a best paper award. Can you ask for anything more?

At my first conference I had the good fortune of meeting Reut Tsarfaty, then an enthusiastic PhD student in Amsterdam. We became friends and collaborators, spent a lot of time together and produced some very interesting work. Reut's enthusiasm, knowledge and breadth are truly inspiring, and it is always a pleasure discussing things with her, even if the discussions can get quite animated on both ends. Very few aspects of this work were not influenced in some way or another by my interactions with her. I am certain that we will continue to collaborate, directly or indirectly, for years to come.

When presenting a joint work with Reut (and Meni and David and Michael) in the EACL conference in Athens, I caught the attention of Djame Seddah@, which led to fun dinners and some discussions and many emails and an interesting panel in IWPT and eventually to the formation of the "parsing morphologically rich languages" group (together with Reut Tsarfaty, Jennifer Foster, Marie Candito, Sandra Kubler, Yannick Varsley, Ines Rehbein, Lamia Tounsi and others), the SPMRL workshops, and many more fun, interesting and inspiring chats, travels and dinners. Thanks, Djame, for putting this group together and making this happen! You have done (and still do) a terrific job, and it is a pleasure hanging around and collaborating with you.

I spent the summer of 2010 at Venice Beach, doing an internship with Kevin Knight, David Chiang and Liang Huang at ISI. This was truly a wonderful time spent with amazing hosts, in which I learned about machine translation, syntax, running a suc-

cessful long term project while maintaining focus and paying attention to details while keeping in mind a much broader big-picture and experimenting in various directions, cool t-shirts, aiming high, living in a great weather, thinking big, longboarding and doing stuff that matters while always trying to have just a little more fun. At ISI I also met Eduard Hovy who reminded me about the power of simplicity, and got to know Zornitsa Kozareva, Ashish Vaswani, Sujith Ravi, Steve DeNeefe, Jason Riesa, Ulf Harmjakob, Victoria Fossum and Dirk Hovy who made my time enjoyable and my mind sharper, and sometimes my stomach full with delicious home-made bread. Talking with Ashish got me interested in comics – I am still not sure if this is a good thing or a bad one. My time at ISI would not have been the same without my fellow interns, Sravana Reddy who shares my interest in linguistic creativity and undertaking projects for their coolness-factor, and Sasha Rush who shares my passion for parsing, neat algorithms, and arguably crappy fast-food.

The (extended) Israeli NLP community is smallish, smart, and friendly. I found some friends and had a good time (and many good ideas) traveling and hanging around with Roi Reichart, Oren Tsur, Omri Abend, Shachar Mirkin, Idan Szpektor, Jonathan Berant, Lev-Arie Ratinov, Dan Goldwasser, Yuval Marton, BGU's NLP group and all the others. You guys made conferences fun! Speaking of conferences, I somehow kept finding myself sitting in a hotel lobby on the banquet evening, chatting for hours with Moshe Koppel and Mark Dredze (and occasionally also Meni Adler). You guys also made conferences fun.

Conversations with Joakim Nivre, Mark Johnson, Ryan McDonald, Slav Petrov, Kenji Sagae, Jenny Finkel, Shay Cohen, Julia Hockenmaier, Owen Rambow, Nizar Habash, Dan Klein, Joseph Van-Genabith, Kobby Crammer, Dan Roth, Hal Daume, Jason Eisner, Noah Smith, Ari Rappoport, Shuly Wintner, Ido Dagan and countless other researchers in the ACL community also made me smarter and my work better in many ways.

And of course, I owe thanks to my parents for having no idea what I do but always being certain it is brilliant, and to my amazing wife and life companion Noa who thinks all of this is amazingly boring but who still let it happen and was extremely supportive and always around, when all I could offer was the vague "and some day we could do perfect automatic translation from English to Hebrew". I don't think we will be getting there any time soon, but really hope she stays with me at least until we do, and for many years after that.

Thanks.

List of Figures

2.1 Chunk-structure 10
2.2 Dependency-structure 11
2.3 Projective and Non-projective dependency structures in English 12
2.4 Constituency-tree structure 13
2.5 Head-annotated constituency-tree 14
2.6 Different syntactic structures for “the boy hit the ball with a bat” 16

5.1 Modern Hebrew head-percolation table 48
5.2 Histogram of sentence lengths in the Hebrew Dependency Treebank . . 49

6.1 Parsing using the ArcStandard system 56

7.1 Parsing with the EASYFIRST parser 64
7.2 Feature templates for the easy-first non-directional parser 72

8.1 Edge-labeler features 98

9.1 A Toy Context Free Grammar (CFG) 106
9.2 Three CFG derivations of a given sentence 106

10.1 A layered POS-tag representation 123
10.2 A latent layered POS-tag representation 124
10.3 Lattice representation for the Hebrew sequence בצלם הנעים 128
10.4 Lattice initialization of CKY-chart 129
10.5 Agreement annotation and validation example: correct tree 135
10.6 Agreement annotation and validation example: agreement violation . . 135

List of Tables

4.1 Effect of training set size on dependency-parsing performance in English 27
4.2 Word-level categories (parts of speech) in the Hebrew Treebank 30
4.3 Phrase-level syntactic categories in the Hebrew Treebank 31
4.4 Lexical categories (parts-of-speech tagset) of the Morphological Analyzer 32

5.1 Average number of children in the Hebrew Dependency Treebank . . . 51

7.1 Complexity of different parsing frameworks 77
7.2 Unlabeled dependency accuracy on PTB Section 23 79
7.3 Unlabeled dependency accuracy on CoNLL 2007 English test set 79
7.4 Parser combination with Oracle 80

8.1 Training on gold segmentation and tagging 87
8.2 Training on predicted tagging 87
8.3 Parsing results with generic support for morphological features 88
8.4 EASYFIRST parser accuracies with and without agreement features . . . 95
8.5 Relative contribution of the various agreement cases 95
8.6 MST1 parser performance with Hebrew-specific agreement features . . 96
8.7 Parser-ensemble scores on the dev-set for various parser ensembles . . . 97
8.8 Labeler effectiveness for the various edge labels 99
8.9 Confusion matrix for the edge-labeler 99
8.10 Labeled parsing accuracies on the dev-set 100
8.11 Final EASYFIRST+Morph test-set parsing results 100
8.12 Ensemble-parsing results on the test-set 101

10.1 Out-of-the-box Berkeley-parser performance on the dev-set 117
10.2 Number of learned splits per POS-category after five split-merge cycles 118
10.3 Number of learned splits per NT-category after five split-merge cycles . 119
10.4 Agreement-feature values 133
10.5 Gender and Number percolation rules 136

10.6 Dev-set results when incorporating an external lexicon 138
10.7 Lattice-parsing with external lexicon dev-set results 138
10.8 Agreement-filter dev-set results (gold segmentation) 139
10.9 Agreement-filter dev-set results (lattice-parsing) 139
10.10 Test-set results of the best-performing models 140

Chapter 1

Introduction

Natural language is the primary means of communication between humans. Language is composed of words, which are combined to form sentences, which, in turn, are combined to form larger units such as paragraphs. While single words can convey myriad meanings, it is the combination of words into sentences that allows for efficient communication and the realization of complex ideas. When words are combined to form a sentence, their combination is governed by a set of rules, called the *syntax*, or the *grammar*, of the language. These syntactic rules pose constraints on the ways in which words can be combined ("kid cake the ate a" is clearly *ungrammatical*), and assign meanings to valid configurations of words ("the kid ate a cake" and "the cake ate a kid" are both well formed sentences and contain the exact same words, yet they convey two entirely different messages. The first is an everyday event, while the second is a creepy event from a bizarre horror movie). In other words, sentences must obey structural constraints posed by the syntax, and the structure of the sentence determines its meaning.

In many cases, the sentence structure can be recovered even when the sentence words are unknown: consider "the plumpets ghoked a gloomp" – without knowing what plumpets, ghoked or gloomp are (these are all made-up words), we know that there is an action of "ghoking", there is something called "plumpets" (probably composed of several "plumpet") which is the subject of the ghoking action (which happened in the past) and that there is something called a "gloomp" which is the object of the ghoking action. This is an important property, that is key to our ability to learn and interpret language. Note that while the sentence words in the above example are unknown, a lot of smaller units are known to us and provide useful hints regarding the structure. The *determiners* "the" and "a" indicate that plumpets and gloomp are probably nouns, the "-s" suffix on plumpets suggests plurality, and the "-ed" suffix on ghoked indicates that it is probably a verb in the past tense. The word-order rules of English mandate that subjects appear before a verb and objects after it, completing our interpretation of the

sentence.

Different languages have different syntactic properties. In English, word-order is relatively fixed, while in other languages word order is much more flexible (in Hebrew, the subject may appear either before or after a verb). In languages with a flexible word-order, the meaning of the sentence is realized using other structural elements, like word inflections or markers, which are referred to as *morphology* (in Hebrew, the marker את is used to mark definite objects, distinguishing them from subjects in the same position. In addition, verbs and nouns are marked for Gender and Number, and subject and verb must share the same gender and number). A limited form of morphology also exists in English: the "-s" and "-ed" suffixes in the example above are examples of English morphological markings. In other languages, morphological processes may be much more involved. The lexical units (words) in English are always separated by whitespace. In Chinese, such separation is not available. In Hebrew (and Arabic), most words are separated by whitespace, but many of the function words (determiners like "the", conjunctions such as "and" and prepositions such as "in" or "of") do not stand on their own but are instead attached to the following words.

This thesis is concerned with the automatic syntactic processing of language, *i.e.*, building a computational system that can assign structures to sentences in natural language. This kind of processing is referred to as *parsing*, and the processing system is called *a parser*. Aside from being interesting in its own right, parsing is an important component in many language understanding systems.

The current approach to natural language parsing is dominated by statistical, data-driven methods. Broadly speaking, in these methods one starts with a collection of pairs of natural language sentences and their known syntactic structures (such a collection is called *a Treebank*), and then use machine-learning algorithms that are *trained* on the Treebank data to learn the patterns and regulations that are exhibited in the data. The product of the learning process is a parser (or a *parsing-model*) that can assign structures to new, unseen sentences based on the patterns it learned from the seen, analyzed sentences.

A large part of the parsing literature is devoted to automatic parsing of English, a language with a relatively simple morphology, relatively fixed word order, and a large Treebank. Data-driven English parsing is now at the state where naturally occurring text in the news domain can be automatically parsed with accuracies of around 90% (according to standard parsing evaluation measures). However, when moving from English to languages with richer morphologies and less-rigid word orders, the parsing algorithms developed for English exhibit a large drop in accuracy. In addition, while English has a large Treebank, containing over one-million annotated words, many other languages

have much smaller Treebanks, which also contribute to the drop in the accuracies of the data-driven parsers. A similar drop in parsing accuracy is also exhibited in English, when moving from the news domain on which parsers have traditionally been trained to other genres such as prose, blogs, poetry, product reviews or biomedical texts, which use different vocabularies and to some extent different syntactic rules.

1.1 Main themes

In this thesis I focus on automatic parsing of Modern Hebrew, a semitic language with a rich and productive morphology, relatively free word order[1] and a small Treebank. The primary goal of the thesis is the creation of automatic systems capable of parsing Modern Hebrew text with good accuracy. When pursuing this goal, I concentrate on aspects in which Modern Hebrew differs from English. Several natural questions arise: can the small size of the Treebank be compensated using other available resources or sources of information? Can morphological information be used effectively in order to improve parsing accuracy? How should the word segmentation issue (that function words do not appear in isolation but attach to the next word, forming ambiguous letter patterns) be handled?

Based on these questions, the thesis revolves around the following themes:

1.1.1 Separation of lexical and syntactic knowledge

There are two kinds of knowledge inherent in a parsing system. One of them is *syntactic knowledge* governing the way in which words can be combined to form structures, which, in turn, can be combined to form ever larger structures. The other is *lexical knowledge* about the identities of individual words, the word classes they belong to and the kinds of syntactic structures they can participate in. I argue that the amount of syntactic knowledge needed for a parsing system is relatively limited, and that sufficiently large parts of it can be captured also based on a relatively small Treebank. Lexical knowledge, on the other hand, is much more vast, and we should not rely on a Treebank (small or large) to provide adequate lexical coverage. Instead, we should aim to find ways of integrating lexical knowledge, which is external to the Treebank, into the parsing process.

[1] To be more precise, in Hebrew the order of constituents is relatively free, while the order of the words within certain constituents is relatively fixed.

1.1.2 Using morphological information to improve parsing accuracy

Morphology provides useful hints for resolving syntactic ambiguity, and the parsing model should have a way of utilizing these hints. There is a range of morphological hints than can be utilized: from functional marking elements (such as the את marker indicating a definite direct object), to elements marking syntactic properties such as definiteness (such as the Hebrew ה marker), to agreement patterns requiring a compatibility in properties such as gender, number and person between syntactic constituents (such as a verb and its subject or an adjective and the noun it modifies). The morphological clues provided by the language should be taken into account when designing a parsing model for that language, and the parsing algorithm should be flexible enough to accommodate the possible morphological cues.

1.1.3 Encoding input uncertainty using a lattice-based representation

Sometimes, the language signal (the input to the parser) may be uncertain. This happens in Hebrew when a space-delimited token such as בצל can represent either a single word ("(an) onion") or sequence of two words or three words ("in shaddow" and "in the shadow," respectively). When computationally feasible, it is best to let the uncertainty be resolved by the parser rather than in a separate preprocessing step. One way of encoding uncertainty is using a lattice structure. This idea is explored for Hebrew constituency parsing in Chapter 10.

1.2 Contributions

1.2.1 Parsing systems for Modern Hebrew

The main contribution of this thesis are systems capable of parsing naturally occurring Modern Hebrew text. Specifically, I present:

A Constituency Parser that parses Hebrew sentences into constituency trees with an accuracy of 77%F_1 (85% assuming gold segmentation).

A Fast Dependency Parser that parses Hebrew sentences into dependency trees with an accuracy of 78.5% unlabeled attachment score (83.5 assuming gold segmentation).

An Ensemble-Based Dependency Parser that parses Hebrew sentences into dependency trees with an accuracy of 79.5% unlabeled attachment score (85% assuming gold segmentation).

The parsers are bundled as open-source software packages and are available for download at my website.

1.2.2 Modern Hebrew dependency Treebank

A secondary contribution of the thesis is the creation of Modern Hebrew resources: I created a Modern Hebrew Dependency Treebank. The Treebank is created by conversion from the Modern Hebrew constituency Treebank and is useful for training data-driven dependency parsers, and for performing quantitative linguistic research.

1.2.3 Algorithmic and methodological contributions

The Hebrew parsing systems developed in this work are based on the following algorithmic and methodological contributions:

For dependency parsing:

- I present an efficient ($O(nlogn)$) dependency parsing algorithm that I call the EASYFIRST parser. The algorithm is a greedy, bottom up dependency parser, based on the principle that easy decisions should be made before more difficult ones. The algorithm is data driven, and learns what is easy and hard for it. For Hebrew, the algorithm provides state-of-the-art parsing results. For English, the algorithm rivals the accuracies of much slower parsing algorithms which are based on exhaustive search.[2]

- The parses produced by the EASYFIRST algorithms differ from those produced by other dependency parsing algorithms. This makes the parser a valuable component in parser-ensemble systems.

- The EASYFIRST dependency parsing algorithm can make use of a rich feature set, which I use to encode morphological agreement information. This work is one of the first to show that agreement information significantly improves the accuracy of a parsing system.

[2]Recently, an independent group of researchers [139] showed that using a well crafted feature set the EASYFIRST algorithm not only matches but also substantially improves upon previous state-of-the-art parsing results for English.

For constituency parsing:

- I show that parsing performance can be enhanced by utilizing a language resource external to the Treebank: specifically, a lexicon-based morphological analyzer. I present a computational model of interfacing the external lexicon and a Treebank-based parser, also in the common case where the lexicon and the Treebank follow different annotation schemes.

- I show that Hebrew word-segmentation and constituency-parsing can be performed jointly using CKY lattice parsing. Performing the tasks jointly is effective, and substantially outperforms a pipeline-based model.

- I suggest modeling grammatical agreement in a constituency-based parser as a filter mechanism that is orthogonal to the grammar, and I present a concrete implementation of the method. While the constituency parser does not make many agreement mistakes to begin with, the filter mechanism is effective in fixing the agreement mistakes that the parser does make, without introducing new mistakes.

These contributions extend outside of the scope of Hebrew processing, and are of general applicability to the NLP community. Hebrew is a specific case of a morphologically-rich language, and ideas presented in this thesis are useful also for processing other languages, as well as English. The lattice-based parsing methodology is useful in any case where the input is uncertain. Indeed, we have used it to solve the problem of parsing while recovering null elements in both English and Chinese [20], and others have used it for the joint segmentation and parsing of Arabic [59]. The EASYFIRST parsing algorithm provides state-of-the-art results for English dependency parsing [139], and is currently being successfully adapted to parsing Arabic (Yuval Marton, personal communication). Extending the lexical coverage of a Treebank-derived parser using an external lexicon is relevant for any language with a small Treebank, and also for domain adaptation scenarios for English.

1.3 Structure of this thesis

The thesis comprises three parts. The first part (Chapters 2 to 4) provides background on syntactic representations of language (Chapter 2), the structure of Modern Hebrew (Chapter 3) and computational aspects of automatic processing of Hebrew, including the key challenges and the existing resources (Chapter 4).

The second part (Chapters 5 to 8) is dedicated to dependency parsing. I present the Hebrew Dependency Treebank (Chapter 5), background, evaluation measures, and

previous work on dependency parsing (Chapter 6), the easy-first dependency parsing algorithm and a general-feature set with an evaluation on English text (Chapter 7), and finally a dependency parsing system for Modern Hebrew which is based on the easy-first parsing algorithm and that successfully incorporates morphological agreement information (Chapter 8).

The third part (Chapters 9 and 10) is dedicated to constituency parsing. Chapter 9 surveys some background and previous work in constituency parsing, and Chapter 10 presents extensions to known constituency parsing algorithms with applications to parsing of Modern Hebrew, including a method of interfacing a PCFG-LA Treebank-derived parser with an external and incompatible lexicon, and a method for joint word-segmentation and parsing using lattice parsing.

Part I

Background

Chapter 2

Syntactic Representations of Language

2.1 Syntactic structures

Natural Language sentences are organized according to rules that govern their structure. *Syntax* is the branch of linguistics that studies these rules.

In what follows I describe three forms of syntactic representations: *chunks*, *dependencies*, and *constituency trees*.

Note that a distinction should be made between the *representation type* and the *syntactic content* [109]. The *representation type* defines the formal properties of the structures (what does the set of all possible structures look like) while *syntactic content* adheres to a syntactic theory and states what is the correct structure for a given sentence.

The presentation is aimed at computer scientists, not linguists. I give intuitions, not precise definitions, about the language phenomena that are captured by these representations, and the discussion emphasizes representation type and not syntactic content.

2.1.1 Chunks

One of the most basic structures of text is that of groups of words that belong together, and that can be treated as single units. "Text Chunks", a term first coined by Abney [2], is an attempt to formalize this structure. According to Abney,

> The typical chunk consists of a single content word surrounded by a constellation of function words, matching a fixed template.

A "Text Chunk" is a contiguous group of words that are syntactically related. Chunks loosely correspond to prosodic units [50]. A chunk contains one word, (the *major head*), which is the main word of the chunk, and words that syntactically modify it. The type

of the chunk is determined by the syntactic category (*i.e.*, part-of-speech) of its major head.

As an example, the following is a sentence annotated with chunks structure:

> [NP Confidence] [PP in] [NP the pound] [VP is widely expected] [VP to take] [NP another sharp dive] if [NP trade figures] [PP for] [NP September] , [ADJP due] [PP for] [NP release] [NP tomorrow] [VP fail to show] [NP a substantial improvement] .

Figure 2.1: *Chunk-structure*

As can be seen, the chunks structure of a sentence reveals information about the clustering of contiguous and syntactically related words, and their syntactic type. It does not, however, give information about the relation between different groups of words. For this reason, text chunks are sometimes referred to as *shallow syntactic structure*. Chunks still convey important structural information and are useful for many language analysis tasks. They are also computationally very cheap to produce.

The words in each chunk must be contiguous to each other, and some words do not belong to any chunk. We can think of the words not belonging to any chunk as belonging to a special kind of a chunk. Thus, computationally, chunking is a *sequence segmentation* task, in which a sequence of length N is to be broken into K pieces. Text chunking is usually performed as a sequence tagging task, in which each word is said to be at the **B**eginning, **I**nside or **O**utside of a chunk. I report on automatic syntactic chunking of Hebrew in [52].

2.1.2 Dependency trees

A structure more involved than text chunks is one of *dependencies*. In a dependency structure, each word in a sentence is said to "depend on", or "modify", another word. The main word of the sentence is (usually) a verb, and it modifies an abstract entity called the *root* of the sentence. Dependency structures as described in this work stem from works by Lucien Tesnière [136] and his followers [47, 98].

As an example, the following is a sentence annotated with dependency structure, as shown in Figure 2.2, which is taken from [106].

The ROOT is modified by the main verb ("had"), which is in turn modified by its arguments ("news" and "effect"). The structure then continues – "news" is modified by its type, "Economic", while "effect" is modified by its degree ("little") as well as its argument ("on", modified by "markets"), etc.

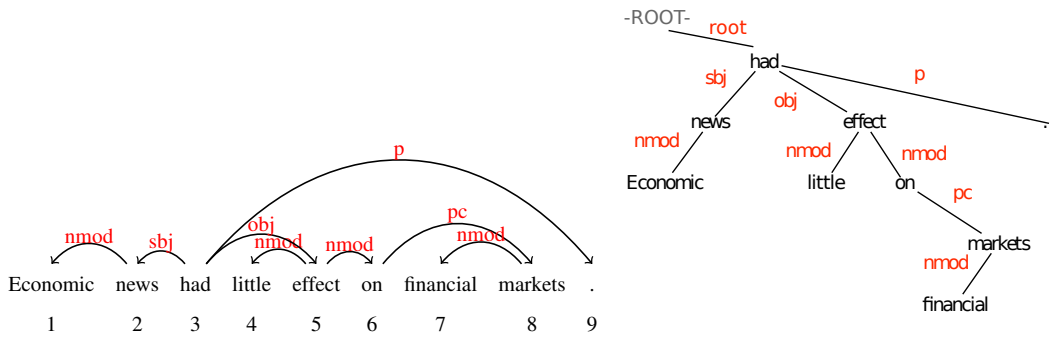


Figure 2.2: *Dependency-structure for the sentence "Economic news had little effect on financial markets .". The two structures are equivalent, but drawn differently.*

One can think of the dependency structure of a sentence as a directed tree spanning the word sequence. The tree is rooted at the root of a sentence, and each sentence word is a node in the tree. Each word has exactly one parent. A dependency structure of a sentence of N words indexed as $w_1, w_2, ..., w_N$ can be uniquely encoded as a sequence of N integers in the range $0, ..., N$, in which the integer at location i specifies the index of the parent of the word w_i, and w_0 represents the special ROOT word. For example, the dependency structure in Figure 2.2 is encoded as $\{2, 3, 0, 5, 3, 5, 8, 6, 3\}$. Another (equivalent) way to represent the dependency structure of a sentence is as a list of N pairs (i, j) in which $i \in 1, N$, $j \in 0, N$, with the constraint that each i appear exactly once. The semantic of a pair (i, j) is that the word w_i is modified by word w_j (alternatively, word w_j is the parent of word w_i).

In addition to the binary dependency relations, the structure may explicitly annotate the function of each dependency: for example, "news" stands in a relation of SBJ (subject) with the verb "had". The representations above can be trivially extended to accommodate these labels by considering an extra sequence of length N in which the item at location i corresponds to the label of the relation between word w_i and its parent.

While dependency structure reveals a wealth of information about the hierarchical relations between words and their functions, it does not explicitly group contiguous related words together. Indeed, word order has little effect in dependency structures.

Projectivity The dependency representation described above allows any directed tree over the words of the sentence. The set of dependency trees can be divided into *projective* and *non-projective* trees. A projective tree over a sentence $s = w_0, w_1, ..., w_n$ is one that can be drawn on a plane above the sentence words without changing the order of the words and without any crossing edges. Alternatively, we can say that a tree is

projective iff an edge from word w_i to w_j (w_i is the parent of w_j) implies that there exists a directed path from w_i to every word between w_i and w_j in the linear order of the sentence. All other trees are said to be non-projective. Figure 2.3 gives examples of projective and non-projective structures.

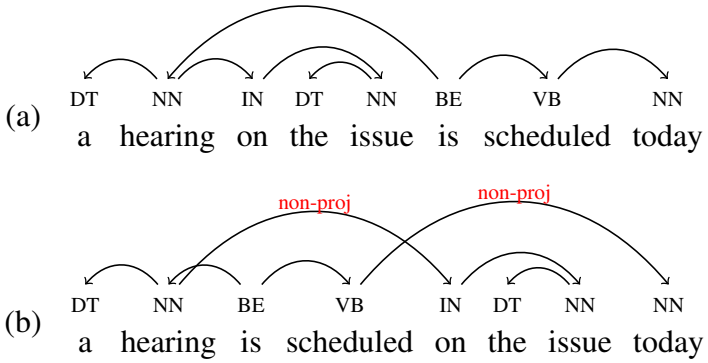


Figure 2.3: *(a) Projective and (b) Non-projective dependency structures in English*

Most natural language sentences under many syntactic theories can be analyzed using projective trees. However, some sentences require a non-projective analysis. Languages vary in the amount of non-projectivity they require. For example, under many syntactic theories English has very few sentences whose structures are non-projective trees, while Czech has many. Even in languages with many non-projective edges, the large majority of edges are projective, or have a restricted form of non-projectivity (see [84] for a discussion). The Hebrew Dependency Treebank (see Chapter 5) does not contain any non-projective structures.[1]

Parsing *algorithms* can also be either projective (allowing only projective dependency trees) or non-projective (allowing the set of all possible dependency trees). The projectivity property provides an effective structuring of the search-space and allows for simple incremental constructions as well as effective polynomial-time dynamic-programming algorithms. See [97] for a discussion on the time-complexity of exact non-projective parsing. I focus on projective parsing in this thesis.

[1]It should be noted, however, that this may be due to the conversion from a constituency Treebank, which is inherently mostly projective. It is possible that in a process of creating a dependency Treebank from scratch by human annotators, a few non-projective cases could be identified. The initial conversion did include about a dozen non-projective cases, but these could all be reconciled into an equivalent projective analysis.

2.1.3 Constituency trees

Constituency, or *Phrase-structure* trees are another very common representation of syntactic structure. In a constituency tree, words are grouped together to form *constituents*, which, in turn, group together to form larger constituents. This yields a hierarchical tree structure, giving rise progressively larger constituents until the complete sentence is spanned.

As an example, the following sentence is annotated with constituency tree structure:

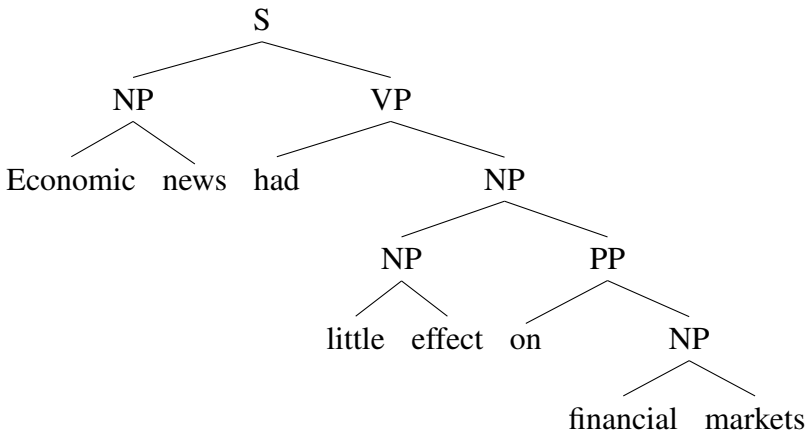


Figure 2.4: *Constituency-tree structure*

In the constituency representation, the sentence words are the leaves of the tree, while the non-leaf nodes of the tree belong to a group of *non-terminal* symbols, that correspond to syntactic categories such as NP (a noun phrase), PP (prepositional phrase), S (a sentence) and so on. This is in contrast to the dependency representation in which all of the tree nodes correspond to words in the sentence.

Notice that the constituency structure reveals information about related groups of words, and the relations between them. One can learn that "Economic news", "little effect", and "financial markets" are all noun-phrases, that "little effect" is connected by the preposition "on" to "financial markets" yielding the bigger noun-phrase "little effect on financial markets" (some of the intermediate level constituents, such as the preposition-phrase "on financial markets" may be somewhat un-natural to non-linguists).

Theories of constituency allow the incorporation of *traces* and *empty-elements*. These are lexical elements (leaves in the syntactic tree) which do not occur in the original sentence, but do occur in its syntactic representation. According to the theory, these elements correspond to lexical materials that were moved, duplicated or elided when moving from the syntactic representation to the sentence form.

While the constituency structure is rich, providing information about word order, grouping of words, and relations between groups of words, one type of information that it does not readily present is the "headedness" – the main word of a constituent. For example, consider the noun-phrase "money transfer": one cannot trivially infer from the constituency tree structure that "money" is the modifier of "transfer" and not the other way around. Another (related) lacking information is argument structure: what are the arguments of the verb?[2]

2.1.4 Constituency to dependency conversion

Heads Constituency-trees can be head-annotated. In a head-annotated constituency tree, each node is associated with a *head word*. The head word of each leaf node is itself. Non-leaf nodes are assigned a head-word from the head-words of their direct children. Figure 2.5 presents an example of a head annotated constituency-tree.

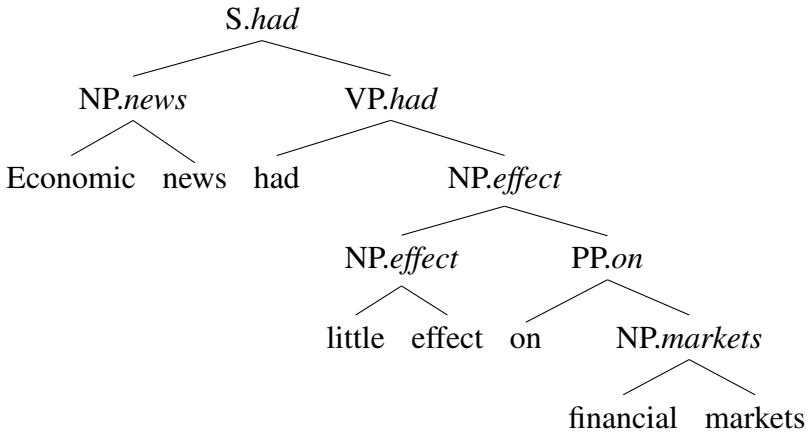


Figure 2.5: *Head-annotated constituency-tree*

Constituency to dependency conversion A head-annotated constituency tree without traces and empty elements uniquely defines a projective dependency tree, and a

[2]Constituency-based linguistic theories use the concept of *configuration* to resolve these issues. Basically, some primitive tree structures are defined, which can then be combined to represent all valid sentence structures of the language. For these primitive structures, it is well defined which branches are the heads, and how the argument structure is assigned. However, a big computational problem with this mechanism is that these linguistic formalisms represent some very common language structure with empty elements and movements – these are branches in the tree whose content is said to have 'moved' to a different location that is then co-indexed with the original location. The original location then has an empty element, which is not shown in the surface form of the sentence. Configurations are then defined over the trees with empty elements and movement information. Current computational models, however, are not successful at restoring such movements and empty elements, leaving the accurate assignment of heads and argument structure an open computational problem.

constituency representation with co-indexed empty elements uniquely defines a possibly non-projective dependency tree. Given a head-annotated constituency tree, a dependency tree is constructed by traversing the nodes and creating dependency links from the head-word of each node to the head-words of its direct children. For example, applying this procedure to the head-annotated constituency tree in Figure 2.5 will result in the following $\langle parent, child \rangle$ dependencies: $\langle$had, news$\rangle$, $\langle$had, effect$\rangle$, $\langle$news, Economic$\rangle$, $\langle$effect, on$\rangle$, $\langle$effect, little$\rangle$, $\langle$on, markets$\rangle$, $\langle$markets, financial$\rangle$. These dependency relations form the dependency structure presented in Figure 2.2.

A constituency tree can be converted to an (unlabeled) dependency tree by head-annotating the constituency tree – deciding which word is the main word of each constituent. Designing an automated head-assignment process that results in dependency structures following a given syntactic theory or guidelines is not trivial, but depending on the syntactic theories that are responsible for the dependency and constituency tree structures, it is possible in many cases. Such automatic head-assignment processes are based on a set of rules that specify the head percolation from the leaf. The rules are typically local in the tree structure. An example of such rule is "if the node is an S and has a VP child, the head of the VP is also the head of the S". Various constituency to dependency conversions for English are available (see, *e.g.*, [33, p. 238-240] [39, 72, 139]). A constituency to dependency scheme for Hebrew is discussed in Chapter 5.

2.2 Syntactic disambiguation

A typical natural language sentence admits many possible syntactic analyses. There are at least $(2k)^n$ different ways to chunk a given sentence (n being the number of words in the sentence, k being the number of distinct chunk types) and an exponential number of ways to construct a dependency or constituency tree as well. The process of choosing the correct syntactic structure for a given sentence is called *syntactic disambiguation*.

A wrong syntactic structure for a sentence can be either illegal (not following the rules of the underlying syntactic theory), legal but semantically wrong, or legal but wrong. Figure 2.6 presents 4 different constituency syntactic analyses for the sentence "the boy hit the ball with a bat". (2.6a) presents an illegal analysis, (2.6b) is a legal but semantically wrong analysis, in which the boy is assumed to hit a ball that has a bat, and (2.6c) is a legal but semantically wrong analysis in which a boy and a bat are assumed to be hitting a ball. Finally, (2.6d) presents the correct structure. Notice that in a hypothetical world in which bats and humans can be friends, (2.6c) can be either correct, or legal but wrong, depending on the situation the sentence is intended to describe (*e.g.*, a boy and a bat hitting a ball, or a boy hitting a ball using a bat). In the

case (2.6c) is correct, (2.6d) is wrong.

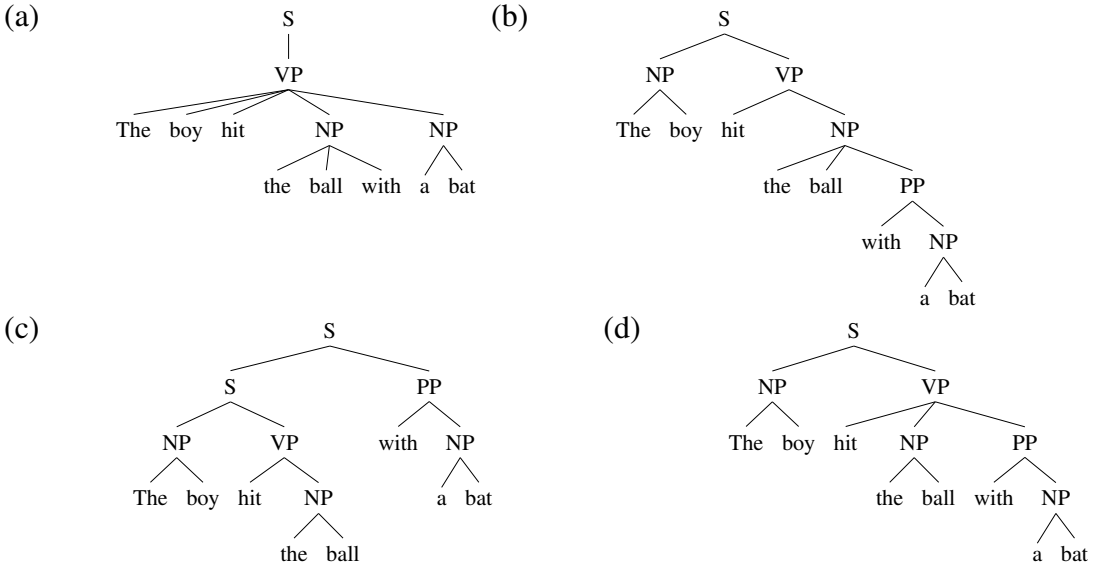


Figure 2.6: *Different syntactic structures for "the boy hit the ball with a bat"*

The role of a syntactic parser is to perform syntactic disambiguation: first discarding any illegal analyses for a given sentence, and then choosing the correct analyses from among the legal ones.

The syntactic ambiguity in Figure 2.6 belong to the class of "PP attachment" ambiguities. These ambiguities are very common, and are caused by the fact that prepositional phrases (PPs), such as "with a bat" or "on the hill", can attach to either verbs, nouns or other sentences.

For example, in the sentence "I ate pizza with a fork", the PP "with a fork" attaches to the verb "ate" (the eating is done with a fork) while in the sentence "I ate pizza with mushrooms" the PP "with mushrooms" attaches to the noun "pizza" (the pizza is with mushrooms).

Another common cause of syntactic ambiguity is the coordinating conjunctions (*e.g.*, the words "and", "or", "but"). Consider the sentence "I ate tasty apples and bananas": did I eat apples which were tasty and some plain bananas, or were the bananas also tasty? In contrast, the sentence "I ate red apples and bananas" is less ambiguous: it is clear to most readers that the adjective "red" refers only to the apples, as bananas are usually not red. The disambiguation is performed using semantic knowledge. The sentence "I ate apples and red bananas" is unambiguous: here the syntax allows only one reading, in which the bananas are red – syntax is stronger than semantics in this case.

A more advanced syntactic phenomena causing ambiguity is that of *control* . Consider the sentence pair:

- I convinced her to eat it.
- I promised her to eat it.

While the two sentences seem very similar, in the first, the "eating" will be performed by her, but in the second, the "eating" will be done by myself. Note that these sentence share **the exact same** constituency structure, and both verbs share the same subcategorization frames (they both expect an animate noun-phrase and a to-infinitive clause). This type of construction is further evidence that extracting argument structure from constituency structures is in no way a trivial task.

2.3 Grammar-based vs. data-driven parsers

Grammar-based parsers are based on a grammar created by humans. The grammar is a large set of rules that specify, for example, which verbs take what kind of arguments, which adjectives are likely to modify which nouns, that a certain verb is more likely to be the main verb of a sentence than another verb, etc. There are typically tens of thousands of such rules in a grammar that aims at providing broad coverage (*i.e.,* aims to be able to assign structure to as many different sentences as possible). For example, LinkGrammar [130], a dependency-like grammar for English, consists of about 800 formulas (high-order rules) in addition to word-level rules for about 60,000 words. Such grammars are difficult to create, and even harder to maintain and extend.

Data Driven dependency parsers, on the other hand, rely on human annotated examples instead of a grammar. Such parsers use the annotated examples as learning material. Learning is usually performed using statistical pattern recognition and machine learning methods, which result in a statistical model of language. The model is then used by the parser in order to assign structure to novel unseen examples. In this sense, one can say that data driven parsers learn their grammar from the annotated data.

Data driven methods are increasing in popularity over grammar-based methods. This can be attributed to the following reasons:

Efficiency of Creation Annotating sentences is easier than creating a grammar, even if only for the fact that the annotation process for each sentence is independent, while every addition or change to the grammar has the potential to affect all previously made choices [18].

Reusability Once a collection of annotated sentences is created, it can be used to train different systems, providing gradual improvement. The annotated sentences can even be used by human grammar designers to build better grammars.

Performance In recent years, data driven methods outperform grammar-based ones in terms of overall accuracy.

I concentrate on data driven methods in this work.

2.4 Applications of syntactic structure

Knowing the syntactic structure of a sentence is a building block for many language related tasks. Here I review some higher level applications and their use of syntactic structure as a building block.

Text simplification Simplification is the task of simplifying a given sentence so it will be easier to read – for example, for younger, or non-native, users of the language [24,70,77,91]. Text simplification can take place at the lexical level, in which rare words are replaced by less sophisticated, more common synonyms, and at the syntactic level, in which long and complicated modifiers or clauses are dropped from the sentence, all the while preserving its essence. For example, in the extreme case, the sentence "Economic news had little effect on financial markets" can be simplified to "news had effect on markets". Dependency structure allows us to do this kind of simplification by just pruning the tree with some constraints. Of course, this simplified version, while syntactically correct, is missing some important information. Another component is needed to intelligently select which parts of the sentence to drop and which to keep. [91] uses dependency structure as the basis of syntactic text simplification. His system is used to simplify sentences such as "The source code , which is available for C , Fortran , ADA and VHDL , can be compiled and executed on the same system or ported to other target platforms" to "The source code can be compiled and executed on the same system or ported to other target platforms". This view of simplification is rather simplistic, requiring the word order and general structure of the sentence to remain fixed. One can envision more involved processes of text simplification in which sentences are paraphrased to convey the same meaning in a simpler form, while allowing more elaborate syntactic transformations, such as rewriting passive constructions as active and undoing topicalizations in order to maintain a more canonical sentence structure (*i.e.,* rewriting "the company was sold by its owner, the newspapers reported" as "the newspapers reported that the owner sold his company"). These kinds of transformation also require access to the syntactic structure of the sentence to be simplified.

Lexical acquisition This broad task is about inference of lexically oriented linguistic knowledge, *i.e.,* knowledge about the vocabulary of the language. Subtasks of lexical

acquisition range from learning word classes (*e.g.,* to which parts-of-speech does the word 'dog' belong), to learning taxonomic relations such as the hyponymy between words (*e.g.,* learning that "a dog is a kind of canine", "a desk is a kind of furniture" and so on), to learning subcategorization frames for verbs (*e.g.,* that the verb "think" needs to bond with a clause: "think that ...", while the verb "eat" bonds to a noun phrase: "I eat food"). Many research efforts have tackled this challenging task. For example, [124] learn Czech subcategorization frames from Czech sentences annotated with dependency structure, and [132] use a learning algorithm and an English dependency parser to extract hypernym/hyponym relations from a large, unannotated text corpus.

Information extraction (IE) IE [110] refers to selectively extracting meaning from natural language text. "Meaning" in this context refers to a set of *entities*, a fixed set of *relations* between entities, and *events* in which entities participate. For example, one track of the sixth MUC (Message Understanding Conference) was about extracting "management succession" information: identification of events in which corporate managers left their posts or assumed new ones. Most information extraction systems rely on (fuzzy) pattern matching techniques. While earlier systems relied on lexical patterns, newer systems (see, for example, [152]) incorporate also syntactic-dependency-based patterns.

Text summarization [71] – similar to the text simplification task but extending it, this task is about automatically generating a summary for a given document. The process involves finding the key sentences, determining the redundant information in these sentences, extracting phrases that contain important information, and recombining these phrases into fluent, natural language text. This is a complex task that relies on multiple components, several of them using dependency structure in their input.

Multi-document summarization [15] – an extension to text summarization, the goal of this task is to generate a readable natural language summary from multiple text documents describing the same event. Again, a dependency parser is used in many of the system components. Moreover, the task of deciding which documents describe the same event, also makes extensive use of dependency structure in text (as well as an Information Extraction component) [62].

Machine-translation [78] – the goal of machine translation is to generate automatic translations from one language to another. When translating, one can use syntax either on the source side (looking at the syntactic structures of the sentences one is attempting

to translate) [64, 86], on the target side (making sure that the translated version of the sentence is well formed and adheres to syntactic constraints) [48,49], both on the source and the target sides [150], or not using syntax at all and working on the word level and relying on a strong sequential language model components [79]. The syntax-less approach is very popular and produces strong results. Recently, syntax-based systems started to gain justified popularity – [65] shows how using syntax and the source side can greatly increase the translation speed while preserving quality, and systems such as [149] use target side syntax and produce the top-rated translations in various academic competitions. Most research on translation, however, concentrate on translating into English, a language with a fairly fixed word order and simple morphology. But when such systems are used for automatic translation out of English and into languages with freer word orders and richer morphologies, the results are of much lower quality, and often include ungrammatical, amusing or unreadable sentences. Effectively using syntax on the target side when translating into such languages is a fertile research direction that goes hand in hand with research into accurately parsing such languages and having good models of morphology and its interaction with syntax.

Chapter 3

Modern Hebrew

This chapter describes elements of Modern Hebrew which are related to the syntactic-parsing task.

3.1 Lexical level

3.1.1 Unvocalized orthography

Most vowels are not marked in everyday Hebrew text, which results in a very high level of lexical and morphological ambiguity. Some tokens can admit as many as 15 distinct readings, and the average number of possible morphological analyses per token in Hebrew text is 2.7, compared to 1.4 in English [4]. This means that on average, every token is ambiguous with respect to its part-of-speech and morphological properties. For example, the word כפיות can be read in at least 8 different ways (“spoons”, “square cotton headkerchiefs”, “coercions”, “as mouths”, “as spouts”, “as fairies”, “ungratefulness”, “fun/adjective (feminine,plural)”), the word כתב in at least 6 ways (“a journalist”, “writing”, “script”, “wrote”, “added someone as a recipient”, “was added as a recipient”) and the word את can be read as a very common case-marker (appearing before definite direct objects), a very common pronoun (“you/$_{\text{feminine}}$”) and a noun (“shovel”).

3.1.2 Affixation

Eight common prepositions, conjunctions and articles may never appear in isolation and must always be attached as prefixes to the following word. These include the function words מ (“from”), ש (“which”/“who”/“that”), כש (“when”), ה (“the”), ו (“and”), כ (“like”), ל (“to”) and ב (“in”),. Several such elements may attach together, producing forms such as ושמהשמש (ו-ש-מ-ה-שמש “and-that-from-the-sun”). Notice that when it ap-

pears by itself, the last part of the token, the noun שמש ("sun"), can be also interpreted as the sequence ש־מש ("who moved"). The linear order of such elements within a token is fixed (disallowing the reading ו־ש־מ־ה־ש־מש in the previous example).

However, the syntactic relations of these elements with respect to the rest of the sentence is rather free. The relativizer ש ("that"), for example, may attach to an arbitrarily long relative clause that goes beyond token boundaries. The attachment in such cases encompasses a long-distance dependency that cannot be captured by local-context (or Markovian) sequential processes that are typically used for morphological disambiguation. The same argument holds for resolving PP attachment of a prefixed preposition or marking conjunction of elements of any kind.

To further complicate matters, the definite article ה ("the") is not realized in writing when following the particles ב ("in"), כ ("like") and ל ("to"). Thus, the form בבית can be interpreted as either ב־בית ("in house") or ב־ה־בית ("in the house").[1]

In addition, pronominal elements (clitics) may attach to nouns, verbs, adverbs, prepositions and others as suffixes (e.g. הביאן (הביא־הן, "brought-them"), עליהם (עלי־הם,"on them")).

These affixations result in highly ambiguous token segmentations: מספרו ("(they) assigned numbers") vs. מספר־ו ("his number" or "the one who cuts his hair") vs. מ־ספר־ו ("from his book" or "from his barber"), הרכבת ("putting together") vs. ה־רכבת ("the train") and בצל ("an onion") vs. ב־צל ("in the shadow") are only a few examples of ambiguities that may arise. Quantitatively, 99,896 out of 567,483 forms in a wide-coverage lexicon of Hebrew can admit both segmented and unsegmented analyses.

In many cases the correct segmentation cannot be determined from local context alone, but can be disambiguated by more global syntactic constraints (in ראיתי שמים כחולים, the middle token is ambiguous between שמים("sky") and ש־מים("that/$_{rel}$ water"), and the sequence can be interpreted as either "I saw blue skies" or "I saw that blue water". On the other hand, ראיתי שמים כחולים פרצו מן הבאר is unambiguous because the past verb פרצו requires the relativizer ש, allowing only the segmented ש־מים reading "I saw that blue water broke from the well". In the other direction, ראיתי שמים כחולים והלכתי לישון is also unambiguous, allowing only the unsegmented reading "I saw blue skies and went to sleep".)

3.1.3 Rich templatic morphology

Hebrew words follow a complex morphological structure, which is based on a root + template system, with both derivational and inflectional elements. Word forms can en-

[1]This overt element is in fact indicated by vocalization, but is not realized in standard written text.

code gender, number, person and tense, and in addition noun-compounding is also morphologically marked (see *construct-state* below). While the exact details of the system are irrelevant (but see [4, 51] for a good overview), we note that this word formation mechanism results in a very high number of possible word forms, and that it is hard to guess the part-of-speech of words based on prefixes and suffixes alone, a method frequently used in other languages.

3.1.4 The participle form

The Hebrew participle form (בינוני, literally the "middle form" of verbs) is a form that shares morphological and syntactic properties of nouns, verbs and adjectives. This form causes many disagreements between human annotators, and large disagreement is found also between major Hebrew dictionaries regarding many word forms (see [7] for a discussion from tagset design and annotation guidelines, including many syntactic, semantic and lexical considerations). For the purpose of this thesis, this form is of interest as it highlights the inherent ambiguity between adjectival, nominal, and verbal readings of many words, which are hard to disambiguate even in context.

3.2 Syntactic level

3.2.1 Relatively free constituent order

The ordering of constituents inside a phrase is relatively free. This is most notably apparent in verbal phrases and sentential levels. In particular, while most sentences follow a subject-verb-object order (SVO), OVS and VSO configurations are also possible (counting in the Hebrew Treebank reveals 5,720 SV cases and 2,613 VS cases, compared to 81,135 SV and 3,018 VS constructions in the English WSJ Treebank). In addition, verbal arguments can appear before or after the verb, and in many orders. For example, the message "went from Israel to Thailand" can be expressed as "went to Thailand from Israel", "to Thailand went from Israel", "from Israel went to Thailand", "from Israel to Thailand went" and "to Thailand from Israel went". Such variations in constituent order are easy to capture using "flat" S structures putting the verbs and all of its arguments on the same clausal level, and this is the annotation approach adopted by the Hebrew Treebank (as well as by Treebanks of other languages, such as French [1]). However, these flat structures result in the grammar having more and longer rules and the Treebank having fewer instances of each rule type, causing a data sparseness problems for statistical estimation methods based on Treebank counts, and making it

more difficult to reliably estimate the grammar parameters.

3.2.2 Verbless constructions

Several constructions in which the verb is not realized are common in Hebrew. These include the possessive constructions such as לעידו צעצועים רבים ("to-Ido toys many" meaning "ido *has* many toys") which also feature a flexible constituent order צעצועים רבים לעידו ("toys many to-Ido", "ido *has* many toys"), and copular constructions such as הילד חמוד ("the-boy cute" "the boy is cute") and הילד משוגע ("the-boy crazy" "the boy *is* crazy")

3.2.3 NP structure and construct-state

While constituents order may vary, NP internal structure is rigid. A special morphological marker (*Construct State*, or סמיכות) is used to mark noun-compounds as well as similar phenomena (this is similar to the *idafa* construction in Arabic).[2] Noun compounding in Modern Hebrew is productive and very frequent – about a quarter of the noun tokens in the Hebrew Treebank are in the construct-state. Construct-state nouns can be highly ambiguous with regular nouns. Some forms are morphologically marked but the marking is not present in unvocalized text (בנות/banot vs. בנות/bnot), while some forms are not marked at all (עורך). The construct-state marker, while ambiguous, is essential for analyzing NP internal structure. While regular nouns are marked for definiteness using the definite marker ה, construct-nouns acquire the definite status of the noun-phrase they compound to. Construct constructions may be nested, as in גוון צבע מכסה קופסת התפוחים ("$\text{shade}_{\text{const}}$ $\text{color}_{\text{const}}$ $\text{lid}_{\text{const}}$ $\text{box}_{\text{const}}$ the apples" meaning "the shade of the color of the lid of the box of the apples").

3.2.4 Definiteness

Definiteness is spread across many elements in the NP. All elements in a definite NP, except for construct-nouns and proper-names, are explicitly marked using the functional element ה which is prefixed to the token. Proper-names are inherently definite and cannot take the definite marker, and construct-nouns acquire their definiteness status from the NP they dominate (definiteness is not explicitly marked on construct-nouns). The definiteness system interacts with verbless constructions: a definite NP followed by an indefinite ADJP is interpreted as a predicative phrase (*e.g.*, ה ילד ה גבוה "the-boy

[2]The construct state is not restricted to nouns, and can also appear on numbers (*e.g.*, עשרות ילדים) and adjectives (גדול הסופרים).

the tall" is a definite NP meaning "the tall boy" while הילד גבוה "the-boy tall" is the predicative phrase meaning "the boy is tall").

3.2.5 Case marking

Definite direct objects are marked. The case marker in this case is the function word את appearing before the direct object. Subjects, indirect objects and non-definite direct objects are not marked.

3.2.6 Agreement

Hebrew grammar forces morphological agreement between adjectives and nominals (adjectives appear after the noun, and agree in gender, number and definiteness), and between subjects and verbs (including the verb-less copular constructions) which agree in gender, number and person. Agreement in the predicative case is a bit complex: when the verb is overt and the predicative-complement is a noun, as in הנסיעה היא תירוץ ("the-trip$_{\text{fem}}$ is$_{\text{fem}}$ an-excuse$_{\text{masc}}$"), gender and number agreement are required between the subject and the verb (but not the predicative-complement), but in the verbless case, the subject and the predicate-complement noun must agree (*הנסיעה תירוץ "the-trip$_{\text{fem}}$ an-excuse$_{\text{masc}}$"). When the predicate-complement is an adjective, gender and number agreement between the subject and the predicate-complement is required regardless of the realization of the verb/copular element: הילד גבוה, *הילדה גבוה, הילד הוא גבוה, הילדה היא גבוה* ("the-boy tall$_{\text{masc}}$", "*the-boy tall$_{\text{fem}}$", "the-boy is$_{\text{masc}}$ tall$_{\text{masc}}$", "*the-girl is$_{textfem}$ tall$_{\text{masc}}$").

Chapter 4

Automatic Syntactic Disambiguation in Modern Hebrew

4.1 Aspects of Modern Hebrew parsing

Syntactic parsing has advanced a lot in the past two decades, resulting in parsing systems that can automatically parse English text with remarkable accuracies. However, these parsing systems were designed with English in mind, and while many of them can be applied to a different language with minimal changes, doing so results in a large drop in accuracy. This section highlights some aspects in which Modern Hebrew differs from English from the perspective of parsing system design.

4.1.1 Small amount of annotated data

While the English Treebank is relatively lage (49,208 sentences, or 1,173,766 words), the Hebrew Treebank [107] is much smaller, containing only 6,220 sentences, or 115,661 tokens (156,316 words[1]).

The small size of the Hebrew Treebank implies a smaller training set for learning-algorithms used to construct the parser. To quantify the effect of the Treebank size used for training on parser performance, I compare the performance on two popular data-driven dependency parsers[2] for English, once when trained using a large training set of 39,832 sentences, and once when trained using a smaller training set of 5,200 sentences, as available in Hebrew. Table 4.1 presents the results. The decrease in the number of sentences used for training the parser causes the parsing performance to drop by as

[1] Because of agglutination, a Hebrew Token may consist of several words, for example the token "בבית" comprises the two words "ב"(in) and "בית"(house).

[2] I use the MALT and MST1 parsers, as described in Chapter 7.

Training Size (# sents)	MALTAccuracy (UAS)	MST1 Accuracy (UAS)
39,832	88.40	90.00
5,200	84.85	86.30

Table 4.1: *Effect of training set size on dependency-parsing performance in English*

much as 3.7 accuracy points[3], indicating that the training set size plays an important role in parsing performance.

4.1.2 Ambiguous word segmentation

Syntactic parsing systems treat the input sentence as observed data – the leaves (in constituency parsing) or nodes (in dependency parsing) of the tree are known in advance, and the parser is expected to build a parse tree around them. This is not the case in Hebrew. As discussed in Chapter 3.1.2, many function words in Hebrew are not separated by whitespace but instead are prefixed to the next word and appear within the same token. This prefixation process results in ambiguous tokens: a single token may correspond to one or more input words. The token "שמשה", for example, can be interpreted as "שמשה", "ש-משה" and "ש-מ-שה". This makes the word sequence unobserved to the parser, which has to infer both the syntactic-structure and the *token segmentation*.[4]

One possible solution to the unobserved word-sequence problem is a pipeline system in which an initial model is in charge of token-segmentation, and the output of the initial model is fed as the input to a second stage parser. This is a popular approach in parsing systems for Arabic and Chinese [59, 69], and it is also the approach I use in Chapter 8. However, as discussed in section 3.1.2 (as well as in [57, 140]), the token-segmentation and syntactic-parsing tasks are closely intertwined and are better performed jointly instead of in a pipeline fashion. This approach is explored in Chapter 10.

4.1.3 Morphological variation and high out-of-vocabulary rate

The intrinsic deficiency caused by the small amount of training data is made even more severe due to Hebrew's rich morphological inflection patterns. The high amount of morphological variation means that many word-forms will not be observed in the training data, making it harder to reliably estimate lexical probabilities based on the annotated resources alone. Out of 3,923 unique tokens[5] in the development-set of the

[3] Evaluation metrics are defined in section 6.1

[4] *token segmentation* is sometimes (erroneously) referred to as *morphological segmentation*.

[5] Tokens in this context are lexical units before word-segmentation, but without punctuation marks already separated as different tokens.

Hebrew Treebank (480 sentences), 1467 tokens (more than a third) are never observed in the training set (5700 sentences, 25,629 unique tokens), 471 tokens are observed once, 344 are observed twice, only 1218 are observed 5 times or more and only 758 tokens are observed 10 times or more. When measuring at the word level (assuming we magically know the correct word segmentation), the coverage is better, but still low: out of 3,115 unique words in the development set, 826 are not observed in the 16,485 unique words of the training set, 335 are observed once, 271 are observed twice, 1359 are observed 5 times or more and 906 words are observed 10 times or more. Very few unique words are seen enough times in the training data for a statistical algorithm to learn robust estimates about their behavior, especially in the non-segmented case. For the segmented case, the numbers are similar to (though still somewhat lower than) the numbers for English on a comparable-size corpus (5700 training sentences, 480 development sentences, 3,015 unique words in the development set, of which 773 are never observed in training, and 1528 are observed 5 times or more). This indicates the importance of correct word-segmentation.

In addition, the case for English is much easier, because parts-of-speech for words are relatively easy to guess based on simple orthographic features (words starting with a capital letters are proper-nouns, words ending in -ed are usually verbs, and so on), while this is not the case for Hebrew. From among the 773 unobserved words for English, 269 start with a capital letter, 58 end with "ed", and 49 end with "ing". Together, these three simple heuristics cover almost half of the unobserved tokens. Such heuristics are not available for Hebrew in the common case of unvocalized text (לא מנוקד, without diacritic marks): proper names are not marked in writing, and word prefixes and suffixes are not indicative of the part-of-speech tags.[6] Thus, the out-of-vocabulary (OOV) problem is much harder in Hebrew than in English and other European languages: on the one hand many words are unobserved in training, and on the other, it is more difficult to guess the analyses of such unknown words.

A system for handling automatic processing of Hebrew text cannot rely solely on manually annotated corpora, as such corpora cannot provide adequate lexical coverage. Systems which attempt to perform disambiguation on the lexical level (such as sequence-based morphological disambiguators, or syntactic parsers that perform morphological disambiguation as part of the parsing process) should be designed to incorporate lexical knowledge from sources external to the annotated corpora. I discuss

[6]While the suffixes are good indicators of gender and number (ים is usually plural masculine, ה is usually singular feminine), they are not good at indicating the core part-of-speech (ה is a suffix can appear in adjectives יפה, verbs שמרה, nouns מחלה, and similarly for ים (יפים, מתחפשים, מנעולים). Furthermore, due to the root+template system, in most cases the first and last letters of the word are part of the root and not of the pattern שומר,ישמור, making the suffixes even less indicative.

methods of enhancing the system's performance based on resources that are external to the Treebank: either a lexicon-based unsupervised part-of-speech tagger trained on a large amount of unannotated text, or a manually created, robust morphological analyzer.

4.1.4 Morphological agreement

The rich morphological system also means that words carry a large amounts of extra information: definiteness, gender, number, tense, and person. Some of this information interacts with syntax through *agreement constraints* . Specifically, nouns and adjectives should agree in gender and number, and subjects and verbs should agree in gender, number, and person. Agreement constraints can provide useful hints for disambiguating the syntactic structure. Consider for example the sentence אשת האדם שאכלה את התפוח ("wife of the man who ate the apple"). The English sentence is ambiguous with respect to the entity that ate the apple, but the Hebrew version is not – the verb אכלה/ate is in feminine form, indicating that it was the wife who did the eating.

Can a parsing system make use of such information? I investigate this issue further in Chapters 8 and 10.

4.2 Existing resources for Hebrew text processing

Several linguistic resources are available for Hebrew, and are used as building blocks for the parsing systems described in this thesis.

4.2.1 The Hebrew constituency-Treebank

A Treebank is a collection of sentences manually annotated with syntactic structure. A constituency Treebank of Modern Hebrew, incrementally developed at the Technion over the course of more than eight years [61, 129], is maintained by MILA, the knowledge center for processing Hebrew[7]. The current version of the Treebank (Version 2) contains 6220 sentences taken from the Israeli daily newspaper הארץ ("Ha'aretz").[8] The sentences are manually annotated on both the lexical and the syntactic levels. Each token is segmented into words, and each word is assigned a part of speech tag which also captures, where applicable, the morphological properties of the word such as number, gender, and person. Then a constituency tree is built on top of the segmented words.

[7] `http://www.mila.cs.technion.ac.il/mila/eng/index.html`

[8] The official release of the Treebank reports 6500 sentences. After removing empty, duplicate and malformed trees, 6220 sentences remain.

The annotation of NPs is relatively nested, while the sentence level structures are relatively flat (the verb and all of its arguments reside on one level under S). The Treebank has 115,661 tokens and 156,764 words. The average sentence length is 18.6 tokens (25.2 words).

The POS tagging scheme in the Treebank is highly syntactic in nature: a part-of-speech is chosen to reflect the syntactic function of the given word in context. For example, demonstrative pronouns are tagged in the Treebank as adjectives when appearing in an adjectival position ("ילד זה", "this/JJ child/NN"), and a special MOD tag is used to mark non-adverbial clausal level modification (that is, modifications that can be treated as adverbial, but that are used to modify something other than a verb). Table 4.2 lists the word-level categories annotated in the Treebank, and Table 4.3 lists the syntactic categories. For a more detailed description of the Constituency Treebank see [107, 129],[142, p. 199-216], as well as the annotation guideline[9].

Category	Description	Example
NN	Noun	מתנדבים, וועדה
NNP	Proper Noun	דני, צים, תאילנד
NNT	Construct-state Noun	מנות משלוח, הממשלה הרכבת, וועדת
PRP	Personal Pronoun	אני, הן, אתם
JJ	Adjective	יפה, חכמה, חזקים
JJT	Construct-state Adjective	רחמים חסרת
CD	Number	עשר, 231
CDT	Number in determiner position	אנשים עשרות
VB	Finite Verb	אכלו, הלך
MD	Modal Verb	אסור, רצוי
VB-INFINITIVE	Infinitive Verb	לאכול, ללכת
RB	Adverb	מהר, בעוצמה
RBR	Comparative Adverb	יותר, פחות
MOD	Non-adverbial Modification	גם, כמעט, בדיוק, פוסט
IN	Preposition	ב, לפני
AUX	Auxiliary verbs (past and future tenses)	היה, תהיה
AGR	Copular-type agreement (present tense)	הן, הם, היא, הוא
REL	Relativizer	ש, אשר
POS	Possessive	של
CC	Coordinating Conjunction	ו
DT	Determiner	כל
WDT	Determiner Question Word	כמה, איזו
QW	Question/WH Word	למה, מה
HAM	The Y/N question word האם	האם
AT	Accusativity Marker	את
H	H definite marker	ה

Table 4.2: *Word-level categories (parts of speech) in the Hebrew Treebank*

Train/dev/test splits Throughout the thesis, I follow the established train/dev/test split for the Treebank, namely sentences 1-483 (0-500 in the original Treebank) are used for development, sentences 484-5740 are used for training the parser, and sentences 5741 to 6220 are used as the final test set.

[9] http://www.mila.cs.technion.ac.il/mila/files/treebank/Decisions-Corpus1-5001.v1.pdf

Category	Description
ADJP	Adjective phrase
ADVP	Adverb phrase
FRAG	Fragment of a declarative sentence
FRAGQ	Fragment of an interrogative sentence
INTJ	Interjection
NP	Noun phrase
PP	Preposition phrase
PREDP	Predicate phrase
PRN	Parenthetical
S	Declarative sentence
SBAR	Clause introduced by a REL or IN word
SQ	Interrogative sentence
VP	Verb phrase

Table 4.3: *Phrase-level syntactic categories in the Hebrew Treebank*

4.2.2 The MILA broad-coverage lexicon

Aside from the Constituency Treebank, Hebrew has a wide-coverage, lexicon-based morphological analyzer which can assign morphological analyses (prefixes, suffixes, core POS, gender, number, person, etc.) to Hebrew tokens. The lexicon (henceforth *the KC Analyzer*) is developed and maintained by the Knowledge Center for Processing Hebrew [68]. It is based on a lexicon of roughly 25,000 word lemmas and their inflection patterns. From these, 562,439 unique word forms are derived. These are then prefixed (subject to constraints) by 73 prepositional prefixes. Even with this seemingly large vocabulary, the KC Analyzer's coverage is not perfect. In [6], we present a machine-learning method that is trained on the basis of the analyzer and that can guess possible analyses for words unknown to the analyzer with reasonable accuracies. Using this extension, the analyzer has perfect coverage (even though the quality is obviously better for words that are present in the analyzer's database).

The tagset used by the lexicon/analyzer is lexicographic in nature, and is discussed in depth in [60]. The tagset is presented in Table 4.4. The table lists the core tags and the additional properties (morphological information) available for each tag, and follows the naming scheme used in the Hebrew Dependency Treebank and throughout this thesis.

Creating a resource such as the morphological analyzer for a morphologically-rich language is a worthwhile and cost-effective effort: after establishing the tagset, it is relatively straightforward to add lemmas to the lexicon, and the automatic inflection process guarantees good coverage of all the possible inflections. This is much more efficient than annotating enough text to obtain a similar coverage. Furthermore, it is shown in [4, 53] that, coupled with a large amount of raw text, such an analyzer can be used to train an effective POS-tagger. In Chapter 10, I use the lexicon to improve the accuracy of a constituency parser.

NN	Noun	ארץ, יום, ממשלה, משטרה
NNP	Proper Nouns	אביב, כהנא, ירושלים, ישראל
NNT	Construct-state nouns	וודעת, ידי
NN_S_PP	Noun with a possessive suffix	שמו, סופו, חייו, דבריו, מותו
BN	Beinoni (participle) form	מדובר, נוגע, נוסף, קשור
BN_S_PP	Beinoni Form with a possessive suffix	יושביה
BNT	Construct-state Beinoni Form	מוכי, מנחילי, אוזלת, מרבה, מטבע
VB-TOINFINITIVE	Infinitive Verbs	לבצע, לתת, למנוע, לשלם, לעשות
VB	Verbs	יודע, נראה, אומר, אמר
MD	Modal	עלול, יכולה, צריך, יכול, אפשר
JJ	Adjective	לאומי, גדול, חדש, רבים, אחרים
JJT	Construct-state Adjective	מרובי, חסרת, רעולי, מודעי, מדובר
RB	Adverbs	אתמול, כבר, עוד, יותר, לא
CD	Number	אחת, 1, 0
CDT	Construct Numeral	עשרות, מאות, אלפי, שתי, שני
NCD	Numerical Expression	03.02, 00.02, 11.61, 28.6.6, 11.31
PRP	Pronouns	זו, הם, היא, זה, הוא
COP	Copula (present) and Auxiliaries (past and future)	יהיה, אינו, היתה, היו, היה
ADVERB	Adverb appearing as prefix	כ
REL-SUBCONJ	Relativizer	ש
CC	Conjunction	ש, בניגוד, ככל, כפי
PREPOSITION	Prefix-Prepositions	ב, ל, מ, כ, כש
S_ANP	Nominative suffix	$_{suff}$הם, $_{suff}$הוא, $_{suff}$היא
S_PRN	Pronomial suffix	$_{suff}$הוא, $_{suff}$הם, $_{suff}$היא
AT	AT marker	את
DEF	H marker	ה
CONJ	The ו coordinating word	ו
CC-COORD	Coordinating Conjunction other than ו	גם, אם, או, אבל, רק
CC-REL	Relativizing Conjunction	אשר
CC-SUB	Subordinating Conjunction	כמו, כאשר, לאחר, כדי, כי
DT	Determiner	כל, מבחר, איזושהי
DTT	Construct-state Determiner	הרבה, שום, אותו, כמה
EX	Existential	היה, ישנם, אין, יש
IN	Preposition	בין, עם, ל, על
POS	Possessive	של
QW	Question Word	היכן, מ, האם, מי, מה
TEMP-SUBCONJ	Temporal Suboordinating Conjunction	מש, כש
INTJ	Interjection	אוי, נא, חלילה, אוף, פוס
P	"Prefix" wordlets	אנטי, תת, בין, אי, בלתי
TTL	Titles	מר, פרופסור, ניצב, עו"ד, ד"ר
PUNC	Punctuation	״ . , -

Table 4.4: *Lexical categories (parts-of-speech tagset) of the Morphological Analyzer*

4.2.3 Hebrew morphological disambiguator

The morphological analyzer provides the possible set of analyses for each token, but does not disambiguate the correct analysis in context. A *Morphological Disambiguator* (henceforth "the Hebrew tagger" or "tagger") was developed by Meni Adler at Ben-Gurion University of the Negev [4,5,53]. After the (extended) morphological analyzer assigns the possible analyses for each token in an input sentence, the tagger takes the output of the analyzer as input and chooses the single best analysis for the entire sentence (performing both token segmentation of words and part-of-speech assignment for each word). The tagger is an HMM-based sequential model that is trained in a semi-supervised fashion using EM based on the output of the morphological analyzer on a large amount (about 70M words) of unannotated Hebrew text. The tagger is described in [4,5].

The tagger's success is due in part to a smart initialization procedure to the EM training process. This initialization procedure takes the output of the analyzer and assigns a conditional probability distribution $P(tag|word)$ for each word. In other words, it assigns an a-priori, context-free likelihood for each analysis of a word (while the word "broke" can be either a verb in the past tense or an adjective, it is more likely to be the former. Such preferences can be modeled as probability distributions, and the initialization procedure attempts to learn the values of these distributions automatically from raw data). I describe the initialization procedure in [53].

The tagger is relatively accurate: it achieves 93% accuracy in predicting segmentation and tagging when measured on the core tagset and 90% accuracy when measured on the complete tagset, which includes the complete set of morphological features. Because the tagger is not trained on a particular annotated training set but instead on a very large corpus of text spanning multiple genres, its performance is robust across domains.

4.2.4 A resource-incompatibility issue

Unfortunately, the KC Analyzer adopted a different tagset than the one used in the Treebank, and analyses produced by the KC Analyzer (and hence by the morphological disambiguator) are incompatible with the Hebrew Treebank. These are not mere technical differences, but derive from different perspectives on the data. The Hebrew Treebank (TB) tagset is syntactic in nature ("if the word in this particular position functions as an adverb, tag it as an adverb, even though it is listed in the dictionary only as a noun"), while the KC tagset [4, 7, 100] takes a lexical approach to POS tagging ("a word can assume only POS tags that would be assigned to it in a dictionary"). The lexical approach does not accommodate generic modification POS-tags such as MOD,

nor does it allow listing of demonstrative pronouns as adjectives.

These divergent perspectives are reflected in different guidelines to human taggers, different principles underlying tag definitions, and different verification procedures. This difference in perspective yields different performances for parsers induced from tagged data, and a simple mapping between the two schemes is impossible to define.

In addition to these fundamental differences, there are also technical ones: For example, the TB and KC annotators have different judgements with respect to certain morphological features (for example, titles such as ח"כ (member of parliament) can be either feminine or masculine in the TB, but only masculine in the KC). In addition, many adverbs and prepositions in Hebrew are lexicalized instances of a preposition followed by a noun (e.g., ברכות, "in+softness", *softly*). These can admit both the lexicalized and the compositional analyses. Indeed, many words admit the lexicalized analyses in one of the resources but not in the other (e.g., לטובת "for+benefit" is Prep in the TB but only Prep+Noun in the KC, while for מצד "from+side" it is the other way around).

Some Hebrew forms, particularly the present-participle and modal forms are inherently hard to define, and the wide disagreement about their status is reflected in practically all Hebrew dictionaries. This kind of disagreement naturally appears also between the KC and TB. See [7] and [100] further discussion on these two interesting cases.

Bridging the discrepancy between the two resources is an important aspect in the creation of a successful parsing system. On the one hand the syntactic annotations in the Treebank are needed in order to train the parser, and on the other hand the information provided by the morphological analyzer is needed in order to provide a good lexical coverage. I discuss approaches to bridging this discrepancy in Chapters 8 and 10 where I describe the dependency and constituency Hebrew parsers.

4.3 Summary

To summarize, the Hebrew language and its analysis poses several challenges to parser design: the amount of annotated material is relatively small, precluding the possibility of learning robust lexical parameters from the annotated corpora. The productive nature of the morphology results in many word forms, adding another obstacle to estimating lexical parameters from annotated data. The nature of the word-formation mechanism in Hebrew makes it hard to guess the morphological analysis of a word based on its prefix and suffix alone as is done in other languages, requiring the use of a more complex system for handling unknown words. Many function words in Hebrew are not separated by whitespace but are instead attached to the next token, making the observed word se-

quence ambiguous. Word-segmentation needs to be performed in addition to syntactic disambiguation. Successful word-segmentation may rely on syntactic disambiguation, suggesting that it is better to perform the segmentation and syntactic-disambiguation tasks jointly. Finally, Hebrew grammar requires various forms of morphological agreement, a fact which hopefully can help disambiguate otherwise ambiguous syntactic structures. The syntactic parser should be able to make use of agreement information.

In terms of existing resources, Hebrew has a small Treebank annotated with constituency structure, and a broad-coverage, manually constructed, lexicon-based morphological analyzer. The morphological analyzer is capable of providing the possible morphological analyses for many lexical forms, and it is extended using a machine-learning technique to also provide possible analyses for word-forms not covered by the lexicon. The extended lexicon provides a good lexical coverage of Hebrew. Also available is a morphological disambiguator that is capable of associating probabilities to the possible analyses of the lexical forms in the lexicon, and disambiguating the analyses of a sequence of lexical items in context based on a sequential model. The constituency Treebank can be used to learn the parameters of a syntactic-model of Hebrew, while the morphological analyzer can be used to provide broad-coverage lexical knowledge. Unfortunately, the Treebank and the lexicon/disambiguator follow different annotation schemes, and are therefore incompatible with each other.

The annotation gap between the two resources must be bridged before they can be used together. In addition, the Hebrew constituency Treebank is not useful for learning a dependency representation of Hebrew, for which a dependency syntactic representation is required.

Part II

Dependency Parsing

Chapter 5

Hebrew Dependency Treebank

For the purposes of this thesis, I created a Modern Hebrew Dependency Treebank on the basis of the available Hebrew Constituency Treebank.
Creation of the dependency Treebank involved two related steps:

1. Deciding on dependency-annotation guidelines for Modern Hebrew
2. Converting the constituency Treebank to dependencies based on these guidelines

5.1 Dependency annotation guidelines

This section describes the major dependency annotation decisions of the Hebrew Dependency Treebank.

5.1.1 Tagging and segmentation

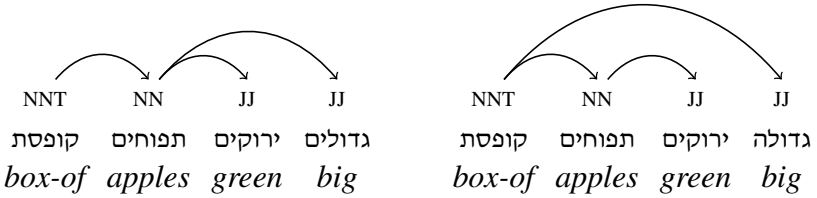
POS-tags The parts-of-speech tagset of the Constituency Treebank is syntactic in nature. In contrast, the tagset used in the dependency version of the Treebank is compatible with the lexicographic tagset used by MILA's Hebrew Morphological Analyzer (section 4.2.2) and morphological disambiguator (section 4.2.3). The tagging conversion was done in a semi-automated process (a heuristic mapping between the tagsets, which accounts for the tree context, was defined and applied. Some hard cases were left unresolved in the automatic process and marked for manual annotation). Words that lack an analysis in the Morphological Analyzer are assigned the tag !!UNK!!, and words that do not have a correct analysis in the morphological analyzer are assigned the tag !!MISS!!.

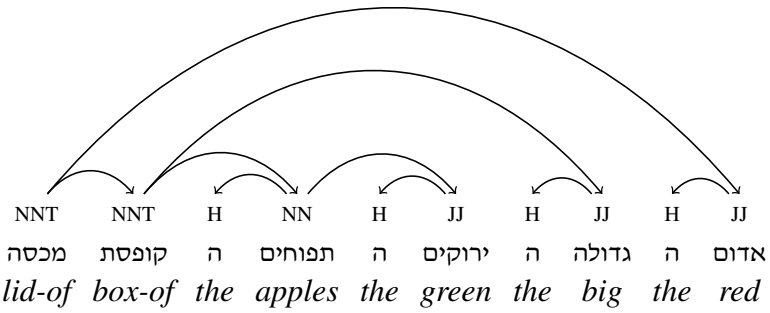
Word-segmentation The Hebrew prefixed-units מ,ש,ה,ו,כ,ל,ב are always treated as separate words. Pronominal suffixes are treated as inflections on nouns (the inflection is noted on the noun's POS-tag) and as a separate word on verbs. Inflected prepositions (בהם,בהן), possessives (שלי,שלנו), and AT markers (אותנו,אותי) are represented as several words, but the pronoun is marked as being part of a complex token (*e.g.*, בו→ $_{\text{suff}}$הוא ב).

5.1.2 Inter-word relations

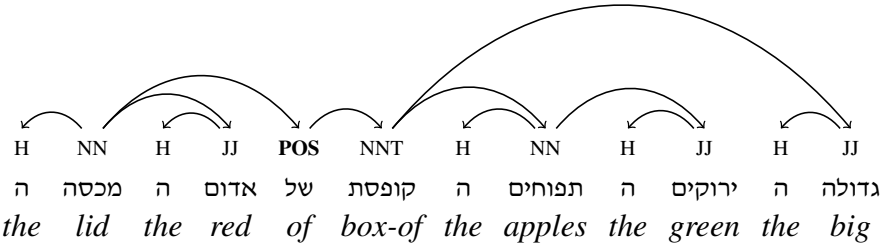
Hebrew examples are to be read from left to right.

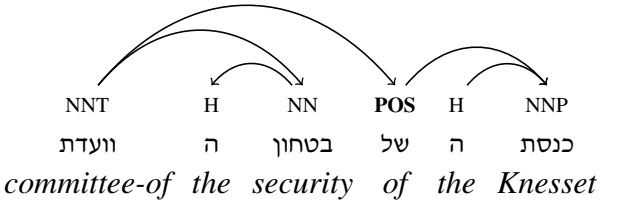
Nouns and noun-phrases Nouns (NN, NNP) are modified by adjectives (JJ) or other nouns. Construct-nouns (NNT) head their clauses and are modified by nouns:



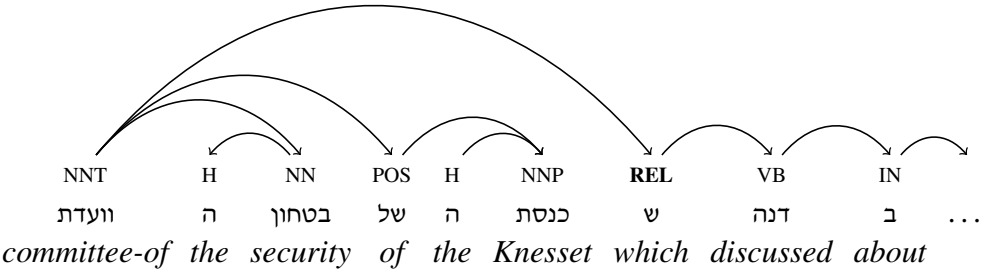


Nouns may be modified by a possessive clause:

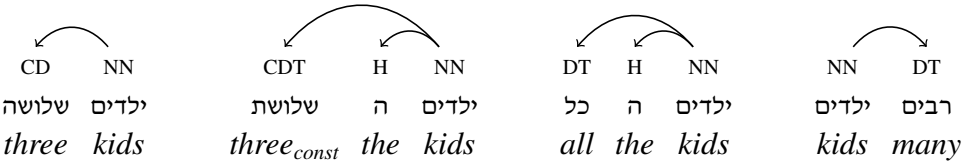




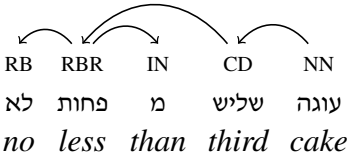
Nouns can be modified by a relative clause:



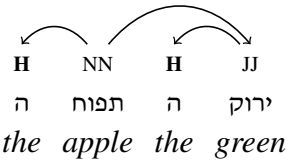
Determiners and quantifiers modify the main noun.



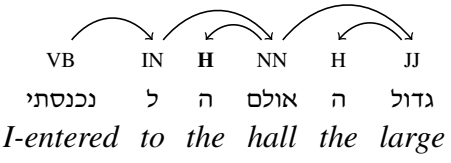
Quantifiers can themselves be modified (*e.g.*, by negation, or by other quantification). Note that partitives (יותר מ,פחות מ, "less than", "more than") are *not* handled as preposition phrases, but as multi-word quantifiers:



H ה (the definite article prefix) is treated as a morphological marker, and depends on the word it modifies:

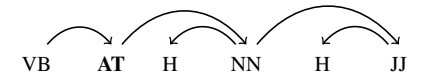


Implicit H (following כ,ל,ב) is explicitly marked:

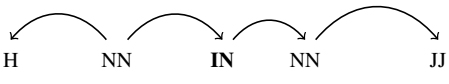


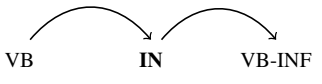
AT את is the parent of the object it is marking[1]

[1]Note that this causes some inconsistency with the non-definite (unmarked) objects, in which the verb is the direct parent of the object, without the intervening AT. A different annotation decision would choose to treat the AT marker similarly to the H definiteness marker, and have AT be a modifier of the object instead of its parent. I chose to designate AT as head because of cases where in which the object may be dropped while the AT remains (שמעתי את (מה) ש אמרת).

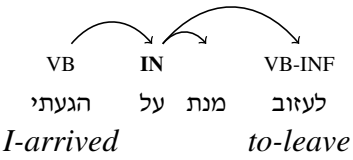


Prepositions (IN) Are the heads of preposition phrases. Prepositions depend on the word they modify (usually nouns or verbs) and are parents of the preposition object.

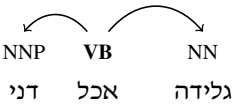


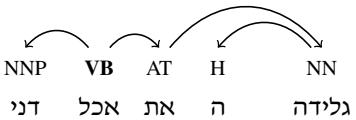


Multiword Prepositions such as (על מנת "in order to" , ב עקבות "following") are annotated as a chain, where each word depends on the word before it. The first word, the main word of the preposition, is the one depending on the modified word and the one being modified by the preposition object.

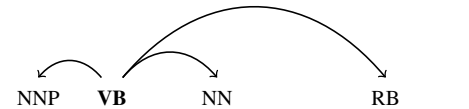


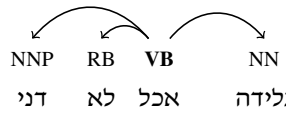
Verbs (VB), Modals (VBMD) and Infinitives (VBINF) Verbs head their clauses and are the parents of their subjects and objects.





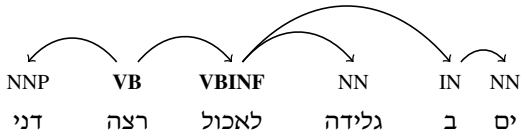
Verbs can be modified by adverbs (including negations) and prepositions.



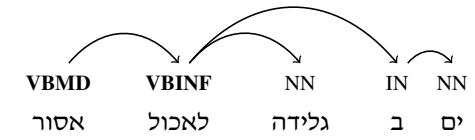


Infinitive forms depend on the main verb. In infinitive constructions, the subjects depend on the main verbs, but the objects, complements, and prepositions depend on the

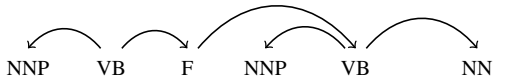
infinitive:[2]

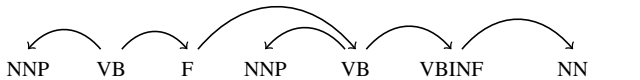


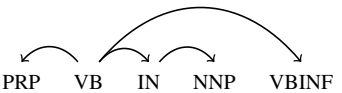
Modals behave as verbs with regard to the infinitive:

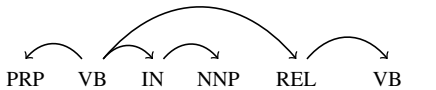


A few more verbal examples:









Auxiliaries (AUX) Auxiliaries modify the main verb when it exists[3]:

When there is no verb, an auxiliary[4] (היה,היתה,יהיה,...) may take its place:

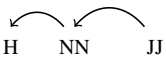
[2]In justification of this choice, note that the grammatical agreement is between the subject and the main verb which must agree in gender and number, while there is a strong semantic relation between the infinitive and its modifiers.

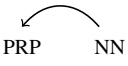
[3]This form of auxiliaries is treated as a regular verb in [60].

[4]These auxiliaries are referred to in [60] as copular elements. Here, I regard future and past copular elements as auxiliaries.

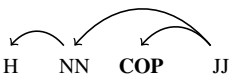
Note that auxiliaries are either in the past or the future, but not in the present (see discussion of the present copulars (היא "is$_{fem}$", הוא "is$_{masc}$") below).

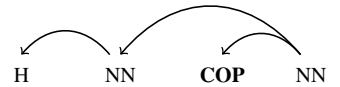
Verbless predicatives In the verbless predicative (הילד חכם "the-boy smart" standing for "the-boy is smart"), the predicate is the parent of the subject:

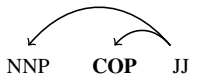




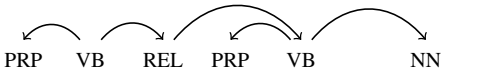
These constructions may appear with a copular element (הוא,היא,הם), which are not treated as auxiliaries: they are not the head of the construction but instead depend on the predicate:



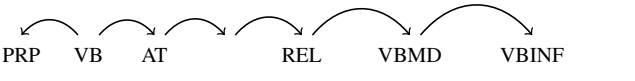


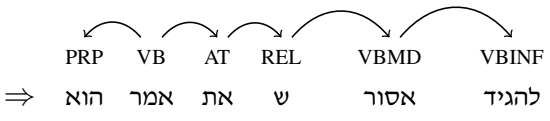


Relative clauses are headed by the relativizer word (usually ש):

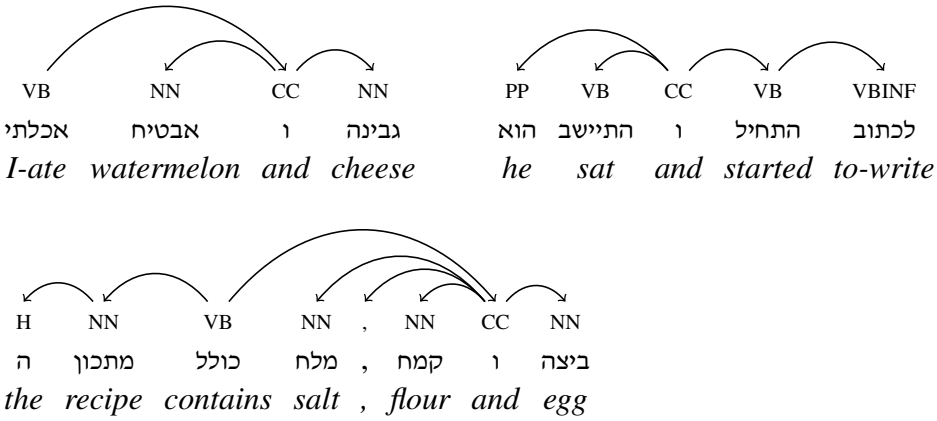


In some cases the word being modified is missing.

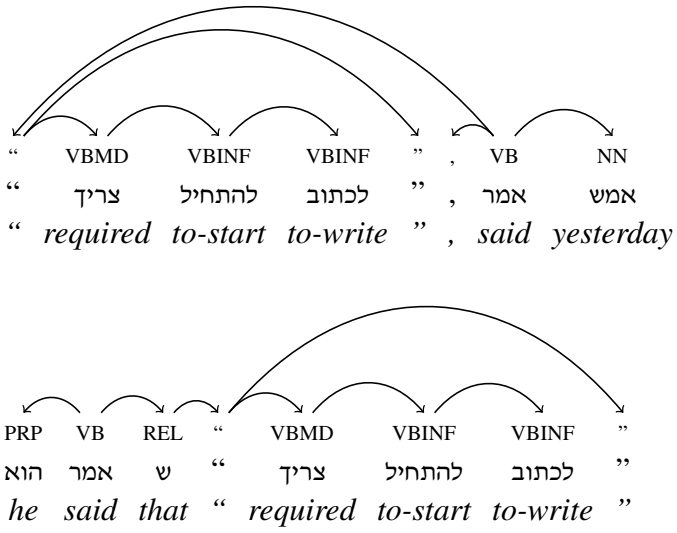




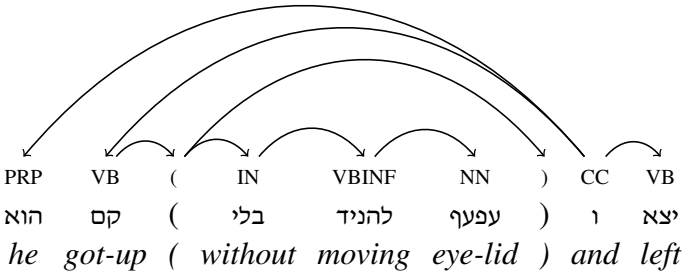
Coordination The coordinating element is the head of the conjunction.[5] In the event that there are several coordinating elements, the last one is chosen as the head of the others.

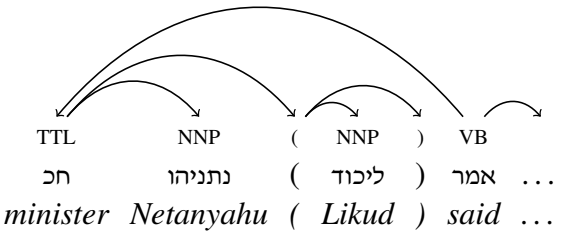


Quotes and parenthesis With balanced quotes/parentheses, the opening quote/-parenthesis is the head of the clause. The main content word and the closing element depend on the opening clause:

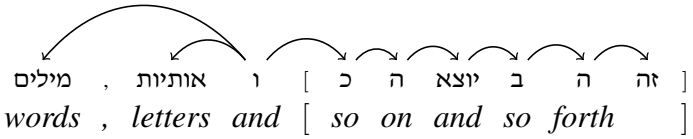


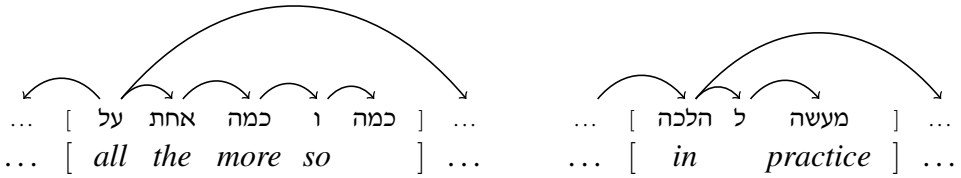
[5] Another popular treatment of coordination in dependency representations is making the first conjoined element the head of the conjunction and having the coordinating element and the other conjuncts depend on it. While it has the advantages of being somewhat easier to parse, the version I chose here is more informative. Specifically, it allows us to distinguish between modifications to the first conjunct only (*e.g.*, “red apples and bananas”), to the second conjunct only, and to both conjuncts (*e.g.*, “tasty apples and bananas”).





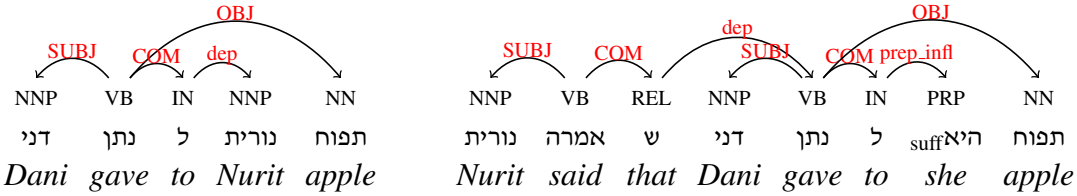
Multiword units Multiword phrases, in which the phrase has a highly lexicalized meaning and function as a single unit (*i.e.,* it can be replaced with a single word), are annotated in a flat, sequential structure, whereby every word depends on the previous one, and not in a compositional manner. The multiword phrases then interact with the sentences through the first word (no word other than the first word can have a parent or a daughter from outside the phrase).





5.1.3 Dependency labels

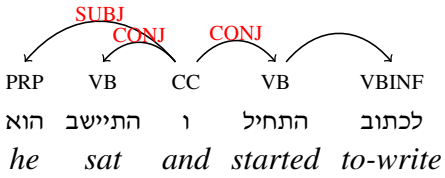
Some of the dependencies in the Treebank are labeled. Specifically, dependency labels are used to mark Subjects (SUBJ), Objects (OBJ), and Complements (COM):



Most other relations are marked with the generic "dep" label.

In addition, labels are used to:

- Distinguish coordinated elements (CONJ) from the modifiers and arguments of the coordinated structure.



  Notice how the two verbs (VB) are coordinated by the ו conjunction while the pronoun (PRP) is an argument (the subject) of both verbs.

- Mark pronominal suffixes of a token:

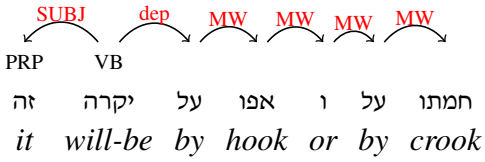
  rb_infl inflected adverbs (עודו ⇒ $_{\text{suff}}$עוד הוא "while-he")

  pos_infl inflected possessive (שלי ⇒ $_{\text{suff}}$של אני "of-I"/mine)

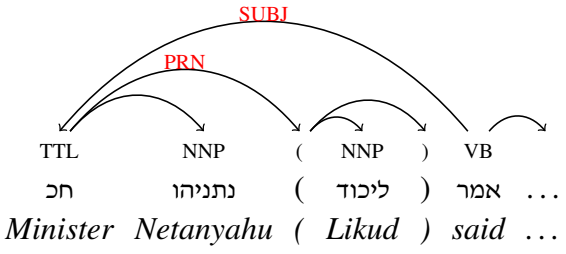
  at_infl inflected AT marker (אותנו ⇒ $_{\text{suff}}$את אנחנו "AT-us")

  prep_infl inflected preposition (בו ⇒ $_{\text{suff}}$ב הוא "in-him")

- Mark the parts of multi-word expressions (MW)



- Mark the beginnings of parentheticals and quotes (PRN)



5.2 Constituency to dependency conversion

Version 2 of the Hebrew Constituency Treebank already has a fair amount of dependency information encoded in it [61]. Most of this information could have been reused, but parts of it needed to be revised. Moreover, it needed to be extended in order to provide complete coverage and accommodate the chosen dependency annotation guidelines. In addition, some structural changes needed to be applied to the constituency trees

in order to support better dependency extraction. These structural modifications may either add structure on top of flat constructions such as NPs or remove unnecessary structure. Once the Treebank is properly revised and annotated, it can be head-annotated, and dependency structure can then be extracted based on the head annotation.

This section describes key components of the conversion procedure – namely, the major changes to the Treebank and the head annotation procedure.

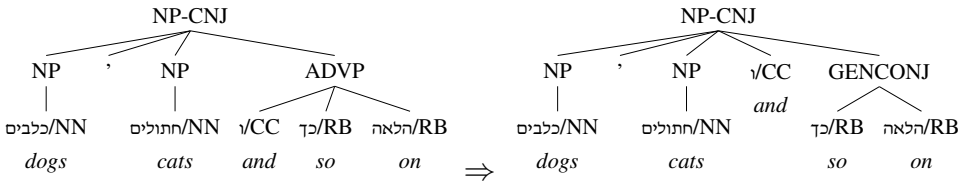
5.2.1 Fixes and modifications to the Hebrew constituency-Treebank

Marking constituents as "idioms" Idioms get a standardized dependency structure that does not necessarily follow their syntactic structure. The Hebrew TB has some idioms marked. In addition, I added nominal (אח ורע), the adjectival (קל וחומר) and the adverbials (פחות או יותר, אחת ולתמיד, על אחת כמה וכמה, כך או, כלל ועיקר, כך או כך, כך או אחרת, כהנה וכהנה, די והותר, בראש ובראשונה, באמת ובתמים, זאת ועוד, כלל וכלל, מאז ומעולם, מאז ומתמיד, מניה וביה, פה ושם, יותר ויותר, and פחות או יותר) elements.

Marking multi-word prepositions as ING Hebrew has several multi-word expressions that function as prepositions (על מנת, יחד עם, כתוצאה מ). They appear as flat structures in the Treebank. I added an extra ING (grammar-level IN) node around them.

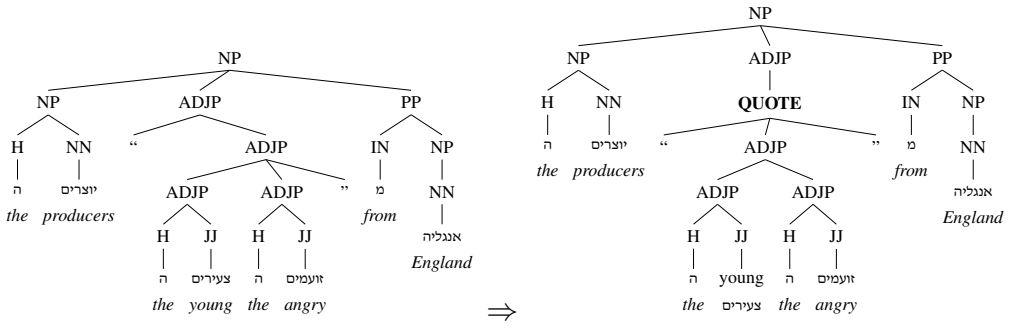
Marking coordinated constituents with a -CNJ marker Coordinated construction receives special treatment in the conversion process and should be clearly marked. Most conjunctions are already marked in the Treebank, but in a handful of cases, marking is either incorrect or missing. Such cases were manually fixed.

Removing ADVP level in coordination Coordinated structures with a generic last element (*e.g.,* כלבים, חתולים וכדומה "cats, dogs, etc") are annotated in the Treebank by representing the coordinator and generic element (ו כדומה) as an ADVP. I removed this extra level so that this structure is unified with the rest of the coordinated constructions, in which all the coordinated elements are on the same level, and marked the generic element as GENCONJ.

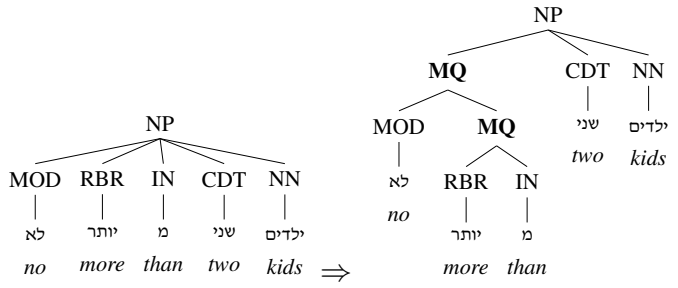


Adding a QUOTE level around quoted expressions The addition of a QUOTE level

facilitates the conversion of quotes and parentheses to dependency structures by making the first quote head of the QUOTE constituent.



Adding MQ level around complex quantifiers Multiword quantifiers, such as פחות מ (“less than”) and לא יותר מ (“not more than”), are originally annotated as flat sequences in the NP. I added an extra MQ level around such multiword quantifiers.



Assorted Mistakes A handful of other mistakes discovered during the conversion process were manually fixed.

5.2.2 Head-assignment rules

After applying the fixes and modifications, the head annotation procedure works as follows:

1. If the constituent has a single child, that child is the head.
2. If the constituent is marked as “FLAT” (*e.g.*, for idioms and multi-word prepositions), construct a chain from the leaves of the constituent, the first as the parent of the second, second as the parent of the third, etc. The first leaf is the head of the constituent.
3. If the constituent is a parenthetical (PRN) or a QUOTE, choose the first opening bracket or quote as the head.
4. If the constituent has an AT child, assign it as the head.

5. If the constituent has only one non-punctuation child, assign it as the head.

6. If one of the children is marked in the Treebank as DEP_HEAD, assign it as the head.

7. If one of the children is marked in the Treebank as DEP_MAJOR, assign it as the head.

8. If the constituent is marked in the Treebank as DEP_MULTIHEAD or as a conjunction ("-CNJ"), choose the last CC as the head. If there are no CC children, choose the last semicolon (;), otherwise choose the last comma (,).

9. In other cases, assign heads based on the *head-percolation table* in Figure 5.1. The table specifies a direction (either FIRST or LAST) and a list of grammatical categories for each constituent type. Heads are assigned by traversing the list of grammatical categories in order, and looking for a category appearing in the children of the current constituent. Then, either the FIRST or LAST child of the constituent matching the category for the list is assigned as the head.

Type	Categories	Direction
S	VB,VP,AUX,PREDP,S,SQ,PP,ADVP,FRAG,CD,NP,NNT,NNP,NNPP,SBAR,PRN	First
SQ	HAM,VP,PREDP,SQ,QW	First
SBAR	RB,DT,COM,REL,AGR,IN,VP,ADVP,S,SQ,SBAR,SQBAR,FRAG,FRAGQ	First
SQBAR	HAM,SQ,SQBAR,S	First
FRAG	FRAG,VP,NP,ADJP,PP,S,SQ,SBAR,INTJ,PRN,ADVP	First
FRAGQ	FRAGQ,HAM,SQ,VP,NP,PP,ADJP,ADVP,SBAR,QW	First
NNP	NNP,NNPP,NNT,VB,VP,NN,NP,NX,CDT	First
NP	NNT,NN,NNP,NNPP,PRP,NX,NP,QW,VB,VP,CD,DT,WDT,PP,S,SQ,SBAR,JJ,ADJP	First
NX	NN,NX,NNP,NNPP,JJ,CD	Last
VP	VB,VP	First
PP	PP,IN,POS,COM,MOD,PRP,VB,NP	First
PREDP	NP,VB,VP,ADJP,PP,SQBAR	First
ADVP	RB,RBR,ADVP,WDT,QW,JJ,CD,DT,NN,NP,DT,PRP,PP	First
ADJP	JJT,JJ,ADJP,ADJX,NNT,NN,NP,QW,ADVP	First
ADJX	JJ	First
CD	CD,CDP	Last
CDT	CD,CDP,CDT,CDPT	Last
INTJ	INTJ,VP,NP,NN,ADJP,ADVP,PP,RB,S,SBAR	First
IN	IN,INP	First
NNPP	NNP,NNPP,NNT,VB,VP,NN,NP,NX,CDT	First
INP	IN,INP	First
CDP	CD,CDP	Last
CDTP	CD,CDP,CDT,CDPT	Last
MQ	MQ,RBR,CD,WDT,DT	First
PRN	yyLRB,VP,NNP,NNPP,NP,ADJP,ADVP,PP,S,SQ,SBAR,SQBAR,FRAG,INTJ,yyELPS	First
QUOTE	yyQUOT	First

Figure 5.1: *Modern Hebrew head-percolation table. The table is based on the head-rules described in [142], with some additions and modifications.*

5.2.3 Treebank statistics

Like the Constituency Treebank, the Hebrew Dependency Treebank has 6220 sentences, and 155,569 words (the משהוכלב prefixes are considered as separate words in this count). Sentences range in length from 1 to 117 words, the average sentence length being 26 words. Figure 5.2 shows a histogram of sentence lengths.

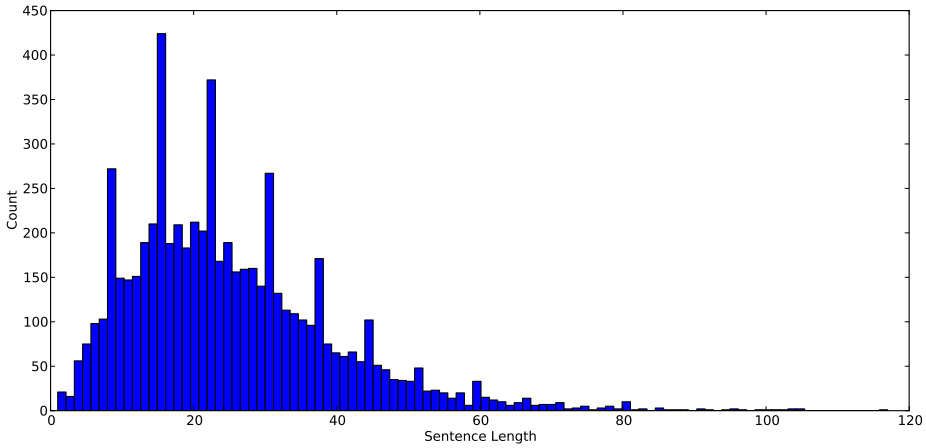


Figure 5.2: *Histogram of sentence lengths in the Hebrew Dependency Treebank*

3905 nodes are labeled as Objects, and 8947 as Subjects. 598 sentences have no subject, 3279 have one subject, 1598 have two, 520 have three and the rest have four or more subjects. 3319 of the sentences have no marked object, 2090 have one object, 608 have two objects, and the rest have three or more objects.

The root element is predominantly verbal, occurring in 4048 of the sentences (2327 verb, 1139 beinoni, 297 modal, and 285 copular). The other sentences are headed by conjunctions (1114), nouns (329), adjectives (120), or other.

There is an average of 0.96 children per node. Table 5.1 breaks down the average number of children by the part-of-speech categories of the head word. Verbs and conjunctions have the highest numbers of modifiers, followed by nouns. While not shown in the table, some verbs and conjunctions have more than eight children.

5.3 Summary

Starting with a constituency Treebank, I obtained a manually verified, accurate dependency Treebank. The main added-value of the transformation process is the identification of lexical heads. This Dependency Treebank serves as the dataset for training the

dependency parsers described in subsequent structures, and is also useful on its own right as a reference syntactic description of Modern Hebrew.

Tag	Avg. # Children	Tag	Avg. # Children
ADVERB	0.00	POS	1.01
INTJ	0.00	AT	1.04
P	0.03	PREPOSITION	1.05
S_PRN	0.03	REL-SUBCONJ	1.05
CDT	0.04	IN	1.12
DEF	0.05	TEMP-SUBCONJ	1.12
S_ANP	0.08	NN	1.26
DTT	0.10	JJT	1.30
RB	0.13	COP	1.31
PRP	0.18	CC	1.42
PUNC	0.19	VB-TOINFINITIVE	1.45
NCD	0.23	NNT	1.48
NNP	0.41	BNT	1.50
CD	0.43	BN	2.51
DT	0.50	EX	2.57
NN_S_PP	0.67	CONJ	2.74
JJ	0.68	VB	2.88
QW	0.85	MD	3.00

Table 5.1: *Average number of children for each category in the Hebrew Dependency Treebank. The numbers are sorted*

Chapter 6

Background on Dependency Parsing

Dependency parsing is the task of automatically assigning a dependency structure to an input sentence. This chapter presents the measures used to evaluate dependency parsing accuracy, and surveys the common methods used for dependency-parsing.

6.1 Evaluation measures

The measures used in the literature for evaluating dependency parsers are:

Unlabeled Attachment Score (UAS) Percentage of tokens (over the entire test corpus) assigned a correct parent.

Labeled Attachment Score (LAS) Percentage of tokens (over the entire test corpus) assigned a correct parent and a correct label.

Unlabeled Exact Match Percentage of sentences with a UAS score of 100%.

Labeled Exact Match Percentage of sentences with an LAS score of 100%.

Root Accuracy Percentage of sentences in which the `ROOT` attachment is correct.

These metrics assume that the sequence of words in the parse tree is fixed – the gold token-segmentation is assumed to be known.

Generalization to handle uncertain word segmentation These metrics cannot be used when the token-segmentation is automatically inferred, because some tokens appearing in the correct structure may not appear in the predicted one, and vice-versa. For evaluating dependency parsing accuracy for the automatic-segmentation case, I propose a generalization of the UAS and LAS metric which is defined in terms of *precision* and *recall*.

Formally: An input sentence S is composed of tokens t_i, where each tokens may correspond to one or more words: $t_i = w_1, \ldots, w_r$. For example the token מספר can be segmented into either one word (מספר) or two words (מ ספר). When dealing with token segmentations, we append each word to the index of the token from which it originated, so if the token מספר was the 5th token in the sentence, its segmentation will be either the word $\langle$ 5,מספר $\rangle$ or the two word sequence $\langle$ 5,מ $\rangle$,$\langle$ 5,ספר $\rangle$.

A given segmentation of the sentence assigns one token segmentation (a sequence of token-indexed words) to each token in the sentence. Let $W = w_1, \ldots, w_m$ be a sequence of token-indexed words for a sentence with n tokens ($m \geq n$). A dependency-structure over the words W can be represented by set of $\langle \text{head}, \text{modifier} \rangle$ pairs, where head $\in W \cap ROOT$ and modifier $\in W$. Let W^g be the gold (correct) word-segmentation of sentence S, and D^g be the gold dependency-structure (set of pairs) over the token-indexed words in W^g. Similarly, W^p and D^p are the predicted word segmentation for the sentence S and the dependency-structure defined over them.

Precision (P) and Recall (R) are defined as:

$$\text{precision} = \frac{|D^g \cap D^p|}{|D^p|} \qquad \text{recall} = \frac{|D^g \cap D^p|}{|D^g|}$$

Precision is the fraction of correctly predicted $\langle \text{head}, \text{modifier} \rangle$ pairs out of all the predicted $\langle \text{head}, \text{modifier} \rangle$ pairs, and Recall is the fraction of correctly predicted $\langle \text{head}, \text{modifier} \rangle$ pairs out of all the correct $\langle \text{head}, \text{modifier} \rangle$ pairs. The harmonic mean F_1 measure combines precision and recall into a single measure:

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

When the word-segmentations of the gold and predicted structures agree ($W^g = W^p$), both the gold and predicted structures have the same number of dependency links ($\langle \text{head}, \text{modifier} \rangle$ pairs), and we get precision = recall = F_1 = UAS. However, when the word segmentations disagree, both the recall and precision are penalized.

Hebrew-specific relaxation Specific to Hebrew, the most common segmentation mistake (accounting for over 40% of the cases) is not introducing a "hidden" determiner when one is appropriate (*i.e.*, the Hebrew token בבית can be interpreted as either ב בית (in house) or ב ה בית (in the house). The ה definiteness marker is not realized in writing after the function words ב, ל or כ). These hidden elements are not critical for recovering the argument structure of the sentence and are ignored during the evaluation (the ה marker appearing after either ב, ל or כ is removed from the gold and parsed sentences prior to evaluation and the relation between it and its parent is not taken into account).

Not taking the hidden determiner into account when calculating the parsing score has the effect of somewhat decreasing precision (because the attachment decision for the ה marker is an easy one) while increasing recall. Segmentation mistakes involving the hidden determiner can still affect parsing performance, as the parser does rely on the determiners when constructing the parse tree for the sentence.

I use F_1 (while ignoring the hidden ה determiner) instead of UAS and *labeled* F_1 instead of LAS[1] for all the reported results involving automatic segmentation.

6.2 Dependency-parsing algorithms

Dependency parsing has been a topic of active research in natural language processing in the last several years. An important part of this research effort are the CoNLL 2006 and 2007 shared tasks [19, 103], which allowed for a comparison of many algorithms and approaches for this task on many languages.

Current data-driven dependency parsers can be broadly categorized into three families – local-and-greedy transition-based parsers (*e.g.*, MALTPARSER [104]), globally optimized graph-based parsers (*e.g.*, MSTPARSER [93]), and hybrid systems (*e.g.*, [105, 122]), which combine the output of various parsers into a new and improved parse.

In contrast to constituency parsers, which usually perform tagging and parsing jointly, dependency-parsers expect POS-tagged sentences as their input. This implies a pipeline architecture in which a POS-tagger is used to tag the input text, which is then passed on to the parser.

6.2.1 Transition-based (Shift Reduce) dependency parsing

Transition-based parsers scan the input from left to right, are fast ($O(n)$), and can exploit rich feature sets, which are based on all the previously derived structures. However, all of their decisions are very local, and the strict left-to-right order implies that, while the feature set can use rich structural information to the left of the current attachment point, it is severely restricted in the information it can use to the right of the attachment point: only the next two or three input tokens are available to the parser. This limited look-ahead window leads to error propagation and worse performance on root and long distant dependencies relative to graph-based parsers [94].

Such parsers are usually greedy and deterministic, but were recently extended to use beam search [154] and dynamic programming [66]. While effective, the extensions

[1] Labeled precision, recall and F_1 (corresponding to LAS) are calculated by replacing the $\langle$head, modifier$\rangle$ pairs by $\langle$head, modifier, label$\rangle$ triplets.

make the parsers significantly slower (they remain $O(n)$ algorithms, but the hidden constant grows significantly making the parsing process about 16 times slower in practice).

Parsing framework The transition-based parsing framework assumes an abstract machine (a *transition system*) that processes sentences and produces parse-trees. The transition system has an internal configuration and a set of actions. Applying a given action at a given configuration results in a new configuration. After a finite set of action applications, the sentence is fully processed, and a parse-tree is produced. A classifier is used to choose the correct action at each configuration, based on features extracted from the configuration. Various such transition-systems are defined in the literature (for a good overview see [102]). The transition-systems differ in their definition of a configuration and their action sets.

I describe a representative transition-system (the "Arc Standard" system) in detail, and refer the reader to [83, 102] for further discussion of various transition-based systems.

The ArcStandard transition system. The configuration of the ArcStandard system is composed of a stack S, a queue Q, and a set D. The items on the stack and the queue are sentence words, where each word may contain an additional fixed set of properties, such as its POS-tag. The set D contains dependency links, which are pairs of the form $\langle \text{head}, \text{modifier} \rangle$ where *modifier* is a sentence word and *head* is either a sentence word or the ROOT symbol. When parsing a sentence of n words $w_1, \ldots, w_n$, the system is initialized with an empty stack, an empty set, and a queue containing the ROOT symbol followed by the words of the sentence in order (the first item to be removed from the queue is the ROOT symbol, followed by w_1, followed by w_2, etc.).

The set of allowed actions are SHIFT, REDUCELEFT, and REDUCERIGHT. The SHIFT action pops a word from the queue and pushes it on the stack. The REDUCELEFT action pops two items (w_a,w_b) from the stack, adds a dependency link of the form $\langle w_a, w_b \rangle$ to D, and pushes w_a back to the stack (w_b is not allowed to be ROOT). The REDUCERIGHT action pops two items (w_a,w_b) from the stack, adds a dependency link $\langle w_b, w_a \rangle$ to D, and pushes w_b back to the stack (w_a is not allowed to be ROOT). The process ends when the stack contains a single ROOT symbol and the queue is empty, and the set of dependencies D is returned. The transition sequence for parsing the sentence "a brown fox jumped with joy" is shown in Figure 6.1.

It is trivial to show that the system terminates in $O(n)$ steps: the SHIFT action decreases the number of elements in the queue by one, and no action is allowed to increase the number of elements in the queue, so it can be applied at most $n + 1$ times

Step	S	Q	D	Action
1		ROOT a brown fox jumped with joy	{}	Shift
2	ROOT	a brown fox jumped with joy	{}	Shift
3	ROOT a	brown fox jumped with joy	{}	Shift
4	ROOT a brown	fox jumped with joy	{}	Shift
5	ROOT a brown fox	jumped with joy	{}	ReduceRight
6	ROOT a fox	jumped with joy	{⟨fox, brown⟩}	ReduceRight
7	ROOT fox	jumped with joy	{⟨fox, brown⟩,⟨fox, a⟩}	Shift
8	ROOT fox jumped	with joy	{⟨fox, brown⟩,⟨fox, a⟩}	ReduceRight
9	ROOT jumped	with joy	{⟨fox, brown⟩,⟨fox, a⟩ ⟨jumped, fox⟩}	Shift
10	ROOT jumped with	joy	{⟨fox, brown⟩,⟨fox, a⟩ ⟨jumped, fox⟩}	Shift
11	ROOT jumped with joy		{⟨fox, brown⟩,⟨fox, a⟩ ⟨jumped, fox⟩}	ReduceLeft
12	ROOT jumped with		{⟨fox, brown⟩,⟨fox, a⟩ ⟨jumped, fox⟩,⟨with, joy⟩}	ReduceLeft
13	ROOT jumped		{⟨fox, brown⟩,⟨fox, a⟩ ⟨jumped, fox⟩,⟨with, joy⟩ ⟨jumped, with⟩}	ReduceLeft
14	ROOT		{⟨fox, brown⟩,⟨fox, a⟩ ⟨jumped, fox⟩,⟨with, joy⟩ ⟨jumped, with⟩,⟨ROOT , jumped⟩}	DONE

Figure 6.1: *Parsing the sentence "a brown fox jumped with joy" using the ArcStandard transition system.*

before the queue is empty. Similarly, the REDUCE actions each decrease the number of elements on the stack by one. As the only items allowed on the stack are the n sentence words (plus the ROOT symbol), REDUCE can also be applied at most n times before the stack contains only a single item. As the first item pushed to the stack is the ROOT symbol and the REDUCE actions cannot change the order of elements on the stack or remove the ROOT symbol, the last item on the stack will be ROOT. Thus, after $2n + 1$ actions, the queue is empty, the stack contains a single ROOT item, and the process terminates. At the end of the process, the set D contains a valid dependency tree: each REDUCE action adds one $\langle \text{head}, \text{modifier} \rangle$ pair to D, and the *modifier* is never returned to the stack or the queue. Thus, the final set D will contain at most one parent for each word. As there are n REDUCE actions, there are exactly n dependency-links in D, ensuring that each of the n words of the sentence appears as a *modifier* in D. It is slightly harder, but possible, to show that this system is capable of producing all (and only) projective dependency trees (a proof is available in [102]). The proof is based on the fact that given a desired dependency tree structure, one can easily construct an "oracle" capable of choosing the correct action at each configuration so that at the end of the process, D will contain the desired structure.

Classifier-based parsing A classifier is trained to mimic the behavior of the oracle. This is done by running the oracle on a set of sentences and their known tree structures,

and recording the sequence of configurations and action taken at each configuration. A classifier is then trained based on this data to predict an action for each configuration. The features for the classifier are extracted from the configuration. Features can include information regarding the words on the stack, the words in the queue, and the set of already built dependencies. A well-crafted feature-set is described in [63].

Incremental, buffer-based processing All the transition-systems defined in the literature share the notion of an input buffer (usually a queue) and a SHIFT action that moves input tokens from the buffer into the processing area (which is usually either a stack or a pair of lists). This amounts to incremental processing of the input sentence (one word at a time, either from left-to-right or right-to-left). This sequential processing dictates a deterministic[2] parsing oracle, that gives rise to a natural training procedure in which the learning algorithm is trained to mimic the deterministic oracle. The sequential processing also constrains the features that can be used in the classifier when predicting the next parsing action. While the classifier can condition on rich syntactic structures based on the parsing history (structures already built based on the words seen so far), it has access to a very limited amount of information in the input buffer (the next few words in the queue), and the entire process is prone to error propagation.

6.2.2 Graph-based dependency parsing

Graph-based parsers are globally optimized. They perform an exhaustive search over the space of all possible parse trees for a sentence, and find the highest scoring tree. In order to make the search tractable, the feature set needs to be restricted to features over single edges (first-order models) or edge pairs or triplets (higher-order models, e.g. [23, 81, 95]). Several attempts have been made to incorporate arbitrary tree-based features but these involve either solving an ILP problem [88, 118] or using computationally intensive sampling-based methods [99]. As a result, these models, while accurate, are slower than their transition-based counterparts ($O(n^3)$ for projective, first-order models, higher polynomials for higher-order models, and worse for richer tree-feature models).

Edge-factored parsing In this section I describe the working of the edge-factored first order dependency parser introduced by [93], and implemented in the freely available MSTPARSER.

[2] In practice, there are few cases where two possible action sequences amount to the same structure, but these are always normalized in a predefined way.

Graph-based parsers view parsing as an optimization problem in which the goal is finding the highest scoring tree over a given sentence.

First-order models In edge-factored models, each of the n^2 possible $\langle parent, modifier \rangle$ pairs for a sentence of length n is assigned a score, and the goal of parsing is to find a set of edges ($\langle parent, modifier \rangle$ pairs) such that their total score is maximized while still honoring the structural constraint requiring the edges to form a directed (projective) tree. The "brains" of the algorithm comprise the scoring function $score(w_i, w_j)$ which is in charge of assigning the scores to the possible edges. The scoring function is learned from data, and is discussed later.

Solving the optimization problem Having n^2 edge scores in hand, the best directed tree (not necessarily projective) can be found in $O(n^2)$ time using the Chu-Liu-Edmonds algorithm [29, 43, 96]. For projective trees, the optimization can be performed using the Eisner dynamic-programming algorithm [44, 74, 93] in $O(n^3)$. Both algorithms involve a very small constant, and are fairly fast to run in practice. For sentences of reasonable length (at least up to 70 words) the bottleneck of both algorithms is in calculating the $O(n^2)$ edge scores prior to the search [74].

The scoring function The scoring function takes a linear form $score(w_i, w_j) = \vec{w} \cdot \phi(w_i, w_j, s)$, where $\vec{w}$ is a weight-vector whose values are learned from data, and ϕ is a feature extraction function which maps a word-pair and its sentence context into a feature-vector. The training procedure assigns the weight values in $\vec{w}$ such that the sum of the edge scores in a correct trees is higher than the sum of the edge scores in any incorrect tree. The exact training procedure is described in [83, 92]. The design of the feature function ϕ is central to achieving good parsing accuracy. A well-optimized set of features was presented in [93]. Note, however, that the feature function is inherently restricted: each edge must be scored separately, and the feature function does not have access to structural information such as the existence of other edges or their possible scores. This restriction is due to the strong *independence assumption* of the parsing model, which assumes that the edge scores are independent of each other. This independence assumption is necessary in order to make the search tractable, but several forms of linguistic information cannot be captured because of it. For example, the scoring function can neither encode the information that verbs must have only one subject nor model PP-attachment accurately. Still, owing to the gloval search performed when parsing, edge-factored models can produce very accurate parses.

Second-order models In second-order models, scores are assigned to edge pairs, or word triplets, such as $\langle parent, modifier1, modifier2 \rangle$. Then, search is performed to recover the tree whose parts have the maximal total score. The optimization problem is now harder – intractable for the non-projective case – but can still be solved in polynomial time using an extension of the Eisner dynamic programming algorithm [95]. This results in slower parsing, as $O(n^3)$ scores need to be calculated, and the dynamic-programming search is also slower. However, the scoring function, while still restricted, is much more flexible and can capture more kinds of linguistic information. Second-order models provide state-of-the-art parsing accuracies while remaining fairly efficient.[3]

6.2.3 Hybrid-systems and ensemble-methods

Hybrid systems are based on the combination of several parsing systems.

Parse-ensembles work by running k parsing systems independently, producing k different parses, and searching for a consensus parse that tries to incorporate the predictions of the various component parsers. This kind of parser combination is very effective when more than two different parsers are available. Accuracy improves as a function of the number and diversity of the different parsers [10, 122, 135].

Stacking Another form of a hybrid-system is *parser-stacking*. In the stacking framework, the predictions of one parser are used as additional features for another parser. Parser stacking was introduced in [105], and formalized in [138]. It is effective when two different parsers are available, but the exact feature-set and parser order needs to be tweaked for each parsers pair. [105] demonstrated that using the MST parser using features based on a first run of the MALT parser is more effective than when stacked in the opposite direction, *i.e.,* using the MALT parser with features derived from a first run of the MST parser.

6.2.4 Labeled dependency parsing

The parsing algorithms described so far produce *unlabeled* dependency trees, but the algorithms can be easily extended to support labeled parsing. The labeled version of the

[3] Recently, [81] introduced a dynamic-programming algorithm for third-order dependency parsing. This algorithm provides even higher accuracies for English parsing, but it is much slower to run both in theory $O(n^4)$ and in practice.

ArcStandard transition-based algorithm works by replacing the REDUCELEFT and REDUCERIGHT actions with REDUCELEFT-X and REDUCERIGHT-X actions. These new actions function as before, but instead of adding a $\langle \text{head}, \text{modifier} \rangle$ pair to D, they add a triplet $\langle \text{head}, \text{modifier}, \text{X} \rangle$, where X is an edge-label (there is a different REDUCELEFT-X action for each possible label X). Similarly, the edge-factored model can also work with edges of the form $\langle \text{head}, \text{modifier}, \text{label} \rangle$ with very few modifications. An alternative approach for producing labeled dependency trees is to use a pipeline process in which a classifier is used to assign edge-labels to the edges of an unlabeled dependency tree after it is constructed. The benefits of the pipeline approach include its fast speed (requires only n classification actions) and the ability to condition edge-labeling decisions on arbitrary large contexts. The downside of the pipeline approach is that edge-label information is not available during the creation of the tree structure, and cannot be used to guide the creation of the structure. This concern applies mostly to the transition-based systems and higher-order graph-based systems, as first-order edge-factorization already precludes the scoring of an edge to be based on the existence of other edges, including their labels. The pipeline approach works well in practice.

6.3 Related work in dependency parsing of morphologically rich languages

The CoNLL 2006 and 2007 shared-tasks [19, 103] were devoted to multilingual dependency parsing. Parsing systems were evaluated based on their accuracies in parsing many different languages. Trends from the shared task results indicate that while the better-performing parsers are more accurate across all the languages, some languages (notably morphologically rich ones such as Arabic, Basque and Greek) are harder to parse than other languages, as reflected by low parsing scores for these languages from all the participating systems [103].

The results of the CoNLL shared tasks suggest that no single generic parsing algorithm is suitable for parsing all languages, and that parsing algorithms can benefit from language-specific enhancements. Indeed, most submissions to the first workshop on statistical parsing of morphologically rich languages [143] that took place in the summer of 2010, presented language-specific studies and tried to answer the question: what gains can be achieved by tailoring aspects of the parsing algorithm to the language at hand? Most dependency-parsing research presented at the workshop [8, 9, 16, 54, 89] revolved around transition-based shift-reduce parsers, and explored variations either in the feature-set provided to the parser or in the amount of information encoded in

the POS-tags. Another line of work [16] applied a deterministic transformation to the syntactic parses, both prior to training and (in reverse order) after parsing, in order to present the parser with structures in which the relevant information needed in order to make correct attachment decisions is more local and readily available to the parser. With respect to the tagset, it was observed for the case of Arabic that a simpler part-of-speech tagset that is easier to predict correctly is more useful than a more elaborate tagset that provides more information but that is more difficult for the tagger to predict [89]. With respect to the feature-set, clear trends have emerged regarding the *kinds* of morphological information that are effective for dependency parsing. Morphological CASE was shown to be beneficial across the board, where it contributed to parsing Basque, Hebrew[4], Hindi, and, to some extent, Arabic.[5] Morphological DEFINITENESS and STATE are beneficial for Hebrew and Arabic when explicitly represented in the model. STATE, ASPECT and MOOD are beneficial for Hindi, but provide only marginal benefit for Arabic. CASE and SUBORDINATIONTYPE are the most beneficial features for Basque transition-based dependency parsing.

A more in-depth examination of the results mentioned in the previous paragraph suggests that, beyond the kind of information that is being used, the way in which morphological information is represented and used by the model has substantial ramifications as to whether it leads to performance improvements. The so-called "agreement features" GENDER, NUMBER, and PERSON, provide for an interesting case study in this respect. When included directly as machine learning features, agreement features benefit dependency parsing for Arabic [89], but not for Hindi (dependency) [8,9] or Hebrew [54]. However, when agreement patterns are directly represented on dependency arcs, they contribute an improvement for Hebrew dependency parsing [54].

All previous results on dependency parsing of Arabic assume that the gold segmentation is known. The best published results for Arabic dependency parsing are those of [89], who report UAS of 83.65% and LAS of 80.45% on gold segmentation and predicted tags, using a specific setting of the MALT parser. The authors of [89] are currently working on adapting the EASYFIRST parser I present in Chapter 7 to Arabic, and they report substantially better results than those obtained by either the MALT or MST parsers (personal communication).

[4]The Hebrew results discussed in this section are based on earlier versions of the model I present in Chapter 8.

[5]For Arabic, CASE is useful when gold morphology information is available, but substantially hurt results when it is not.

Chapter 7

EASYFIRST Dependency Parsing

In this chapter I propose a new dependency parsing algorithm, inspired by [128]: a non-directional easy-first parser.[1]

The proposed parser follows a greedy, deterministic parsing approach, but relaxes both the left-to-right processing order of transition-based parsing algorithms, and the independence assumptions of graph-based parsers. Because of its flexible but deterministic processing order, the parser can explicitly incorporate rich structural features derived from both sides of the attachment point, and implicitly take into account the entire previously derived structure of the whole sentence. This extension allows the incorporation of much richer features than those available to transition- and especially to graph-based parsers, and greatly reduces the locality of transition-based algorithm decisions. On the other hand, it is still a greedy, deterministic, search-less algorithm leading to an efficient implementation. The move from left-to-right to easy-first processing order introduces a non-deterministic parsing oracle, and requires a new training procedure.

I present a concrete $O(nlogn)$ parsing algorithm and a corresponding training procedure. The algorithm substantially outperforms state-of-the-art transition-based parsers, while closing the gap to graph-based parsers.

7.1 Non-directional easy-first parsing

When humans comprehend a natural language sentence, they arguably do it in an incremental, left-to-right manner. However, when humans consciously annotate a sentence with syntactic structure, they hardly ever work in fixed left-to-right order. Rather, they start by building several isolated constituents by making easy and local attachment

[1] Most of the material presented in this chapter was previously published in [55].

decisions, and only then do they combine these constituents into bigger constituents, jumping back-and-forth over the sentence while proceeding from easy to more difficult phenomena to analyze. By the time the more complex decisions must be made, much of the structure is already in place, and can be used to deciding on a correct attachment.

Our parser follows a similar kind of annotation process that begins with easy attachment decisions and then proceeds to progressively harder ones. As such, for decisions made farther into the process, the parser has access to the entire structure built in earlier stages. During the training process, the parser learns its own notion of easy and hard, and it learns to defer specific kinds of decisions until more structure is available.

The easy-first parsing algorithm can be described as a transition-system: the parser maintains an internal configuration, and works in steps. At each step, action is chosen and applied on the current configuration, resulting in a new configuration. After a finite set of action applications, the process terminates and a parse-tree is produced. However, while "traditional" transition-systems consume the output from left-to-right, one word at a time, the easy-first algorithm does not impose such a restriction.

7.2 The transition system

I begin by describing the transition-system: defining the parser's configuration, the set of actions, and their semantics.

Configuration A configuration is composed of a list L, and a set D. The items in L are sentence words, and the items in D are dependency-links: pairs of the form $\langle \text{head}, \text{modifier} \rangle$. Indexes in L are zero-based.

Initialization Let $S = w_1, \ldots, w_n$ be an n-word sentence to be parsed. D is initialized as the empty set, and L is initialized to contain the special `ROOT` word followed by the sentence words in order: $L = \texttt{ROOT}, w_1, \ldots, w_n$.

Termination Parsing of an n-words sentence is complete when the list L contain a single `ROOT` item, at which point the set D contain n dependency links that represent a valid projective dependency tree rooted by the `ROOT` symbol.

Actions The possible actions are ATTACHLEFT$_i$ and ATTACHRIGHT$_j$, $0 \leq i < |L|$, $0 < j < |L|$.

Semantics

- the action ATTACHLEFT$_i$ adds $\langle L_i, L_{i+1} \rangle$ to D and removes L_{i+1} from L.
- the action ATTACHRIGHT$_j$ adds $\langle L_{j+1}, L_j \rangle$ to D and removes L_j from L.

Step	L	D	Possible Actions	Chosen Action
1	ROOT$_0$ a$_1$ brown$_2$ fox$_3$ jumped$_4$ with$_5$ joy$_6$	{}	ATTLEFT$_0$,...,ATTLEFT$_5$ ATTRIGHT$_1$,...,ATTRIGHT$_5$	ATTRIGHT$_2$
2	ROOT$_0$ a$_1$ fox$_2$ jumped$_3$ with$_4$ joy$_5$	{⟨fox, brown⟩}	ATTLEFT$_0$,...,ATTLEFT$_4$ ATTRIGHT$_1$,...,ATTRIGHT$_4$	ATTRIGHT$_1$
3	ROOT$_0$ fox$_1$ jumped$_2$ with$_3$ joy$_4$	{⟨fox, brown⟩,⟨fox, a⟩}	ATTLEFT$_0$,...,ATTLEFT$_3$ ATTRIGHT$_1$,...,ATTRIGHT$_3$	ATTRIGHT$_1$
4	ROOT$_0$ jumped$_1$ with$_2$ joy$_3$	{⟨fox, brown⟩,⟨fox, a⟩ ⟨jumped, fox⟩}	ATTLEFT$_0$,...,ATTLEFT$_2$ ATTRIGHT$_1$,ATTRIGHT$_2$	ATTLEFT$_2$
5	ROOT$_0$ jumped$_1$ with$_2$	{⟨fox, brown⟩,⟨fox, a⟩ ⟨jumped, fox⟩,⟨with, joy⟩}	ATTLEFT$_0$,ATTLEFT$_1$ ATTRIGHT$_1$	ATTLEFT$_1$
6	ROOT$_0$ jumped$_1$	{⟨fox, brown⟩,⟨fox, a⟩ ⟨jumped, fox⟩,⟨with, joy⟩ ⟨jumped, with⟩ }	ATTLEFT$_0$	ATTLEFT$_0$
7	ROOT$_0$	{⟨fox, brown⟩,⟨fox, a⟩ ⟨jumped, fox⟩,⟨with, joy⟩ ⟨jumped, with⟩, ⟨ROOT , jumped⟩ }		DONE

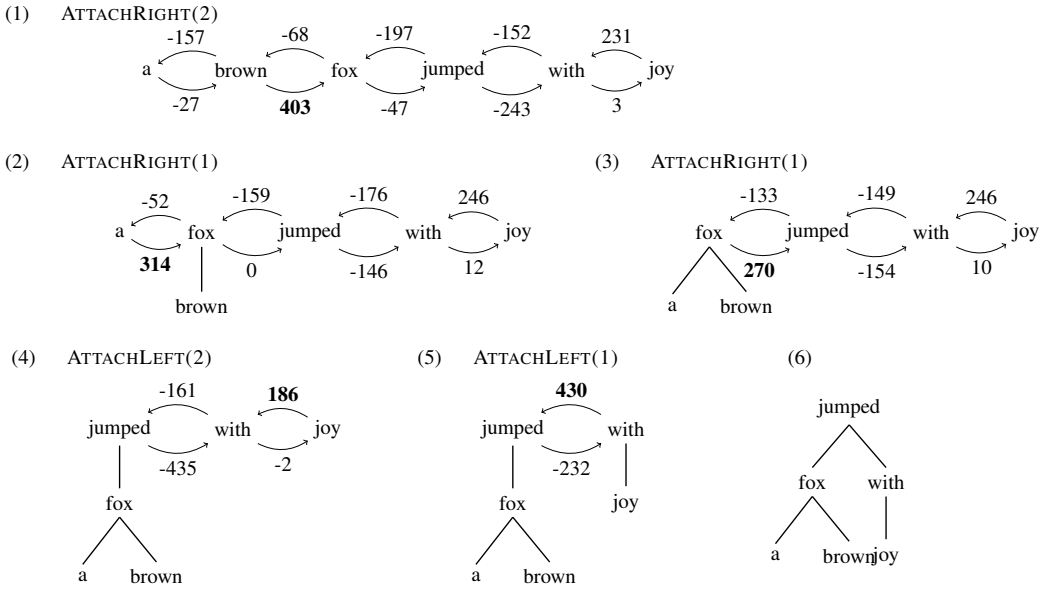


Figure 7.1: *Parsing the sentence "a brown fox jumped with joy". Top: sequence of states in the transition system. Bottom: graphical representation.*

While the transition-system is defined in terms of a list of words and a set of dependencies, it is useful to think of the algorithm as working with a list $p_0, \ldots, p_k$ of rooted dependency trees. The root of the tree p_i is the word l_i, and its structure is defined based on the dependencies in D.

Intuitively, the transition-system builds the (projective) parse tree bottom up, using two kinds of actions: **ATTLEFT$_i$** and **ATTRIGHT$_j$**. These actions are applied to a list of partial dependency-trees $P = p_0, \ldots, p_k$, which is initialized with the ROOT symbol followed by the n words of the sentence: ROOT $, w_1, \ldots, w_n$. Each action connects the heads of two neighboring structures, making one of them the parent of the other, and removing the daughter from the list of partial structures. In practice, the list contains just the head-words of the partial dependency-tree structures, and the structures themselves

are captured in the set of dependencies, D.

Figure 7.1 shows an example of parsing the sentence "a brown fox jumped with joy". The top part lists the transition sequence in terms of the L and D data structures, while the bottom part uses a graphical representation, in line with the list-of-partial-trees structure. The rounded arcs in the bottom part of the picture represent possible actions, while the numbers on the arcs represent the score assigned to each action, to be discussed later.

It is easy to see that for an n-word sentence, every sequence of actions terminates in at most n steps: the system is initialized with $n + 1$ items in L. Each of the possible actions remove one item from L, shortening it by 1. Thus, after n steps, $|L| = 1$. By definition, the actions cannot change the order of elements in L, nor can they remove the first element (which is initialized to be `ROOT`). Thus, $|L| = 1$ means that $L = \{\texttt{ROOT}\}$, and the system terminates after n steps.

7.3 Parsing algorithm

The transition-system framework suggests a very natural parsing algorithm:

Algorithm 7.1 Abstract transition-based parsing algorithm

1: **Input:** a sentence $w_1, \ldots, w_n$
2: **Initialize** a configuration
3: **while** not in a terminating configuration **do**
4: enumerate the possible actions
5: choose the best action
6: apply the chosen action to the current configuration
7: **return** dependency set D

The abstract algorithm is made concrete by plugging in the transition system defined above:[2]

[2]The parsing algorithm described below works in $O(n^2)$ time in the length of the sentence. We consider a more efficient version of the algorithm, working in time $O(n \log n)$, in Section 7.6 below.

Algorithm 7.2 Concrete non-directional stack-less transition-based parsing algorithm

1: **Input:** a sentence $S = w_1, \ldots, w_n$
2: **Initialize**: $D = \emptyset \quad L = \texttt{ROOT}, w_1, \ldots, w_n$
3: **while** $|L| > 1$ **do**
4: *// enumerate the possible actions*
5: actions $\leftarrow$ $\textbf{ATTRIGHT}_0, \ldots, \textbf{ATTRIGHT}_{|L|}, \textbf{ATTLEFT}_0, \ldots, \textbf{ATTLEFT}_{|L|}$
6: *// choose the best action*
7: bestAction $\leftarrow \underset{\text{act}_i \in \text{actions}}{\arg\max}\, score(\text{act}_i, D, L, S)$
8: *// apply the chosen action*
9: **if** bestAction = $\textbf{ATTRIGHT}_\textbf{i}$ **then**
10: $\langle parent, child \rangle = \langle L_i, L_{i+1} \rangle$
11: **else if** bestAction = $\textbf{ATTLEFT}_\textbf{i}$ **then**
12: $\langle parent, child \rangle = \langle L_{i+1}, L_i \rangle$
13: D.add($\langle parent, child \rangle$)
14: L.remove($child$)
15: **return** dependency set D

The only part of the parsing algorithm that remains unspecified is choosing the best action in line 7. The algorithm defines $score()$, a scoring procedure for actions and chooses the highest scoring action according to this procedure. The scoring can condition on the action ($act \in \{\textbf{ATTLEFT}, \textbf{ATTRIGHT}\}$) to be taken and its location (i), the list of yet to be connected head-words L, the already built structures (D), and the original sentence S. The $score()$ function is learned from data, and its form will be discussed in the next section. It is important to note that the scoring function reflects not only the correctness of an attachment, but also *the order* in which attachments should be made. For example, consider the attachments (brown,fox) and (joy,with) in Figure (7.1/1). While both are correct, the scoring function prefers the (adjective,noun) attachment over the (prep,noun) attachment. Moreover, the attachment (jumped,with), while correct, receives a negative score for the bare preposition "with" (Fig. (7.1/1) - (7.1/4)) and a high score once the verb has its subject and the PP "with joy" is built (Fig. (7.1/5)). Ideally, we would like to score easy and reliable attachments *higher* than harder, less likely attachments, thus performing attachments in order of confidence. This strategy allows us both to limit the extent of error propagation, and to make use of richer contextual information in later, more difficult attachments. Unfortunately, this kind of ordering information is not directly encoded in the data. We must, therefore, learn how to order the decisions.

I first describe the learning algorithm (section 7.4) and a feature representation (sec-

tion 7.5) that enables us to learn an effective scoring function.

7.4 Learning algorithm

Partial oracle The role of the training procedure is to learn a scoring function that performs well. For the left-to-right transition-based parsing systems described in the previous chapter, "performing well" is achieved by mimicking an oracle as best as possible. However, in our case an oracle is not available. Instead, we have access only to a partial oracle. Rather than providing the best sequence of actions, our partial oracle can tell us the set of valid actions at each parsing step, where a valid action is one that can lead to a correct parse. If we repeatedly apply the oracle and choose an action from its proposed set of valid actions, we are guaranteed to achieve a complete parse.

The partial oracle can be defined in terms of an isBad(act_i, L, D, S, G) function which tells us, for a configuration (L, D), a sentence S, an action act_i and a correct set of dependencies G, if applying the action at the given configuration will prevent us from obtaining the correct parse G. The set of valid actions is defined as all the possible actions for which isBad() returns $False$.

The isBad() function for our transition-system is defined in algorithm 7.3. In words, the function verifies that the proposed edge is indeed present in the gold parse (line 7) and that the suggested daughter already found all its own daughters (line 10).[3] The condition in line 10 is necessary because when attaching a $\langle parent, modifier \rangle$ pair, the modifier is removed from the list L and can not be used in any further steps. If the gold tree includes the edges $\langle a, b \rangle$ and $\langle b, c \rangle$, and the parser attaches $\langle a, b \rangle$, before it attaches $\langle b, c \rangle$, b is removed from L and the edge $\langle b, c \rangle$ could never be built.

[3] This is in line with the Arc-Standard parsing strategy of shift-reduce dependency parsers [101].

Algorithm 7.3 isBad(action, L, D, S, G)

// extract the dependency pair to be added by the action
if action = **ATTRIGHT$_i$** **then**
  $\langle parent, child \rangle = \langle L_i, L_{i+1} \rangle$
else if action = **ATTLEFT$_i$** **then**
  $\langle parent, child \rangle = \langle L_{i+1}, L_i \rangle$
// if the dependency does not exist in gold, it is bad
if $\langle parent, child \rangle \notin G$ **then**
  return True
// if the proposed child did not yet collect all of its children, it is bad
else if $\exists grand, \quad \langle child, grand \rangle \in G \wedge \langle child, grand \rangle \notin D$ **then**
  return True
else
  return False

Scoring model In this work, the $score()$ function is based on a linear scoring model:

$$score(\text{act}_i, L, D, S) = \vec{w} \cdot \phi(\text{act}_i, L, D, S)$$

where $\phi(\text{act}_i, L, D, S)$ is a feature representation and $\vec{w}$ is a weight vector. For brevity, I sometimes write ϕ_{act_i} to denote the feature representation extracted for action act at location i (L, D, and S are omitted in this shortcut but do take part in the computation). The job of the training procedure is to set the values of the weight vector $\vec{w}$ so that the resulting parser performs well.

The model is trained using a variant of the structured perceptron [34], similar to the algorithm of [127, 128]. As usual, I use parameter averaging to prevent the perceptron from overfitting.

The training algorithm is initialized with a zero parameter vector $\vec{w}$. The algorithm makes several passes over the data. At each pass, the training procedure given in Algorithm 7.4 is applied to every sentence in the training set.

Algorithm 7.4 Structured-perceptron EASYFIRST training for non-directional parser, over one sentence.

1: **Input:** sentence S, gold tree G, current $\vec{w}$, feature representation ϕ
2: **Output:** a weight vector $\vec{w}$
3: **Initialize**: $D = \emptyset \quad L = \texttt{ROOT}, w_1, \ldots, w_n$
4: **while** $|L| > 1$ **do**
5: *// enumerate the possible actions*
6: actions $\leftarrow$ $\textbf{ATTRIGHT}_0, \ldots, \textbf{ATTRIGHT}_{|L|}, \textbf{ATTLEFT}_0, \ldots, \textbf{ATTLEFT}_{|L|}$
7: validActions $\leftarrow \{\text{act}_i | \text{act}_i \in \text{actions} \wedge \neg isBad(\text{act}_i, L, D, S, G)\}$ *// partial oracle*
8: *// choose the best action*
9: bestChosen $\leftarrow \underset{\text{act}_i \in \text{actions}}{\arg\max}\ \vec{w} \cdot \phi(\text{act}_i, L, D, S)$
10: bestValid $\leftarrow \underset{\text{act}_i \in \text{validActions}}{\arg\max}\ \vec{w} \cdot \phi(\text{act}_i, L, D, S)$
11: **if** bestChosen $\notin$ validActions **then**
12: *// current best action is not valid: update weights and try again*
13: $\vec{w} \leftarrow \vec{w} + \phi(\text{bestValid}, L, D, S) - \phi(\text{bestChosen}, L, D, S)$
14: **else**
15: *// current best action is valid: apply it*
16: **if** bestChosen = $\textbf{ATTRIGHT}_\textbf{i}$ **then**
17: $\langle parent, child \rangle = \langle L_i, L_{i+1} \rangle$
18: **else if** bestChosen = $\textbf{ATTLEFT}_\textbf{i}$ **then**
19: $\langle parent, child \rangle = \langle L_{i+1}, L_i \rangle$
20: D.add($\langle parent, child \rangle$)
21: L.remove($child$)
22: **return** dependency set D

At training time, each sentence is parsed using the parsing algorithm and the current $\vec{w}$. Whenever an invalid action is chosen by the parsing algorithm, it is not performed (lines 11-13). Instead, the parameter vector $\vec{w}$ is updated by decreasing the weights of the features associated with the *invalid* action, and increasing the weights for the currently highest-scoring *valid* action. The algorithm then proceeds to parse the sentence with the updated values. The process repeats until a valid action is chosen.

Note that each single update does not guarantee that the next chosen action is valid, or even different than the previously selected action. Yet, this is still an aggressive update procedure: we do not leave a sentence until our parameters vector parses it correctly, and we do not proceed from one partial parse to the next until $\vec{w}$ predicts a correct location/action pair. However, as the best ordering, and hence the best attachment point is not known to us, we do not perform a single aggressive update step. Instead, the

aggressive update is performed incrementally in a series of smaller steps, each pushing $\vec{w}$ away from invalid attachments and toward valid ones. This way I integrate the search of confident attachments into the learning process.

7.5 Feature representation

The feature representation for an action can take into account the original sentence S, the current list of head-words L, as well as the entire parse history D.

In this section I describe a generic feature-set which can be used as the basis of other, language-specific feature-sets. This generic feature-set demonstrates the kinds of information that can be considered by the parser, and is shown experimentally to work reasonably well for parsing English.[4] It also serves as the basis for the feature set used in the Hebrew parser described in 8.

The feature set is composed of binary valued features, and each feature is conjoined with the type of action. The features are defined in terms of a list of dependency structures $P = p_0, \ldots, p_k$ which are headed by the words $l_0, \ldots, l_k$ in L, and whose structures are captured in the dependency set D.

When designing the feature representation, I kept in mind that the features should not only direct the parser toward desired actions and away from undesired actions, but they should also provide the parser with the means of choosing between several desired actions. I wanted the parser to be able to defer some desired actions until more structure is available and a more informed prediction can be made. This desire is reflected in the choice of features: some of the features are designed to signal to the parser the presence of possibly "incomplete" structures, such as an incomplete phrase, a coordinator without conjuncts, and so on.

When considering an action act_i, we limit ourselves to features of partial structures around the attachment point: $p_{i-2}, p_{i-1}, p_i, p_{i+1}, p_{i+2}, p_{i+3}$, that is the two structures to be attached by the action (p_i and p_{i+1}), and the two neighboring structures on each side[5].

While these features encode local context, it is local in terms of syntactic structure, and not purely in terms of sentence surface form. This feature set lets the parser capture some, though not all, long-distance relations.

The amount of local context which is used for each score calculation is referred to as the *window size* of the feature function. When deciding an attachment between list

[4]The feature-set is not designed to be optimal, but to demonstrate the type of structural and contextual information that can be efficiently taken into account by the EasyFirst algorithm. A followup to this work [139] extends the feature set presented here and produces state-of-the-art accuracies for English parsing using EasyFirst.

[5]The sentences are padded from each side with sentence delimiter tokens.

items p_i and p_{i+1}, the feature function takes into account two list items to each side of the attachment point (from p_{i-2} to p_{i+1+2}), that is a window size of 2 to each side. The window size of the feature function has a direct effect on parsing time, as described in Section 7.6 below. I find a window size of 2 to achieve a good balance between feature-function expressiveness and parser's efficiency.

For a partial structure p, I use w_p to refer to the head word form, t_p to the head word POS tag, and lc_p and rc_p to the POS tags of the left-most and right-most child of p, respectively.

All the prepositions (IN) and coordinators (CC) are lexicalized: for them, t_p is in fact $w_p t_p$. This lexicalization is done because in many cases the parts-of-speech which are assigned to prepositions and coordinators in Treebanks are too coarse to capture grammatical distinctions which are relevant to syntactic disambiguation. For example, the coordination word *and* can be used to conjoin large spans of a sentence, while the coordinating symbol "&" is used mostly for local structures. Similarly, the preposition *of* usually attaches to nouns and not to verbs. In addition, the list of prepositions and coordinators is small enough for the statistical model to learn robust statistics regarding the usage patterns of individual words without needing to back-off to the part-of-speech tag for unknown words.

For coordinators (CC), t_p is dynamically extended with more information. Coordinators are lexicalized, and include the POS of the current left and right children of the coordinator (for CC, t_p is $w_p t_p lc_p rc_p$). This encoding of coordination helps the model distinguish between coordinators with zero, one or more children, as well as provides the model with the means to learn about a preference for the coordination of likes.[6]

I define *structural*, *unigram*, *bigram*, *3gram*, *4gram* and *pp-attachment* features.

The *structural* features are: the length of the structures (len_p), whether the structure is a word (contains no children: nc_p), and the surface distance between structure heads ($\Delta_{p_i p_j}$). The *Ngram* features are adapted from the feature set for left-to-right Arc-Standard dependency parsing (described in [63]), which I extended to include the structure on both sides of the proposed attachment point, and features making use of a richer sequential context.

In the case of unigram features, I added features that specify the POS of a word and its left-most and right-most children. These features provide the non-directional model with means to prefer some attachment points over others based on the types of structures already built. In English, as well as many other languages, the left- and rightmost POS-tags are good indicators of constituency.

[6]This only scratches the surface in terms of dynamic features. See [139, supplementary material] for an extended selection of dynamic features useful for parsing English.

Structural	
for p in $p_{i-2}, p_{i-1}, p_i, p_{i+1}, p_{i+2}, p_{i+3}$	len_p , nc_p
for p,q in (p_{i-2}, p_{i-1}),(p_{i-1}, p_i),(p_i, p_{i+1}),($p_{i+1}, pi+2$),(p_{i+2}, p_{i+3})	Δ_{qp} , $\Delta_{qp} t_p t_q$
Unigram	
for p in $p_{i-2}, p_{i-1}, p_i, p_{i+1}, p_{i+2}, p_{i+3}$	t_p , w_p , $t_p lc_p$, $t_p rc_p$, $t_p rc_p lc_p$
Bigram (including left/right children)	
for p,q in (p_i, p_{i+1}),(p_i, p_{i+2}),(p_{i-1}, p_i),(p_{i-1}, p_{i+2}),(p_{i+1}, p_{i+2})	$t_p t_q$, $w_p w_q$, $t_p w_q$, $w_p t_q$ $t_p t_q lc_p lc_q$, $t_p t_q rc_p lc_q$ $t_p t_q lc_p rc_q$, $t_p t_q rc_p rc_q$ $t_p t_q lc_p$, $t_p t_q rc_p$ $t_p t_q lc_q$, $t_p t_q rc_q$
3gram	
$t_{p_i} t_{p_{i+1}} rc_{p_{i+1}} t_{p_{i+2}}$, $t_{p_i} t_{p_{i+1}} lc_{p_i} t_{p_{i-1}}$, $t_{p_{i+1}} rc_{p_{i+1}} t_{p_{i+2}} t_{p_{i+3}}$, $t_{p_i} lc_{p_i} t_{p_{i-1}} t_{p_{i-2}}$	
4gram	
$t_{p_{i-1}} t_{p_i} t_{p_{i+1}} t_{p_{i+2}}$, $t_{p_i} t_{p_{i+1}} rc_{p_{i+1}} t_{p_{i+2}} t_{p_{i+3}}$, $t_{p_{i-2}} t_{p_{i-1}} t_{p_i} lc_{p_i} t_{p_{i+1}}$	
PP-Attachment	
if p_i is a preposition	$w_{p_{i-1}} w_{p_i} rc_{p_i}$, $t_{p_{i-1}} w_{p_i} rcw_{p_i}$
if p_{i+1} is a preposition	$w_{p_{i-1}} w_{p_{i+1}} rc_{p_{i+1}}$, $t_{p_{i-1}} w_{p_{i+1}} rcw_{p_{i+1}}$ $w_{p_i} w_{p_{i+1}} rc_{p_{i+1}}$, $t_{p_i} w_{p_{i+1}} rcw_{p_{i+1}}$
if p_{i+2} is a preposition	$w_{p_{i+1}} w_{p_{i+2}} rc_{p_{i+2}}$, $t_{p_{i+1}} w_{p_{i+2}} rcw_{p_{i+2}}$ $w_{p_i} w_{p_{i+2}} rc_{p_{i+2}}$, $t_{p_i} w_{p_{i+2}} rcw_{p_{i+2}}$

Figure 7.2: *Feature templates for the easy-first non-directional parser*

The *pp-attachment* features are similar to the bigram features, but fire only when one of the structures is headed by a preposition (IN). These features are more lexicalized than the regular bigram features, and include also the word-form of the right-most child of the PP (rcw_p). This encoding of prepositions should help the model learn lexicalized attachment preferences such as (hit, with-bat).
Figure 7.2 enumerates the feature templates in the generic feature set.

7.6 Computational complexity and efficient implementation

I now turn to analyse the runtime complexity of the EasyFirst parsing algorithm.

The analysis considers the "standard" computational model used in computer science, assigning a fixed cost to each computational unit, as well as a computational model which is concerned only with the number of feature-extraction and dot-product operations (the *scoring* operations), which dominate the runtime of polynomial time discriminative parsers.

I begin by showing that the EasyFirst parsing algorithm described in Algorithm 7.2 runs in $O(n^2)$ in the length of the sentence. I then suggest a more efficient implementation and prove that it works in $O(n \log n)$ with respect to the standard units of

computation, and in $O(n)$ with respect to the number of scoring operations.

As is common, the analyses all assume that the feature-set is fixed and that the number of active features at each stage is bounded by a small constant, and thus feature-extraction and dot-product are constant-time operations.[7]

Claim 1 : The parsing algorithm in Algorithm 7.2 terminates in $O(n^2)$ times, and uses $O(n^2)$ feature-extraction and dot-product operations.

Lemma 1.1 : For a sentence of n words, the algorithm 7.2 terminates after exactly n iterations.

Proof : for an n-words sentence, the algorithm is initialized (line 2) with the list L containing $n+1$ items (the words of the sentence + `ROOT` symbol). It then iterates until L contains a single item (line 3). Each iterations removes exactly one item from L (line 14), and no operation adds items to L. Thus, the algorithm terminates after n iterations.

Lemma 1.2 : Each iteration of algorithm 7.2 takes $O(n)$ time.

Proof : Line 5 enumerates the possible operations. Each element in L yields at most two operations into the list ($\textbf{ATTLEFT}_\textbf{i}$ and $\textbf{ATTRIGHT}_\textbf{i}$), and L is bounded by $n+1$, so the enumeration is performed in $O(n)$ time. The argmax in line 7 involves computing a score for each of the $O(n)$ actions, and finding the maximal scoring one. Scoring each item is done in constant time using one feature-extraction and one dot-product operation. Finding the maximum of $O(n)$ items is performed in $O(n)$ time. Thus, the argmax operations is performed in $O(n)$ time. The other operations are trivially performed in a constant time. Thus, no step in the loop consumes more than $O(n)$ operations (either basic operations or feature-extraction and dot-product operations).

Claim 1 follows naturally from the combination of lemmas 1.1 and 1.2.

I now turn to show that the algorithm could be improved to support a runtime of $O(n \log n)$ in terms of basic operations, and $O(n)$ in terms of feature-extraction and dot-product operations.

[7] While these operations are indeed fixed time in practice, the constants involved can be quite large and dominate the runtime performance. However, these costs are in many ways orthogonal to the discriminative parsing algorithm being used. For this reason, I believe it is more informative to measure the runtime complexity of parsing algorithms by means of the number of feature-extractions and dot-products operations involved, as I do below.

Note that an action act$_i$ at in a given list L is uniquely identified by the $\langle parent, child \rangle$ pair induced by the action ($\langle L_{i+1}, L_i \rangle$ for **ATTLEFT$_i$**, and $\langle L_i, L_{i+1} \rangle$ for **ATTRIGHT$_i$**). This representation for actions does not rely on the list indices i which change between iterations. I refer to this representation as $signature(\text{act}_i)$.

The main observation is that most of the possible actions and their scores remain the same from one iteration of the algorithm to the next. Specifically, after the application of an **ATTRIGHT$_i$** or **ATTLEFT$_i$** action (and considering the indices i before removing element L_i from L), the actions $\langle L_{i+1}, L_i \rangle$, $\langle L_i, L_{i+1} \rangle$, $\langle L_{i-1}, L_i \rangle$ and $\langle L_i, L_{i-1} \rangle$ are no longer available (because L_i is about to be removed from L), and are replaced by the actions $\langle L_{i+1}, L_{i-1} \rangle$, $\langle L_{i-1}, L_{i+1} \rangle$. The other possible actions remain unchanged.

Similarly, the scores of most actions also remain constant between consecutive iterations. Only scores of actions involving L_{i-k},...,L_{i+1+k} change after an application of an action involving L_i, where k is the window size used in the feature extraction mechanism.

Thus, at each stage of the algorithm we can maintain the list of possible actions and their scores by:

1. Associating scores with action signatures of the form $\langle parent, child \rangle$.
2. Maintain a data structure associating scores to actions.
3. After application of an action involving L_i, let $obsolete$ actions be $\langle L_i, L_{i+1} \rangle$, $\langle L_{i+1}, L_i \rangle$, $\langle L_i, L_{i-1} \rangle$, $\langle L_{i-1}, L_i \rangle$, let new actions be $\langle L_{i-1}, L_{i+1} \rangle$,$\langle L_{i+1}, L_{i-1} \rangle$, and let $rescored$ actions be actions involving L_{i-k},...,L_{i+1+k}.
4. Remove the scores for the actions in $obsolete$ and $rescored$ from the data structure of scored actions.
5. After performing the action, recalculate and add scores for the actions in $rescored$ and new.

A hashmap is an appropriate data structure to map actions to scores. Using a hashmap allows updating the scores in constant time, but then finding the maximum score becomes an $O(n)$ operation. A better option is, therefore, to combine a hashmap and an AVL: the hashmap maps actions to nodes in an AVL tree, and nodes in the AVL tree contain a pair (action, score) and are ordered by score. With this combined data structure, both score update (adding a node and removing a node) and finding the maximal scoring item are $O(\log n)$ operations.

The updated algorithm is given below:

Algorithm 7.5 Efficient non-directional stack-less transition-based parsing algorithm

Input: a sentence $S = w_1, \ldots, w_n$
Parameter: feature-extractor window-size k
Initialize: $D = \emptyset \quad L = \texttt{ROOT}, w_1, \ldots, w_n$
// enumerate the possible actions
actions $\leftarrow$ **ATTRIGHT$_0$**,...,**ATTRIGHT$_{|L|}$**,**ATTLEFT$_0$**,...,**ATTLEFT$_{|L|}$**
// Calculate action scores
for act$_i$ in actions **do**
  scoredActions.$add(signature(\text{act}_i), score(\text{act}_i, D, L, S))$
while $|L| > 1$ **do**
  // choose the best action
  bestAction $\leftarrow scoredActions.getMax()$
  // apply the chosen action
  if bestAction = **ATTRIGHT$_i$** **then**
    $\langle parent, child \rangle = \langle L_i, L_{i+1} \rangle$
  else if bestAction = **ATTLEFT$_i$** **then**
    $\langle parent, child \rangle = \langle L_{i+1}, L_i \rangle$
  D.add($\langle parent, child \rangle$)
  $obsolete \leftarrow calculate\_obsolete\_actions(L, i)$
  $new \leftarrow calculate\_new\_actions(L, i)$
  $rescored \leftarrow calculate\_rescored\_actions(L, i, k)$
  L.remove($child$)
  for act$_i$ in $obsolete \cup rescored$ **do**
    scoredActions.$remove(signature(\text{act}_i))$
  for act$_i$ in $new \cup rescored$ **do**
    scoredActions.$add(signature(\text{act}_i), score(\text{act}_i, D, L, S))$
return dependency set D

Claim 2 : Algorithm 7.5 works in time $O(n \log n)$ with respect to the number of basic computational steps, and in time $O(n)$ with respect to feature-extraction and dot-product operations.

Lemma 2.1 : For a sentence of n words, the algorithm 7.5 terminates after exactly n iterations.

Proof : the proof for lemma 1.1 applies to lemma 2.1 as well.

Lemma 2.2 : Each iteration of algorithm 7.5 can be performed in $O(\log n)$ computation steps.

Proof : The $scoredAction$ component is implemented using an AVL tree with nodes containing a pair (action, score) and sorted by score, and a hashmap mapping action signatures to the corresponding nodes in the AVL; the $remove$, add and $getMax$ operations are performed in $O(\log n)$ time. By construction, each of the new, $obsolete$ and $rescored$ action sets are bounded in size by $2k + 1$ ($|new| = 2$, $|obsolete| = 4$, $|rescored| = 2k + 1$), and can be enumerated in constant time (given a fixed window-size k). The sets are generated anew at each iteration, and their union is also bounded in size by $O(k)$. Thus, at most $O(k) = O(1)$ add, max and $remove$ operations are performed in a single iteration, resulting in an $O(\log n)$ cost per iteration.

Lemma 2.3 : Each iteration of algorithm 7.5 takes $O(1)$ feature-extraction and dot-product operations.

Proof : Feature extraction and dot-product computation take place inside $score$, which is called at line 25. This $score$ is called once for each element in the union of the new and $rescored$ action sets, and the union size is bounded by 2k+1, thus $O(1)$ scoring operations take place in each iterations (for a fixed window-size k, where $2k = 4 \ll n$ for most sentences).

The initialization outside of the main loop takes $O(n)$ scoring operations and $O(n \log n)$ computational steps. Together with lemmas 2.1, 2.2 and 2.3, **Claim 2** follows.

Note that the dominating factor in polynomial-time discriminative parsers, is by far the feature-extraction and dot-product calculation. It makes sense to compare parser complexity in terms of these operations only.[8]

Table 7.1 compares the complexity of the easy-first parser to other dependency parsing frameworks.

In terms of feature extraction and score calculation operations, the easy-first algorithm has the same cost as traditional shift-reduce (MALT) parsers, and is an order of magnitude more efficient than graph-based (MST) parsers. Beam-search decoding for left-to-right parsers [154] is also linear, but has an additional linear dependence on the beam-size. The reported results in [154] use a beam size of 64, compared to the constant of

[8]Indeed, in my implementation I do not use AVL+hash-map, opting instead to find the argmax using a simple $O(n)$ *max* operation. This $O(n^2)$ algorithm is as fast as, or faster in practice than, the AVL-based algorithm, as both are dominated by the $O(n)$ feature extraction, while the cost of the $O(n)$ *max* calculation for sentences of reasonable lengths is negligible compared to the constants involved in the tree maintenance.

Parser	Runtime	Features / Scoring
MALT	$O(n)$	$O(n)$
MST	$O(n^3)$	$O(n^2)$
MST2	$O(n^3)$	$O(n^3)$
BEAM	$O(n * beam)$	$O(n * beam)$
EASYFIRST (This Work)	$O(n \log n)$	$O(n)$

Table 7.1: *Complexity of different parsing frameworks.* MST*: first order MST parser;* MST2*: second order MST parser;* MALT*: shift-reduce left-to-right parsing;* BEAM*: beam search parser, as in [154].*

$2k = 6$ used by the easy-first parser with the feature representation described in this chapter.[9]

My python-based implementation[10] parses about 40 tagged sentences per second on an Intel-based MacBook laptop.

A note on parallelization The algorithm can be parallelized. The initial stage of the algorithm performs n feature-extraction and scoring operations, and subsequent steps perform $2k$ feature-extraction and scoring operations. These extraction and scoring operations are independent of each other, and can be performed concurrently. Scoring requires access to the weight vector $\vec{w}$, which is kept constant during parsing. A separate copy of this vector must be held by each process/thread in order to keep them completely independent and avoid contention, or, if memory usage is a concern (since $\vec{w}$ is a very high-dimensional vector), a single read-only copy of the vector can be shared in a lock-free data structure without affecting performance.

7.7 Evaluation on English

To demonstrate the general applicability of the parsing system, I report the results of some experiments on dependency parsing of English.

Here, the parser is evaluated using the WSJ Treebank. The trees were converted to dependency structures with the Penn2Malt conversion program,[11] using the head-

[9]Note that the effect of the $2k = 6$ constant can be reduced even further using somewhat more sophisticated data structures and a modest amount of bookkeeping. In the current implementation (and analysis) each action requires $2k$ complete feature-extraction and scoring operations. However, a given action affects only a fraction of the feature values of actions related to neighboring structures. With some clever bookkeeping, one could track the features whose values have a potential of actually changing due to an action, and only extract and rescore these features, speeding up the parsing process.

[10]The perceptron is implemented in a `C` extension module.

[11]`http://w3.msi.vxu.se/~nivre/research/Penn2Malt.html`

finding rules from [151].[12]

Sections 2-21 are used for training, Section 22 for development, and Section 23 as the final test set.

The text is automatically POS tagged using a trigram HMM-based POS tagger prior to training and parsing. Each section is tagged after training the tagger on all other sections. The tagging accuracy of the tagger is 96.5 for the training set and 96.8 for the test set. While better taggers exist, I believe that the simpler HMM tagger overfits less, and is more representative of the tagging performance on non-WSJ corpus texts.

Parsers I evaluate our parser against the transition-based MALT parser and the graph-based MST parser. I use version 1.2 of MALT parser[13], with the settings used for parsing English in the CoNLL 2007 shared task. For the MST parser[14], I use the default first-order, projective parser settings, which provide state-of-the-art results for English. All parsers are trained and tested on the same data. The parser is trained for 20 iterations.

Evaluation measures I evaluate the parsers using the UAS, Root Accuracy and Unlabeled Exact Match measured, as defined in Chapter 6.1. Unlike most previous work on English dependency parsing, I *do not* exclude punctuation marks from the evaluation.

Results are presented in Table 7.2. The non-directional easy-first parser substantially outperforms the left-to-right greedy MALT parser as well as the globally optimized edge-factored MST parser in terms of UAS, and even more substantially outperforms both parsers in terms of exact match. The globally optimized MST parser is better in root-prediction.

I evaluated the parsers also on the English dataset from the CoNLL 2007 shared task. While this dataset is also derived from the WSJ Treebank, two important aspects distinguish it from the previous dataset in two important aspects: it is much smaller in size, and it is created using a different conversion procedure, which is more linguistically adequate. For these experiments, I use the dataset POS tags, and the same parameters as in the previous set of experiments: the non-directional easy-first parser is trained for 20 iterations, with the same feature set. The CoNLL dataset contains some non-projective

[12] While other and better conversions exist (see, *e.g.*, [72, 123]), this conversion heuristic is still the most widely used. Using the same conversion facilitates comparison with previous works.

[13] `http://maltparser.org/dist/1.2/malt-1.2.tar.gz`

[14] `http://sourceforge.net/projects/mstparser/`

Parser	UAS	Root	Exact Match
MALT	88.36	87.04	34.14
MST	90.05	93.95	34.64
EASYFIRST (this work)	90.40	91.84	39.56

Table 7.2: *Unlabeled dependency accuracy on PTB Section 23, automatic POS-tags, including punctuation.*

Parser	UAS	Root Accuracy	Exact Match
MALT	85.82	87.85	24.76
MST	89.08	93.45	24.76
EASYFIRST (this work)	89.20	93.45	30.37

Table 7.3: *Unlabeled dependency accuracy on CoNLL 2007 English test set, including punctuation.*

constructions. MALT and MST deal with non-projectivity. For the easy-first parser, I projectivize the training set prior to training using the procedure described in [23].

Results are presented in Table 7.3.

While all models suffer from the move to the smaller dataset and the more challenging annotation scheme, the overall story remains the same: the EASYFIRST parser is better than both MALT and MST in terms of root prediction accuracy, and also much better in terms of producing complete correct parses.

That the EASYFIRST parser has similar accuracy but more exact matches than the MST parser can be explained by it being a deterministic parser, and hence still vulnerable to error propagation: after it has erred once, it is likely to do so again, resulting in low accuracies for some sentences. However, due to the easy-first policy, it manages to parse many sentences without a single error, which leads to higher exact-match scores. The EASYFIRST parser avoids error propagation by not making the initial error. On average, the EASYFIRST parser manages to assign correct heads to over 65% of the tokens before making its first error.

The both the MST and the EASYFIRST parser would have ranked 5^{th} in the shared task. The better ranking systems in the shared task are either higher-order global models, beam-search-based systems, or ensemble-based systems, all of which are more complex and less efficient than the EASYFIRST parser.

7.7.1 Parse diversity

The parses produced by the EASYFIRST parser differ from the parses produced by the graph-based and left-to-right parsers. To demonstrate this difference, I performed an

Combination	UAS	Exact Match
Penn2Malt, Train 2-21, Test 23		
MALT+MST	92.29	44.03
EASYFIRST+MALT	92.37	46.44
EASYFIRST+MST	92.75	46.06
EASYFIRST+MST+MALT	93.65	50.62
CoNLL 2007		
MALT+MST	91.50	33.64
EASYFIRST+MALT	91.18	36.91
EASYFIRST+MST	91.98	35.04
EASYFIRST+MST+MALT	92.76	40.66

Table 7.4: *Parser combination with Oracle, choosing the highest scoring parse for each sentence of the test-set.*

Oracle experiment, in which the outputs of several parsers are combined by choosing, for each sentence, the parse with the highest score. Results are presented in Table 7.4.

A non-oracle blending of MALT+MST+EASYFIRST using Sagae and Lavie's (2006) simplest combination method assigning each component the same weight, yields an accuracy of 90.6% on the CoNLL 2007 English dataset, making it the highest scoring system among the participants.

7.7.2 Error analysis / limitations

When investigating the POS category of mistaken instances, it is apparent that for all parsers, nodes with structures of depth 2 and more which are assigned an incorrect head are predominantly PPs (headed by 'IN'), followed by NPs (headed by 'NN'). All parsers have a hard time dealing with PP attachment, but MST parser is better at it than EASYFIRST, and both are better than MALT.

Looking further at the mistaken instances, I notice a tendency of the PP mistakes of the EASYFIRST parser to involve, before the PP, an NP embedded in a relative clause. This reveals a limitation of the parser: recall that for an edge to be built, the child must first acquire all its own children. This means that in case of relative clauses such as "I saw the boy [who ate the pizza] with my eyes", the parser must decide whether the PP "with my eyes" should be attached to "the pizza" or not *before* it is allowed to build parts of the outer NP ("the boy who…"). In this case, the verb "saw" and the noun "boy" are both outside of the sight of the parser when deciding on the PP attachment, and it is forced to make a decision in ignorance, which, in many cases, leads to mistakes. The globally optimized MST does not suffer as much from such cases.

7.8 Related work

Deterministic shift-reduce parsers are restricted by a strict left-to-right processing order. Such parsers can rely on rich syntactic information on the left, but not on the right, of the decision point. They are forced to commit early, and suffer from error propagation. The non-directional easy-first parser addresses these deficiencies by discarding the strict left-to-right processing order, and attempting to make easier decisions before harder ones. Other methods of dealing with these deficiencies were proposed over the years:

Several Passes Yamada and Matsumoto's [151] pioneering work introduces a shift-reduce parser which makes several left-to-right passes over a sentence. Each pass adds structure, which can then be used in subsequent passes. Sagae and Lavie [122] extend this model to alternate between left-to-right and right-to-left passes. Yamada and Matsumoto's model is similar to mine, in that it attempts to defer harder decisions to later passes over the sentence, and allows late decisions to make use of rich syntactic information (built in earlier passes) on both sides of the decision point. However, the model is not explicitly trained to optimize attachment ordering, has an $O(n^2)$ runtime complexity, and produces results which are inferior to current single-pass shift-reduce parsers.

Beam Search Several researchers dealt with the early-commitment and error propagation of deterministic parsers by extending the greedy decisions with various flavors of beam-search [121, 137, 154]. This approach works well and produces highly competitive results. Beam search can be incorporated into my parser as well, though it is likely to hamper its efficiency significantly. I leave this investigation to future work.

Strict left-to-right ordering is also prevalent in sequence tagging. Indeed, one major influence on my work is Shen *et.al.*'s bi-directional POS-tagging algorithm [128], which combines a perceptron learning procedure similar to the one presented in this chapter with beam search to produce a state-of-the-art POS-tagger, which does not rely on left-to-right processing. Shen and Joshi [127] extend the bidirectional tagging algorithm to LTAG parsing, with good results. I build on that work and present a concrete and efficient greedy non-directional easy-first dependency parsing algorithm.

Structure Restrictions Eisner and Smith [45] propose improving the efficiency of a globally optimized parser by posing hard constraints on the lengths of arcs it can produce. Such constraints pose an explicit upper bound on parser accuracy.[15] My parsing model does not impose such restrictions. Shorter edges are arguably easier to predict,

[15] In [40], constraints are chosen "to be the minimum value that will allow recovery of 90% of the left (right) dependencies in the training corpus".

and the parser builds them early in time. However, it is also capable of producing long dependencies at later stages in the parsing process. Indeed, the distribution of arc lengths produced by the parser is similar to those produced by the MALT and MST parsers.

7.9 Chapter summary

This chapter presents and evaluates the EASYFIRST parser: a *non-directional* deterministic dependency parsing algorithm, which is not restricted by the left-to-right parsing order of other deterministic parsers. Instead, it works in an *easy-first* order. This strategy allows using more context at each decision. The parser learns both *what* and *when* to connect. The parsing algorithm is evaluated on two instances of English parsing, and is shown to substantially outperforms a left-to-right deterministic algorithm as well as a globally optimized parsing algorithm. While the algorithm still lags behind globally optimized parsing algorithms in terms of root prediction, it is much better in terms of exact match, better at UAS, and much faster. The work of [139] demonstrates that by using a better-tuned feature-set, the easy-first parsing algorithm can provide even higher accuracies for English parsing.

The easy-first parsing framework can easily and efficiently utilize more structural information than globally optimized parsers, and allows the definition of rich and complex features. This capability will prove to be especially important in the next chapter, where I use it to define specialized features to account for various morphological agreement patterns and phenomena specific to Modern Hebrew.

Moreover, the easy-first parser produces different structures than those produced by both left-to-right and globally optimized parsers, making it a good candidate for inclusion in an ensemble system.

Chapter 8

Hebrew Dependency Parsing

The previous chapter presented the EASYFIRST parsing algorithm.

In this chapter I evaluate the applicability of this algorithm for dependency parsing of Modern Hebrew. One key aspect of the algorithm is the flexibility it allows in feature-set design. I make use of this flexibility by defining features which target specific morphological agreement patterns. These features are then shown to improve parsing accuracy. The algorithm presented in the previous chapter produces unlabaled dependency structures. I present an external edge-labaler which can be trained to add dependency labels to unlabeled parse-trees.

8.1 Architecture

Like all current dependency parsing systems, the Hebrew Dependency Parser works using a pipeline approach: a POS-tagger (See section 4.2.3) is used to perform word-segmentation and morphological disambiguation. The output of the tagger is then used as input to the parser, which considers it as fixed.

8.1.1 Justification for a pipeline architecture

In Chapters 4 and 10 I argue that word-segmentation and syntactic parsing are two aspects of the same task, and that the two should be performed jointly. The same arguments apply also when working with dependency representations. Nonetheless, for the work presented in this chapter I chose to use a pipeline architecture in which word-segmentation (and tagging) is performed prior to dependency parsing.

The justification for this choice stems from the desire for computational efficiency: deterministic parsing systems have very little to gain from performing the task jointly,

and current search-based algorithms will suffer too much in runtime to make them worthwhile.

Limited potential gain for deterministic systems The shift-reduce algorithms and the EASYFIRST algorithm I proposed in the previous chapter are deterministic, and do not perform search. Thus, they are not able to trade later decisions with earlier ones. Performing joint segmentation and parsing in such systems provides only a marginal potential benefit over the pipeline approach: while some segmentation decisions could potentially condition on and benefit from existing syntactic structure, what is expected in practice is that the system will first segment/tag parts of the text, and then build syntactic structures on top of that. While the syntactic parsing and word-segmentation processes will be interleaved, there will be no real flow of information between the tasks, therefore resulting in no benefit over the pipeline architecture.

The small potential benefit should be weighted against the cost in efficiency. While the EASYFIRST algorithm could be adapted to work over a lattice, the change will substantially increase the number of possible actions at each step, rendering it inefficient. A similar argument holds for the left-to-right transition-based shift-reduce parsers – while the runtime penalty for the shift-reduce parser will be smaller, so is the expected gain in performance.

Prohibitive cost for search-based systems The first-order edge-factored graph-based parsing algorithm performs global search in $O(n^3)$ time with a small constant. This efficiency is achieved by using dynamic programming to find the highest scoring consistent set of n arcs from the set of $O(n^2)$ available scored arcs. The arc scores are computed prior to the dynamic programming process, and are considered as fixed by the dynamic-programming algorithm. Many of the features rely on part-of-speech information, which is assumed to be fixed. Doing tagging jointly with parsing is possible in the dynamic programming algorithm[44], but edge factorization precludes the scoring function from conditioning on the tags of neighboring structures. Performing tagging and parsing jointly while allowing the same set of features in the scoring function renders the parsing algorithm intractable [92]. As an approximation, one could use a preprocessing POS-tagging stage and run a parsing+tagging algorithm on top of the results: the precomputed POS-tags are used in the feature-function for score calculation, but the final parse is allowed to suggest alternative POS-tags. Such an approach is taken by [85]. However, the empirical benefits of the method are limited, and the addition of tagging to the edge-factored model makes the grammar constant much larger, which significantly degrades runtime performance. In a preliminary experiment in which I made

the parser choose one out of five tags for the word "and" and use the assigned tags for all other words, parsing time was about 4 times slower. This is expected to grow considerably with more tags. Extending the parsing algorithm to perform not only tagging but also segmentation by using a lattice is technically possible, but the runtime is expected to be much worse than the runtime for mere POS-tagging and parsing.

8.1.2 Lexicon/syntax separation

The pipeline approach does provide a good fit for the lexical/syntactic separation I argue for in the introduction chapter.

While the parser has access to lexical features, many of its features are based on the morphological information provided by the tagger, on which it can fall back when parsing words it has not seen in training. As discussed in Chapter 4, the Treebank is small, and its lexical coverage is limited. Meanwhile, the tagger I use for providing word-segmentation, POS-tagging, and additional morphological information, was trained based on several tens of millions of tokens, is making use of a comprehensive lexicon, and its performance is robust across domains (the tagger is discussed in section 4.2.3).

8.2 Data and baseline experiments

Both the transition-based MALT and graph-based MST parsers achieved high scores in the CoNLL 2006 and 2007 dependency-parsing shared tasks [19, 103]. These shared tasks were about building multilingual parsing systems – the parsers were evaluated on test sets in several languages (the parsers were trained and tested separately for each language). Thus, these parsers represent the best in general-purpose parsing systems, and are expected to perform well also on the Hebrew Treebank.

I begin by training and testing the MALT and MST parsers, as well as the EASY-FIRST parser with generic features described in the previous chapter, on the Hebrew Dependency Treebank.

Data and experimental setup I follow the established train-test-dev split of the Treebank for all the experiments in this chapter. Specifically, sentences 484-5724 of the Hebrew Dependency Treebank are used for training, and sentences 0-483 are used for development and for reporting scores of all intermediate results. Results on the test set (sentences 5725-6220) are reported at the end of the chapter.

The data in the Treebank is segmented and POS-tagged. Training and testing is performed either on gold segmentation and tags, on gold segmentation and predicted tags, or on the realistic scenario in which both segmentation and tags are predicted. In the predicted segmentation and/or tagging settings, tagging and segmentation were performed using the morphological disambiguator described in Chapter 4 prior to parsing.

Parsers and parsing models I use the freely available implementation of MALT and MST parsers, using the default settings for each of the parsers. In addition, I use my own implementation of an arc-standard shift reduce parser (ARCSTN), with the feature set described in [63].

For MALT, I experiment both with the default feature representation (MALT) and the feature representation used for parsing Arabic in CoNLL 2006 and CoNLL 2007 multilingual dependency parsing shared tasks (MALT-ARA), the latter of which makes use of some morphological information.

For the MST parser, I experimented with first-order (MST1) and second-order (MST2) models. The MST parser has an out-of-the-box generic mechanism for using morphological features. The parsers were trained and tested either with or without the morphological features.

I also report the results of using the EASYFIRST parser with the generic features described in the previous chapter.

8.2.1 Baseline results

The first set of experiments (results in Table 8.1) establishes the upper bound for the parsing models, by providing gold segmentation, POS-tags and morphological information both in training and in parsing time. The results are encouraging, ranging from 80.7% to 85.96% UAS. However, when the models trained on gold data are applied to text with predicted POS-tags, results drop considerably, by 3 to 4.5 UAS points.

Parser	Test Data	UAS	Root	Exact Match
MALT	Gold Seg+Tag	80.7	67.7	19.6
	Predicted Tag	76.4	63.0	16.1
	All Predicted	72.2	57.4	14.7
ARCSTN	Gold Seg+Tag	83.8	76.6	25.5
	Predicted Tag	79.7	70.6	20.5
	All Predicted	75.1	69.0	18.0
MST1	Gold Seg+Tag	84.5	82.8	22.7
	Predicted Tag	81.2	76.2	19.5
	All Predicted	76.0	75.3	13.7
MST2	Gold Seg+Tag	85.6	84.0	27.7
	Predicted Tag	82.4	79.1	21.9
	All Predicted	77.0	77.5	15.5
EASYFIRST	Gold Seg+Tag	85.96	82.4	28.2
	Predicted Tag	82.30	77.0	23.6
	All Predicted	77.75	74.4	20.4

Table 8.1: *Training on gold segmentation and tagging*

Training on data with predicted tags For the next experiment (results in Table 8.2), training was performed on predicted tags[1], and parsing was performed on either predicted tags or predicted tags and segmentation.

Parser	Test Data	UAS	Root	Exact Match
MALT	Gold Seg	79.6	70.8	19.3
	All Predicted	75.5	65.1	17.0
ARCSTN	Gold Seg	81.9	72.9	21.7
	All Predicted	77.6	71.2	19.7
MST1	Gold Seg	83.4	78.2	21.1
	All Predicted	78.7	77.2	17.2
MST2	Gold Seg	84.5	82.4	25.8
	All Predicted	79.5	82.2	18.0
EASYFIRST	Gold Seg	85.0	80.7	26.5
	All Predicted	80.3	78.5	23.0

Table 8.2: *Training on predicted tagging*

The move to using predicted data in training proved crucial to parsing performance, and results for all parsers jump back up to the 80 to 85 range for the gold-segmentation case. The left-to-right transition-based parsers are the least resilient to the move to predicted data.

Training on data with predicted tags provides a simple and effective solution to the resource incompatibility problem discussed in section 4.2.4. If the best tagger has a different tagset than the one available in the Treebank, all one needs to do is automatically re-tag the Treebank sentences using the tagger. Re-tagging the Treebank data using the

[1]Because the tagger was not trained on the Treebank data, no jacknifing procedure was needed.

tagger which will be used in test time is recommended also if the tagger and Treebank share the same tagset – it facilitates better machine learning by making the training distribution more similar to the testing distribution. From this point on, all of the reported results will be for the case of training the parser on automatically predicted tags.

Training with out-of-the-box morphology features In this set of experiments I use predicted data for training and testing, and evaluate the MST and MALT-ARA parsers on data with morphological features, using the parser's out-of-the-box support for morphology information. MALT-ARA uses the feature set optimized for Arabic, a language similar to Hebrew, and the MST parser uses a generic set of morphological features which include conjunctions of the morphological properties values with other features such as POS-tags of the edge end-points and so on. These results are reported in Table 8.3.

Parser	Test Data	UAS	Root	Exact Match
MALT-ARA	Gold Seg	79.9	67.7	20.5
	All Predicted	76.1	63.8	17.4
MST1	Gold Seg	83.4	78.2	21.1
	All Predicted	78.2	77.6	17.1
MST2	Gold Seg	84.6	82.0	25.7
	All Predicted	79.6	81.3	17.8

Table 8.3: *Parsing results with generic support for morphological features*

Adding the morphological features does not help the parsers – the parsing scores remain mostly the same. Note, however, that a fair amount of morphological information is already captured in the original feature sets: the POS-tags distinguish construct and non-construct nouns, adjectives and determiners, distinguish regular and infinitive verbs, and assign a specific tag to the את case marker.

While the morphological information did not improve the accuracy of the out-of-the-box parsers, it may prove beneficial for the EASYFIRST parser. One reason for the ineffectiveness of the morphological features for the globally-optimized MST parsers is that the parsers are good enough at coming up with good structures to begin with, and the resulting parses do not have many agreement mistakes to begin with. However, in the EASYFIRST parser, the order of attachment is also important – an effective order for attaching edges can result in better parses overall, and the agreement information may provide good hints as to the desirability of given actions. Thus, there is hope that the use of agreement information may able to improve the accuracy of the EASYFIRST parser.

8.3 Handling morphological agreement in the EASYFIRST parser

Having established that the EASYFIRST parser is effective for parsing Hebrew, I extend the feature-set of the EASYFIRST parser to incorporate agreement information.

For the most part, agreement can be described as a relation between two head words, and is easily captured by dependency grammars. There are some cases where the agreement indicators[2] do not originate on the head word itself, and are instead propagated from one of its dependents. These cases will be considered later.

I concentrate on gender and number agreement. As discussed in Chapter 3, Hebrew words can be morphologically marked for Gender (either Masculine (`Masc`), Feminine (`Fem`), Both (`Both`) or None (`NA`)) and Number (either Singular (`Sg`), Plural (`Pl`), Dual, Both (`Both`) or None (`NA`)). The `NA` marking is used for words that do not bear gender or number, such as prepositions or adverbs. In this work, I unify the Dual and Plural categories into a single `Pl` (plural) category.

Gender agreement means that the gender-features should be compatible. Specifically, `NA` does not agree with anything, `Both` agrees with either `Fem` or `Masc`, `Fem` agrees with `Fem` and `Masc` agrees with `Masc`.

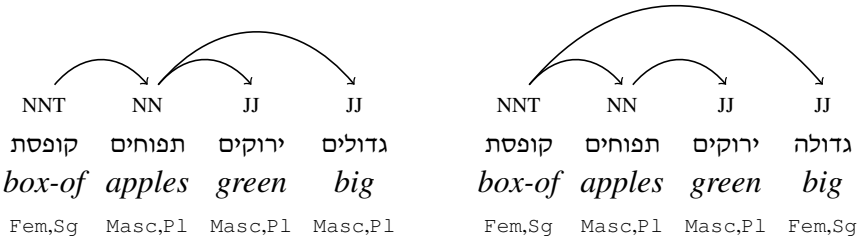
Number agreement means that the number-features should be compatible. Specifically, `NA` does not agree with anything, `Both` agrees with either `Sg` or `Pl`, `Sg` agrees with `Sg` and `Pl` agrees with `Pl`.

8.3.1 Cases where agreement can disambiguate attachments

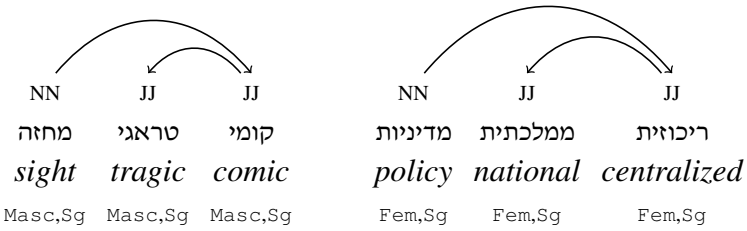
Instead of using the generic agreement feature set, I design a Hebrew-specific feature set based on the knowledge of Hebrew syntax and agreement rules. The feature set handles the following cases where Hebrew syntax requires gender and/or number agreement.

Noun-adjective agreement This is the most common case of agreement in Hebrew. Adjectives and Nouns should agree in both gender and number.

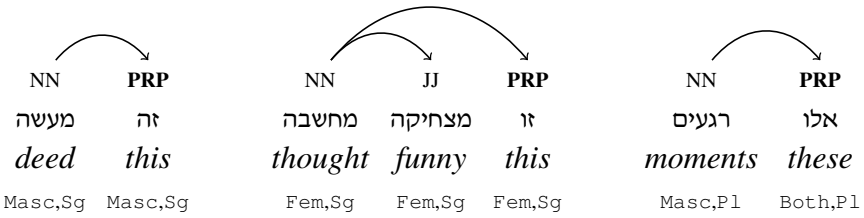
[2] A note regarding terminology is in order: when discussing agreement, the word "feature" is usually used to indicate the gender or number properties of the lexical item. Here, I reserve the word "feature" to be used in the machine-learning sense, and use the word "indicator" when referring to the linguistic sense of the term "feature".



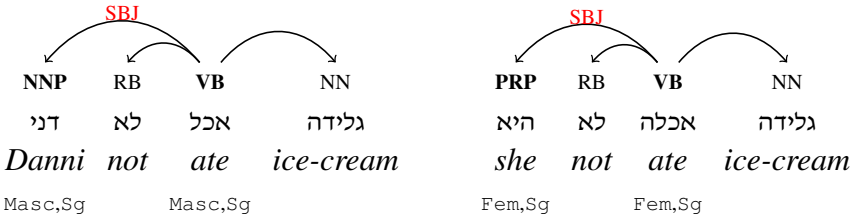
Adjective-adjective agreement In some constructions an adjective modifies another adjective. In these cases, both adjectives should agree in gender and number:



Noun-demonstrative agreement The demonstratives זה, זו, אלו, and כלשהו (this, these, some) may appear in an adjectival position after a noun and should agree with the noun in gender and number.

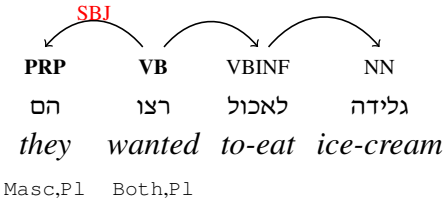


Subject-verb agreement Subjects should agree with the verb in gender and number. The subject word is usually[3] either a noun, or a personal pronoun (הוא, אני, הן, . . .).

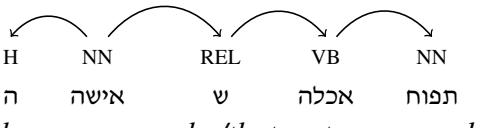


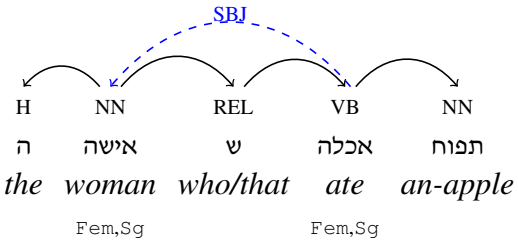
In the case of an infinitive construction, the infinitive verb does not carry either gender or number, and agreement should be between the subject and the finite verb.

[3]In the Hebrew Treebank, 92.5% of the subjects are nouns (including proper nouns and construct nouns) or pronouns. The additional 7.5% of the cases are due to the head word being a coordinating conjunction (3%), a question word (1%) or other (3.5%). Question words carry neither gender nor number. Coordination is more complex: the number of the coordination is usually plural, while the gender is tricky to define.

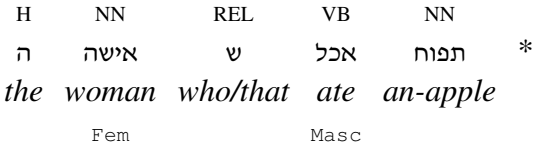


Null-subject Agreement Sometimes, the subject is not realized as a verb argument in the dependency structure. For example, consider the noun-modifying relative clause below:



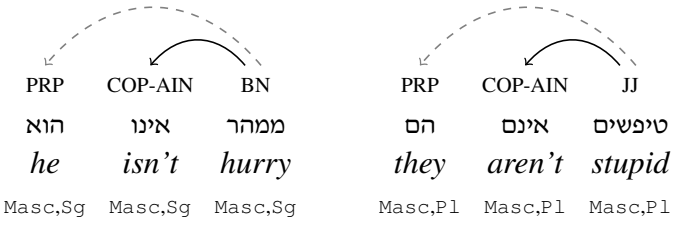


Here, the subject of the embedded verb אכלה (“ate”) is the external noun אישה (“woman”). This relation is indicated by the dashed (blue) line above, but it is not part of the dependency tree as it defies the tree structure (introducing a cycle). Nevertheless, subject-verb agreement still holds between the verb and its subject, rendering the following ungrammatical:

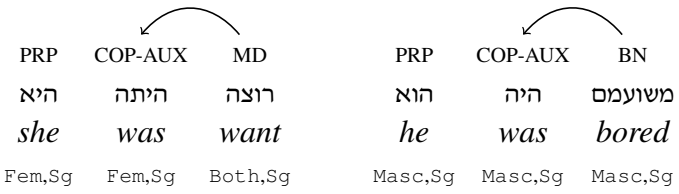


*

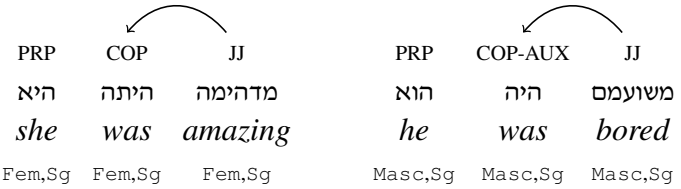
Copular and auxiliary agreement In predicative construction involving the inflected אין copular elements (“he *isn’t* X”), we verify agreement of the inflected אין and the main predicate. Note that agreement is not checked on the dashed edges.



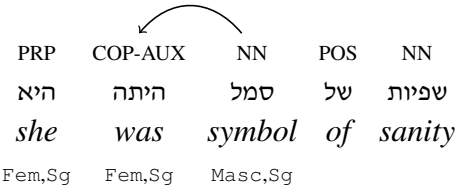
The past forms of copulas are treated as auxiliaries when attached to Beinoni or modal forms, and should agree in gender and number.



Past and Future forms of the copulas agree both in gender and number with adjectives.



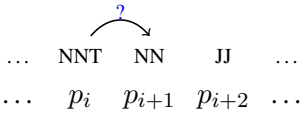
Agreement is *not enforced* between copulas and nouns:



8.3.2 Easy-first morphological agreement features

Having surveyed the grammatical agreement constraints in Hebrew, I integrate them into the EASYFIRST dependency parser as features.

Agreement of neighboring attachments The features should guide the parser toward good easy actions and away from bad or uncertain ones. When considering an action that will attach two structures, p_i and p_{i+1} (either **ATTLEFT$_i$** or **ATTRIGHT$_i$**), agreement features indicating if the agreement indicators of the head words of p_i and p_j are compatible could help the parser decide whether to attach the structures or not. It is also useful to look at the agreement status of neighboring items. For example consider a configuration such as:

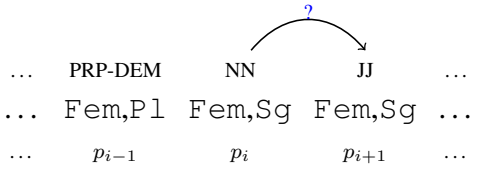


Here, the parser needs to score the attachment of the construct-noun at p_i as the parent of the noun at p_{i+1}. This decision competes with the attachment of the noun at p_{i+1} as the parent of the noun at p_{i+2}. Knowing that there is no morphological agreement between the noun at p_{i+1} and the adjective at p_{i+2} is a useful information in favor of the construct-noun attachment.

When scoring an action between structures p_i and p_{i+1} I include agreement features reflecting the agreement between the pairs $\langle p_i, p_{i+1} \rangle$, $\langle p_{-1}, p_i \rangle$ and $\langle p_{i+1}, p_{i+2} \rangle$.

Form of agreement features All the agreement features are binary valued functions, indicating whether an agreement relation (either number or gender) holds between two head-words. Agreement features make use of the functions $genderAgree(w_i, w_j)$ and

$numberAgree(w_i, w_j)$, which return T if the words are compatible in their number/-gender properties and F otherwise. The value of this function is conjoined with the types of the words being checked for agreement, as well as with their locations relative to the decision being made. For example, features extracted for the configuration



include

"`i/i+1/noun-adjective-numberAgree=T`" and

"`i-1/i/dem-noun-numberAgree=F`".

(Note the +1 and -1 refer to the index of the word on the current list of processed items, and not to the original sentence indices of the words.)

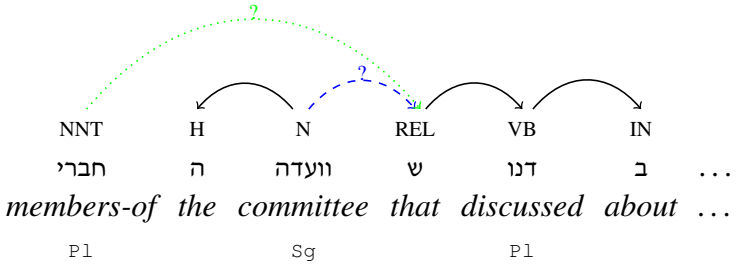
Simple cases The feature extraction for noun-adjective, adjective-adjective, noun-demonstrative and copular agreement cases is straightforward. Subject-verb agreement is more involved.

Subject-verb agreement Hebrew subjects are not morphologically marked. While most subjects are nouns, nouns can also function as objects. Objects are also arguments of verbs, but there is no agreement requirement between verbs and their objects. Thus, an indication as to whether a noun and a verb are or are not in agreement is of little use without knowing if the noun is a subject or an object. A way of distinguishing subjects from objects is needed. One possible indication is word order. Hebrew's word order is flexible, allowing subjects to appear before or after the verb. In the Hebrew Treebank, subjects appear before the verb roughly 70% of the times, and after it 30% of the time. Fortunately, objects appear almost exclusively *after* the verb. A noun appearing before its verbal parent is almost certain to be a subject and not an object. On top of that, definite direct objects are morphologically marked in Hebrew. This means that a non-definite noun which is not marked by the את marker cannot be an object, and therefore it can be treated as a potential subject.

The procedure for subject-verb agreement feature extraction make use of this information, and only fires agreement features for noun-verb pairs in which the noun is likely to be a subject. To do that, we need to know the definite status of nouns. Proper-names (NNP) are always definite, Nouns (NN) are marked for definiteness by the ה marker, and construct-nouns (NNT) inherit the definiteness status of the NP they modify. The EASYFIRST parsing algorithm can be easily extended to track definite

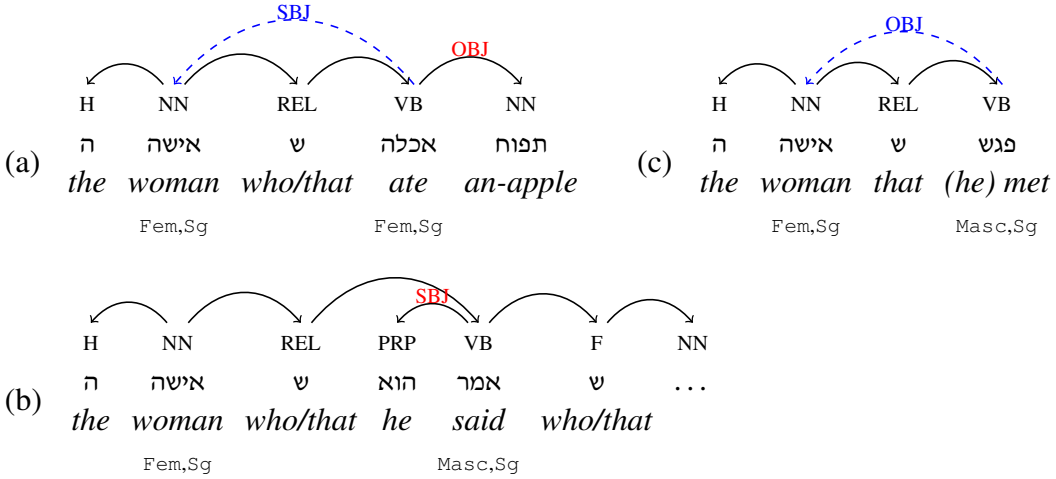
information throughout the parsing process. Initially, proper-names are marked as *definite*, and all other nouns are marked as *maybe-definite*. Whenever the parser connects an ה as the child of a noun, the definiteness of the noun changes to *definite*. Whenever the parser connects a construct-noun as the parent of either NN, NNT or NNP, the construct-noun definiteness changes to the definiteness status of its new child.

Null-subject agreement The null-subject case, in which the subject-verb relation is not indicated in the dependency structure, is still useful for disambiguating some of the parser attachment decisions. Consider the following configuration:



The sentence reads "the members of the committee that discussed ...". The structure in black is already in place, and the parser should decide whether to attach the relative clause "that discussed" as a modifier of the singular "committee" (dashed blue) or the plural "members" (dotted green). The verb "discussed" is in the plural form, indicating that the correct attachment is the green (dotted) one. However, the plural marking is on the verb, and not on the relativizer word ש, which is not morphologically marked.

A naive approach would be to pass the agreement information from a verb child to its ש (relative marker) parent. However, this is not always possible. Consider:



In (a), the subject link is indeed not realized, the subject of the embedded verb is the external NP, and the morphological features of the verb should propagate to the relativizer word ש. However, in (b), the subject of the embedded verb is realized in the embedded clause, and the external noun is not the subject of the verb. In (c), the external noun is

Parser	Test Data	UAS	Root	Exact Match
EASYFIRST	Predicted Tags	85.0	80.7	26.5
EASYFIRST+Morph		85.4	80.5	28.7
EASYFIRST	All Predicted	80.3	78.5	23.0
EASYFIRST+Morph		80.8	79.5	25.1

Table 8.4: *Parsing accuracies for the* EASYFIRST *parser with and without the agreement features*

Test Data	Agreement Type	UAS	Root	Exact Match
Predicted Tags	No agreement	85.0	80.7	26.5
Predicted Tags	noun-adj	85.0	79.7	29.1
Predicted Tags	verb-subj	84.9	81.2	28.2
Predicted Tags	rel-null-subj	85.0	81.2	28.4
Predicted Tags	copular	84.9	79.9	27.3
Predicted Tags	All agreement	85.4	80.5	28.7

Table 8.5: *Relative contribution of the various agreement cases*

taken to be the object, and not the subject, of the embedded verb. In both (b) and (c), we do no expect agreement between the extenral noun and the verb, as the external noun is not the subject.

Morphological indicators should be propagated from the verb to its v parent only for the *relevant* cases. I approximate the relevant cases by verifying that the verb does not have a nominal (noun or pronoun) left child. This approximation fails for the case in (c) above where the subject of the embedded verb is not realized at all.

Agreement features involving v fire only for relevant instances of v, *i.e.,* those instances that have morphological information propagated to them from their children.

8.3.3 Evaluation and results

Table 8.4 presents the parsing results after the incorporation of the agreement features. The features provide useful information for the parser, increasing parsing accuracy on the development set by about 0.4 UAS points, as well as contributing to the exact-match scores.

Next, I turn to investigate the individual contribution of the various agreement types to the final accuracy. Table 8.5 presents the results of experiments in which only specific agreement kinds were used in the features. Interestingly, non of the agreement features bring improvement on their own – yet their combination contribute positively to parsing accuracy.

MST with specific morphological features The generic morphological features did not improve the accuracy of the globally optimized MSTparser. Maybe the generic features were too noisy? I modified the feature extraction of the first-order MST parser

Parser	Test Data	UAS	Root	Exact Match
MST1	Predicted Tags	83.4	78.2	21.1
MST1+HebMorph		83.2	78.7	22.1
MST1	All Predicted	78.7	77.2	17.2
MST1+HebMorph		78.2	77.6	18.2

Table 8.6: *Performance of the* MST*1 parser using Hebrew-specific agreement features*

to include Hebrew-specific agreement features. The edge-factored nature of the MST parser precluded the incorporation of some features: the agreement features were restricted to features that could be verified reliably based on the edge-pairs alone. This allowed the incorporation of the noun-adjective, subject-verb for pronouns and nouns before the verb, and the copular features, but not the definite-subjects after the verb or the relative-clause embedded subject features. In addition, features are restricted to one edge at a time, so the "neighboring agreement" features of EASYFIRST could not be incorporated. Still, the noun-adjective features provide a fair amount of information and I expected them to contribute to parsing performance. The agreement features in the MST parser are conjoined with the binned-distance and direction between the head and the modifier. Results are presented in Table 8.6.

Interestingly, parsing results for the MST parser did not improve as much as the EASYFIRST parser with the incorporation of the specific features: while the Root and Exact Match scores increased slightly, the UAS scores decreased.

8.4 An ensemble-based parser

The different parsing paradigms (left-to-right, easy-first and graph-based) produce markedly different results. Indeed, while UAS scores of the EASYFIRST+Morph and the MST2 parsers on gold-segmentation and predicted tagging are quite close (84.5 and 85.4) they only agree on 86% of the edges. This suggests that even better parses could be achieved using an ensemble method for combining the predictions of the different parsers.

The parse combination method I use is fairly simple: each of the parsers in the ensemble parse the same sentence. Then, each parser "votes" for the edges it produced. If the same edge was suggested by two parsers, it will have two votes. If it was suggested by three parsers it will have three votes. Then I look for a valid parse tree that has the edges with the most votes. This is done by using the dynamic-programming algorithm of the graph-based parser to select the best parse using the votes as the edge-scores.[4]

[4] An alternative would be to use a faster approximate algorithm which works very well in practice, as suggested in [10].

8.4.1 Evaluation and results

I experimented with parse ensembles based on different combinations of parsers. The development set results are summarized in Table 8.7.

ARCSTN	MST1	MST2	EASYFIRST+Morph	COMBINATION
77.5	78.7	79.6	80.8	-
•	•	•		80.3
•	•		•	80.9
	•	•	•	80.4
•		•	•	**81.2**
•	•	•	•	**81.2**

Table 8.7: *Parser-ensemble UAS scores on the development set for various parser ensembles. The parsers were trained on gold segmentation and predicted tagging, and tested on predicted segmentation and tagging.*

Most parse ensembles improve the performance over a single parser. However, diversity matters. The first-order and second-order MST parsers provide highly redundant information: an ensemble based only on them and the EASYFIRST+Morph parser drags the results down (from 80.8 to 80.4 UAS), while adding the MST1 parser to an ensemble based on the ARCSTN , MST2 and EASYFIRST+Morph does not provide any benefit. The predictions of the EASYFIRST+Morph parser are markedly different than those of the other parsers, and the ensembles in which it participates provide higher gains in parsing accuracy. The overall quality of the ensemble parts also matters: ensembles including both the MST2 and EASYFIRST+Morph parsers provide the highest parsing accuracies (81.2% UAS).

The ensemble method is effective, especially when used with a diverse set of parsers. The EASYFIRST+Morph parsing algorithm results in different parses than those produced by the graph-based and transition-based parsers. If one is willing to invest computing-time for better parsing accuracy, a parser ensemble based on the ARCSTN , MST2 and EASYFIRST+Morph parsers is an appealing option.

8.5 Adding an edge labeler

While the EASYFIRST parser produced unlabeled parse trees, some of the edges in the Hebrew Dependency Treebank are annotated with edge-labels. These labels include SBJ (on edges between a subject and its parent), OBJ (on edges between an object and its parent), COM (indicating verb complements), and CONJ (on edges between the coordinated elements and their coordinating-word parent in coordination constructions). This section presents an edge-labeler – a simple and effective machine-learning component for recovering the edge labels.

Unigram Features
c_t, p_t, c_f, p_f
c_t^{-1}, $c_t^{-1}c_t$, c_f^{-1}, p_t^{-1}, $p_t^{-1}p_t$, p_f^{-1}
c_t^{-2}, $c_t^{-2}c_t$, c_f^{-2}, p_t^{-2}, $p_t^{-2}p_t$, p_f^{-2}
c_t^{+1}, $c_t^{+1}c_t$, c_f^{+1}, p_t^{+1}, $p_t^{+1}p_t$, p_f^{+1}
c_t^{+2}, $c_t^{+2}c_t$, c_f^{+2}, p_t^{+2}, $p_t^{+2}p_t$, p_f^{+2}
Bigram Features
c_tp_t, c_fp_f, p_fc_t, p_tc_f
Trigram Features
$c_t^{-1}c_tc_t^{+1}$, $p_t^{-1}p_tp_t^{+1}$
Sibling Features
$\forall s \in \text{sib}(c)$: s_t, s_tc_t, s_tp_t
if $\text{sib}(c) = \emptyset$: $\texttt{nosib}p_t$, $\texttt{nosib}c_t$
Grandchildren Features
$\forall g \in \text{children}(c)$: g_t, g_tc_t, g_tp_t, g_f
if $\text{children}(c) = \emptyset$: $\texttt{nogrn}p_t$, $\texttt{nogrn}c_t$
Between Features
$\forall m$ between c and p: m_t, c_tm_t, m_tp_t, $c_tm_tp_t$

Figure 8.1: *Edge-labeler features. p: parent node; c: child node; c^{+1}: the node following c in the linear order of the sentence; sib(c): all the sibling nodes of c; children(c): all the children nodes of c; x_t: the POS-tag of x; x_f: the word-form of x. All definite nouns (including construct nouns) are marked for definiteness on the POS-tag. The POS-tag of CC is conjoined with the POS-tag of the left-most child of the CC. All features appear also when conjoined with the direction between child and parent.*

The proposed edge-labeler assigns an edge-label for each of the edges in a parse-tree. This is achieved by treating each edge as an independent instance of a multiclass classification problem. The classifier can condition on any aspect of the unlabeled dependency tree. In practice the features are all derived from the endpoints and the tree structure surrounding the edge under considerations. The complete feature-set is detailed in Figure 8.1. Assuming each edge classification can be done in constant time, labeling the entire parse tree is linear in the length of the sentence.

8.5.1 Evaluation and results

I used the multiclass averaged-perceptron as the multiclass classifier. The classifier is trained for 10 iterations over the training data. Table 8.8 presents the precision and recall of each edge label when trained and tested on the gold parse-trees from the training and development sets, and Table 8.9 presents the confusion matrix for the edge-labeler.

Label	**Precision**	**Recall**	**F_1**
SBJ	89.9	94.4	92.1
OBJ	91.6	84.4	87.8
COM	76.3	46.1	57.5
CONJ	96.1	96.9	96.5

Table 8.8: *Labeler effectiveness for the various edge labels, when trained and tested on gold parse-trees*

Predicted → **↓ Correct**	**SBJ**	**OBJ**	**CONJ**	**COM**	**dep**
SBJ	608	8	3	0	25
OBJ	20	250	0	1	25
CONJ	1	0	588	0	18
COM	0	2	0	206	239
dep	47	13	21	63	8905

Table 8.9: *Confusion matrix for the edge-labeler*

The labeler works well for recovering the SBJ and CONJ labels, performs reasonably well for the OBJ label, and does not succeed in recovering the COM label. The COM label is confused with the generic "dep" label, indicating that it is hard for the labeler to distinguish arguments from adjuncts. This is expected, as the argument-adjunct distinction is hard also for human annotators in many cases, and the Treebank data is not sufficient for learning an accurate classifier for this task – particularly for verbs which are encountered only a few times in the data. Argument-adjunct distinction is better

Parser	Test Data	UAS	LAS	Exact Match	Labeled Exact Match
EASYFIRST+Morph+Labeler	Gold Seg	85.4	84.0	28.7	25.9
	All Predicted	80.8	79.5	25.1	23.0

Table 8.10: *Labeled parsing accuracies (considering the subject, object and coordination labels) on the development set, using a parser + labeler pipeline*

performed based on data external to the Treebank, such as a resource describing the subcategorization frames (required arguments) for the various Hebrew verbs. Such a subcategorization dictionary is not currently available for Hebrew. A research project at the Technion, headed by Alon Itai and Shuly Wintner, is currently underway and is expected to produce such a dictionary. Recent work [108, 119] describes ways of automatically acquiring subcategorization frames for English based on automatically parsed corpora. Once such a dictionary is available, I expect it could be used to substantially improve the argument/adjunct classification for Hebrew.

How well does the labeler's success translate to predicted trees? To answer this question, I measure the labeled attachment score (LAS) and labeled exact-match of the EASYFIRST parser (including the morphological agreement features) + labeler on the development set. In this setting the labeler is trained on the gold trees, but is applied to automatically parsed trees. I focus on the subject, object, and conjunction labels, ignoring the COM relation, which the labeler cannot produce reliably even when the gold trees are available. Table 8.10 presents the results.

8.6 Evaluating the final system

In this section I evaluate the final parsing system, which uses the Edge-labeler on top of either the EASYFIRST+Morph parser or a parse-ensemble, to produce labeled dependency parsing results. The numbers are based on the test section of the Treebank. None of the previous experiments were run against this test section, and no parameter selection or optimization was performed. Thus, these numbers should hopefully provide a good estimate regarding the generalization capacity of the parsing system. Tables 8.11 and 8.12 present the final accuracy results.

Parser	Test Data	UAS	LAS	Exact Match	Labeled Exact Match
EASYFIRST+Morph+Labeler	Gold Seg	83.7	82.5	22.2	20.0
	All Predicted	78.6	77.5	14.6	14.1

Table 8.11: *Parsing results for the final system on the test-set, using the* EASYFIRST+*Morph parser*

The parsing results on the test section are lower by about 1-2 points than the results on the development set. This is consistent with previous work [142] which also observed

Parser	Test Data	UAS	LAS	Exact Match	Labeled Exact Match
Ensemble+Labeler	Gold Seg	85.0	83.8	22.8	20.3
	All Predicted	79.6	78.5	15.2	14.3

Table 8.12: *Parsing results for the final system on the test-set, using an ensemble parser based on the* EASYFIRST+*Morph,* ARCSTN *and* MST2 *parsers*

the test-set to be "harder" than the development-set.

8.7 Conclusions

In this chapter, I experimented with Hebrew dependency parsing based on both out-of-the-box parsers and the EASYFIRST parser introduced in the previous chapter. All the parsers are based on a pipeline system in which a POS-tagger performs tagging (including assignment of morphological features) and word-segmentation prior to parsing. One important finding is that training the parser based on the output of the tagger (and not on gold-tagged data) is necessary for obtaining good parsing performance. The EASYFIRST parser is well suited to parsing the Hebrew data: it achieves parsing accuracies which are better than the globally optimized second-order MST2 parser, which is a state-of-the-art parser for English.

I presented an extended feature set for the EASYFIRST parser which is designed to make use of specific grammatical agreement patterns required by the Modern Hebrew grammar. While the parser makes few agreement mistakes to begin with, the extended feature-set did contribute positively to parsing performance, increasing the UAS score by about 0.4 UAS scores in both the gold segmentation scenario and the full parsing scenario. Using the extended feature set, the parser outperforms the globally optimized second-order MST2 parser (while the EASYFIRST parser being much faster). An attempt to use similar agreement features in the globally optimized MST parser did not show any improvements in parsing accuracy.

The EASYFIRST parser predictions differ from those of the graph-based and left-to-right parsers. Using an ensemble based on the three kinds of parsers improves the results even further, resulting in development-set accuracy of 81.2 UAS on the full parsing task.

I introduced a machine-learning-based edge-labeling algorithm that assigns edge labels to unlabeled parsed trees. The edge labeler works well for identifying the SBJ, OBJ and CONJ, but not the COM relation. Complements and adjuncts are hard to distinguish without external lexical knowledge. With this labeler, the labeled parsing scores (LAS) are only about 1-point lower than the unlabeled (UAS) scores. .

When evaluated on the final test-set, the EASYFIRST with morphology features

produced UAS scores of 83.7 when starting with gold segmentation, and 78.6 on automatically tagged data. The ensemble system produced UAS scores of 85.0 UAS when starting with gold segmentation, and 79.6 UAS on automatically tagged data. These are the best published results for Hebrew dependency parsing.

Part III

Constituency Parsing

Chapter 9

Background on Constituency Parsing

In this chapter I provide some background on the methods and evaluation measures for constituency parsing. The reader is referred to [87, Chap. 11,12],[76, 113] for further details and discussion.

9.1 Evaluation measures

Each tree is treated as a set of labeled constituents[1]. Each constituent is represented as a 3-tuple $\langle i, j, L \rangle$, in which i and j are the indices of the first and the last words in the constituent, respectively, and L is the constituency label. For example, $(2, 4, \text{NP})$ indicates an NP spanning from word 2 to word 4.

The performance of a parser is evaluated based on the amount of constituents it recovered correctly.

Let G denote the set of constituents in a gold-standard constituency tree, and P denote the set of constituents in a predicted tree. We now define *precision* (P) and *recall* (R), both of which range from 0 to 1.

$$\text{precision} = \frac{|G \cap P|}{|P|} \qquad\qquad \text{recall} = \frac{|G \cap P|}{|G|}$$

Precision is the fraction of correctly predicted constituents out of all the predicted constituents, and Recall is the fraction of correctly predicted constituents from among the real constituents. The F_1 measure is used to combine precision and recall into a single measure:

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

[1] This assumes unary-chains do not contain cycles.

F_1 also ranges from 0 to 1, and it is 1 iff both precision and recall are 1, indicating the trees are identical.

When evaluating over a test-set composed of many trees, one can either compute the P,R and F_1 for each tree individually and report the average, or clump all the constituents in the gold and predicted trees into two large sets (each constituent is now assigned a unique tree-id as well) and calculate the measures based on these sets. The former method tends to score a bit higher, while the latter approach, which is more standard, is the one I use in this work.

Another evaluation measure is *exact match*: the fraction of trees in the test-set that are assigned an F_1 score of 1.

In this work the reported numbers are in percentages rather than fractions (ranging from 0 to 100), and they are calculated to a precision of two decimal points.

When measuring the performance of models in which the token-segmentation is predicted and can contradict the gold-standard, a generalization of the above measures is used. Instead of representing a constituent by a triplet $\langle i, j, L \rangle$, each constituent is represented by a pair containing the concatenation of the words at its yield, and its label L. This measure was suggested by [140] and used in subsequent works [56–58, 141]. This is equivalent to reassigning the i and j indices above to represent character positions instead of word numbers. When the yields of the gold and the predicted trees are the same, this is equivalent to the standard evaluation measure using the $\langle i, j, L \rangle$ triplets of word indices and a label, and it will produce the same precision, recall and F_1 as above.

9.2 Probabilistic context-free grammars

Context-free grammars A constituency tree can be thought of as being generated from a context-free grammar (CFG), a set of rewrite rules of the form $LHS \rightarrow RHS$, where LHS is a *non-terminal* symbol and RHS is either a sequence of non-terminal symbols or a *terminal-symbol* . A tree can be generated by the grammar by starting with an initial non-terminal symbol, and recursively rewriting the non-terminal symbols according to the grammar rules (a non-terminal symbol is being rewritten by choosing a rule with an LHS matching the non-terminal symbol, and replacing the non-terminal symbol with the content of the RHS of the rule) until a string consisting of only terminal symbols is produced. Consider the toy grammar in Figure 9.1. Items starting with uppercase letters are non-terminal symbols, and items starting with lowercase letters are terminal symbols.

This grammar can generate the sentence “fruit flies like a banana” by starting with

(1)	S	→ NP VP	(8)	Adj	→ fruit
(2)	NP	→ Adj Noun	(9)	Noun	→ flies
(3)	NP	→ Det Noun	(10)	Vb	→ like
(4)	NP	→ Noun	(11)	Det	→ a
(5)	VP	→ Vb NP	(12)	Noun	→ banana
(6)	VP	→ Vb PP	(13)	Noun	→ tomato
(7)	PP	→ Prep NP	(14)	Adj	→ angry
			(15)	Vb	→ flies
			(16)	Prep	→ on
			(17)	Prep	→ like
			(18)	Noun	→ fruit

Figure 9.1: *A Toy Context Free Grammar (CFG)*

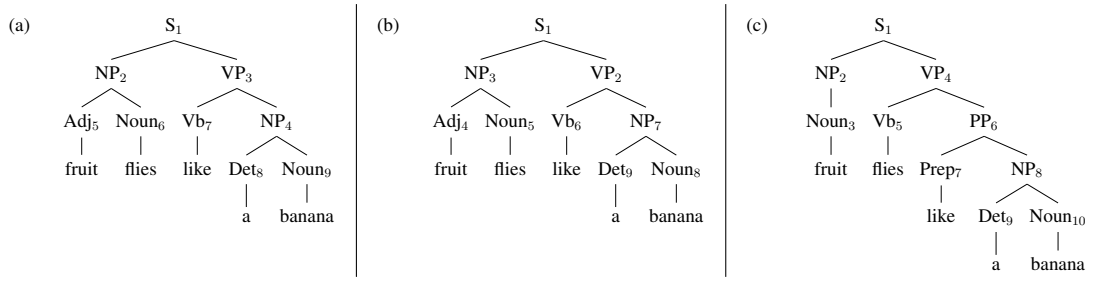


Figure 9.2: *Three derivations of the sentence "fruit flies like a banana" using the grammar in Figure 9.1. The numbers represent the order of rule application. Derivations (a) and (b) are equivalent and differ only in the order of derivation, while the derivation in (c) is different and uses a set of rules.*

the non terminal S, and rewriting it using rule (1) to "NP VP". Then, using rule (2), the first NP is written as "Adj Noun" producing "Adj Noun VP". Applying rule (5) on the last VP will produce "Adj Noun Vb NP". Applying rule (3) on the last NP will produce "Adj Noun Vb Det Noun" and then applying rules (8), (9), (10), (11) and (12) at the Adj, Noun, Vb, Det and Noun respectively will produce the desired sentence "fruit flies like a banana". By tracing the steps of the derivation, we come up with the tree in Figure 9.2a. The numbers represent the order in which the steps were taken, though the order of the derivation does not matter. Various different orders could have resulted in the exact same tree, for example the derivation in 9.2b.

The grammar could have produced a different derivation (using a different set of rules, not just a different order) for the same sentence, as depicted in Figure 9.2c.

Probabilistic context-free grammars A probabilistic context free grammar (PCFG) is an extension of a CFG in which each rule is a score such that all scores are non-negative, and the sum of scores of rules with the same LHS sum to 1. Using a PCFG one can assign probabilities to CFG derivations: multiplying the scores of the rules in the derivation gives the probability of generating the tree and the sentence under the given set of PCFG rules. The multiplication implies that the rule applications are

independent – this is the "context-freeness" in context-free grammars.

PCFGs can be used to disambiguate the correct structure of a sentence when several different trees share the same yield. Disambiguation is achieved by choosing the highest probability tree.

9.3 Grammar binarization

The CKY algorithm discussed below requires a grammar with only binary or unary rules (having either one or two symbols on each RHS). A grammar in this form is said to be in *Chomsky normal-form (CNF)*. This requirement on the grammar form is not a restriction, as n-ary rules can be easily binarized. For example, a 4-ary rule of the form A $\rightarrow$ B C D E can be replaced with the following set of rules[2]:
A $\rightarrow$ B A/B-CDE
A/B-CDE $\rightarrow$ C A/BC-DE
A/BC-DE $\rightarrow$ D E

Notice how each of the new rules remembers both the original parent (A) as well as the complete horizontal context. The grammar produced by such binarization is equivalent to the grammar prior to binarization in that they can both produce the same set of trees[3], and probabilities can be assigned to the binarized grammars so that resulting trees will have the same probabilities under both grammars (*e.g.*, by setting the probability of the first binarization step to the probability of the original rule, and by setting the probabilities of the rules in subsequent binarization steps to 1).

However, sometimes it is useful to purposefully forget some of the context when binarizing. For example, rule of the form S $\rightarrow$ NP VP PP can be binarized into:
S $\rightarrow$ NP S/VP
S/VP $\rightarrow$ VP PP

Notice how the second rule S/VP $\rightarrow$ VP PP does not encode the PP in the left-hand side. Binarizing many rules in this forgetful scheme results in a grammar which can produce rules not seen in the original grammars. This may have a good smoothing effect in which the grammar learns to generalize well to new useful rules. Alternatively, it can have a disastrous effect, in which the grammar is too permissive and allows the creation of many bad rules. Such forgetfulness binarization is referred to in the literature as "rule markovization". See [76] for further discussion.

[2]The example follows a right-most binarization. Other binarization options are possible, see [116, 134] for an overview.

[3]This kind of equivalence between grammars is known as *strong equivalence* – the grammars can produce the same set of trees. This is in contrast to *weak equivalence*, which holds if both grammars can produce the same set of terminal strings, but does not require the derivation trees to be equivalent.

9.4 The CKY algorithm

The CKY algorithm [30,75,153] is a popular dynamic programming algorithm that can be used to reconstruct the most probable derivation tree over a given input string and a given grammar. The "classic" CKY algorithm is used for *recognition* , answering the question "can this sentence be generated by this grammar". The algorithm is then extended to solve the problem of finding the most probable tree for the given sentence and grammar, if such a tree exists. I will present this extended version.

The discussion of the CKY algorithm assumes that the grammar is in binary form. We distinguish three kinds of grammar rules: *lexical rules* are rules of the form A $\rightarrow$ b where A is a nonterminal symbol and b is a terminal symbol (a word in the vocabulary), *unary rules* are rules of the form A $\rightarrow$ B where both A and B are nonterminal symbols, and *binary rules* are of the form A $\rightarrow$ B C where A, B and C are all nonterminal symbols.

The CKY algorithm searches for the tree in a bottom-up fashion. Conceptually, the algorithm works with items of the form $\langle i, j, L\rangle$, where i and j are start and end word indices of a constituent, and L is a non-terminal label. Given a sentence $s = w_1, \ldots, w_n$ of n words, and a grammar G, an item of the form $\langle 1, 4, \text{NP}\rangle$ means that the grammar is capable of generating the word sequence $w_1, \ldots, w_4$ when starting from the NP symbol. The goal of the algorithm is to prove that an item of the form $\langle 1, n, \text{S}\rangle$ is possible, meaning that the entire sentence can be generated by the grammar starting from the symbol S.

I refer to the quantity $(j - i + 1)$ as the *width* of an item $\langle i, j, L\rangle$. The algorithm starts with a set of items, and expands it, building wider items based on narrower ones. The main intuition behind the algorithm is that if you have items of the form $\langle i, j, \text{B}\rangle$ and $\langle j + 1, l, \text{C}\rangle$, and the grammar has a rule of the form A $\rightarrow$ B C, then you can derive an item of the form $\langle i, l, \text{A}\rangle$. In other words, an item of the form $\langle i, l, \text{A}\rangle$ can be derived if there exists k such that $\langle i, i + k, \text{B}\rangle$ and $\langle i + k + 1, l, \text{C}\rangle$ are already derived, and A $\rightarrow$ B C is a rule in the grammar. The algorithm is initialized by going over all the sentence words, looking for lexical rules of the form A $\rightarrow w_i$, and creating items $\langle i, i, A\rangle$. Then, the algorithm attempts to create all width-2 items (of the form $\langle i, i + 1, A\rangle$) using the existing width-1 items, after that creating width-3 items using the already created width-2 and width-1 items, width-4 items using the already created narrower items, and so on.

In order to extend the algorithm to return the most probable tree, each item should store two values: (1) the current highest probability of deriving this item, and (b) the narrower items that lead to this most probable derivation. If these values are available,

the most probable tree can be found by starting from $\langle 1, n, \mathrm{S}\rangle$ and tracing back to the two items $\langle 1, k, A\rangle$ and $\langle k+1, n, B\rangle$ that resulted in the most probable derivation of S. This process is then repeated recursively down to the leaves of the tree. These values are easy to track during the creation of the items. Let p_{ijA} be the probability associated with item $\langle i, j, A\rangle$, and $p_{A\rightarrow\ BC}$ the probability associated with grammar rule A $\rightarrow$ B C. The probabilities of the initial width-1 items ($\langle i, i, A\rangle$) are initialized with the grammar probability of the rule A $\rightarrow w_i$, and no backpointers ($p_{iiA} = p_{A\rightarrow\ w_i}$). Then, the probability p' of creating $\langle i, l, A\rangle$ from $\langle i, k, B\rangle$, $\langle k+1, l, C\rangle$ and grammar rule A $\rightarrow$ B C is $p' = p_{ikB} \times\ p_{k+1lC} \times\ p_{A\rightarrow\ BC}$. This probability is compared to p_{ilA}, the current best probability of deriving $\langle i, l, A\rangle$. If $p' > p_{ilA}$ (the new probability is higher), it is stored in $\langle i, l, A\rangle$ together with the symbols B,C and the split-point k.

The non-terminal labels are usually represented as integers, and items are stored in a three-dimensional array. The algorithm works in $O(n^3G^3)$ time and $O(n^2 + G^3)$ memory, where G is the number of non-terminals in the grammar.

9.5 Treebank grammars and grammar refinement

One method of obtaining PCFG grammars is to read them off of a Treebank. This method is appealing because it is easier to annotate sentences for their structures than it is to come up with the set of rules which govern these structures.

A PCFG is made of two parts: (a) the set of rules and (b) the probability of each rewrite. The set of rules can be approximated from a Treebank by extracting all the production rules in the derivations in the Treebank. Obtaining the probabilities can be performed in various ways. One of the simplest ways is to use relative-frequency estimates over Treebank counts: the probability of a rule $LHS \rightarrow RHS$ equals to the number of times this rule was observed in the Treebank, divided by the number of times its LHS was observed. Using grammars obtained this way in order to disambiguate the structure of natural language sentences works poorly. The grammars are not informative enough and the context-free assumptions are too strong. However, it was noted by [73,76] and others that better grammars can be derived by adding additional information to the Treebank trees before reading the grammar from them. For example, each non-terminal node could be annotated by its parent-node, thus capturing a limited amount of context. Other examples of tree annotation include annotating base-NPs (NPs that do not contain any other NPs inside them) differently than recursive NPs, distinguishing between finite and non-finite VPs, distinguishing locative and temporal PPs, and so on. Using a carefully crafted set of tree annotations, [76] managed to increase Treebank parsing accuracy from 77% F_1 to 87% F_1.

9.6 Automatic state-split grammars (PCFG-LA)

Klein and Manning [76] demonstrated that linguistically informed splitting of non-terminal symbols in Treebank-derived grammars can result in accurate grammars. Their work triggered investigations in automatic grammar refinement and state-splitting [90, 117], which was then perfected by [111, 113].

State-split models assume that each non-terminal label has a latent annotation which should be recovered. Instead of a single NP symbol, these models hypothesize that there are many different NP symbols, $NP_1, \ldots, NP_k$, and each is used in a different context. However, the labels are hidden, and we can only observe the core category label (NP). The job of the training process is to come up with the hidden set of label assignments to non-terminals, such that the resulting grammar assigns a high-probability to the observed Treebank data. Such models are called PCFG with latent annotations (PCFG-LA) and are shown empirically to produce very accurate parsing results.

The model of [113] and its publicly available implementation, the Berkeley parser[4], learns the latent annotations by starting with a bare-bones Treebank derived grammar and automatically refining it in split-merge-smooth cycles, and setting the parameters using EM. I provide a brief description of the model and learning process. (Refer to [111, 113, 114] for the full details.)

The learning works by following an iterative split-merge-smooth cycle, in which the following steps are performed repetitively:

Splitting each non-terminal category in two . All of the grammar symbols are split. In the first round, NP is split into NP_1 and NP_2. In the second round these are split into NP_{11}, NP_{12}, NP_{21}, NP_{22}, etc. Each splitting round results in new grammar in which a rule of the form $A \rightarrow BC$ is replaced by eight rules, the result of splitting each A, B, and C in two. An EM procedure is then used to set the probabilities of each of the split rules. The EM training is constrained by the grammar on the one hand and by the annotated tree structures on the other.

Merging back non-effective splits . Not all of the splits are useful. For example, the punctuation POS-tag will always result in punctuation, and there is no reason to split it into two punctuation POS-tags. Having a grammar with too many states is difficult to manage in terms of memory, storage and parsing time, and is also prone to overfitting the data. Thus, the model aims to undo splits if they are not useful. The splits are evaluated based on an information gain criteria, and splits which are not useful are merged back into their parent symbol, resulting in

[4] `http://code.google.com/p/berkeleyparser/`

a smaller grammar (if the symbols B_1 and B_2 are merged back into B, the rules $A \rightarrow B_1$ C and $A \rightarrow B_2$ C are merged into $A \rightarrow$ B C). The merging step is also followed by an EM procedure for setting the rule probabilities for the resulting grammar.

Smoothing the split non-terminals toward their shared ancestor . Finally, split symbols may still share some information (although an NP in subject position and an NP in object position behave differently, they also retain some common properties). The smoothing procedure joggles the probability mass of the grammar and moves some probability from the split symbol to its parent. This step is also followed by parameter re-estimation using EM.

Performing five or six such split-merge-smooth cycles results in accurate grammars, with annotations that capture many latent syntactic interactions. Six cycles mean that symbols can have as many as 64 different substates.

At inference time, the latent annotations are (approximately) marginalized out, resulting in the (approximate) most probable unannotated tree according to the refined grammar (the score of the unsplit rule $A \rightarrow B\ C$ is taken to be $\sum_x \sum_y \sum_z A_x \rightarrow B_y\ C_z$).

The grammar learning process is applied to binarized parse trees, with 1st-order vertical and 0th-order horizontal markovization. This means that in the initial grammar, each of the non-terminal symbols is effectively conditioned on its parent alone, and is independent of its sisters. For example, the rule S $\rightarrow$ NP VP NP PP is binarized as:
S $\rightarrow$ NP @S
@S $\rightarrow$ VP @S
@S $\rightarrow$ NP @S
@S $\rightarrow$ PP
This tells us that S rules start with an NP, can be followed by a sequence of zero or more NPs and VPs, and end with a PP. Such an extreme markovization suggests a very strong independence assumption, and is too permissive on its own. However, it allows the resulting refined grammar to encode its own set of dependencies between a node and its sisters, as well as ordering preferences in long, flat rules. For example, the binarized grammar allows the production S $\rightarrow$ NP NP PP, which may be incorrect. However, by annotating the symbols as follows:
S $\rightarrow$ NP @S_1
@S_1 $\rightarrow$ VP @S_2
@S_2 $\rightarrow$ NP @S_2
@S_2 $\rightarrow$ PP
the grammar now forces the VP to be produced before the NP, but still allows the NP to

be dropped. Similarly, by annotating the symbols as:
$S \rightarrow NP\ @S_1$
$@S_1 \rightarrow VP\ @S_2$
$@S_2 \rightarrow NP\ @S_3$
$@S_3 \rightarrow PP$
the grammar effectively allows only the original rule to be produced.

Initial experiments on Hebrew confirm that moving to higher order horizontal markovization (encoding more context in the initial binarized rules) degrades parsing performance, while producing much larger grammars.

The PCFG-LA parsing methodology is very robust, producing state-of-the-art accuracies for English, as well as many other languages including German [115], French [22] and Chinese [67] among others.

9.7 Related work in constituency parsing of morphologically rich languages

There is a large body of evidence indicating that parsing models developed for English are not easily adapted to other languages [143]. The Head-Driven models of the type that [32] proposed have been ported to parsing many languages, often via the implementation of [17]. For Czech, for instance, the first adaptation by [35] culminated at 80pt F_1-score. For Italian, [36] used the Stanford parser and Bikel's parser emulation of Collins' model 2 [32] and obtained substantially lower results compared to English. It is notable that these models were applied without adding morphological signatures, using gold lemmas instead. [36] further tried different refinements of the annotation scheme, including parent annotation and horizontal markovization, but none of them obtained the desired improvement. For German (a language with rich morphology and moderately free word-order) [41] showed that, to parse German, adding case and morphology information together with smoothed markovization and an adequate unknown-word model is more important than lexicalization [42]. Nevertheless, the results are substantially lower than those for parsing English.

For French, [37] and [126] show that, given a corpus comparable in size and properties (*i.e.,* the number of tokens and grammar size), the performance level, both for Charniak's parser [26] and the PCFG-LA Berkeley parser [113] was higher for parsing the English Penn Treebank than it was for French. Yet, the PCFG-LA parser consistently outperform various lexicalized and unlexicalized models for French [125] as well as German [115]. In this respect, the PCFG-LA can be considered MRLs-friendly, due

to its language agnostic design.

Several works show that the handling of unknown words is a major component to be considered when adapting a parser to a new language. For example, the work in [11] use language-specific unknown-word signatures for several languages based on various indicative prefixes and suffixes, while [67] suggest a Chinese-specific model based on the geometric average of the emission probabilities of the individual characters in the rare or unknown word.

Another method of coping with lexical sparsity is word clustering. In [21], the authors demonstrate that replacing words by a combination of a morphological signature and a word-cluster (based on the linear context of a word in a large unannotated corpus) improves parsing performance for French. The technique provides more reliable estimates for in-vocabulary words (a given cluster appears more frequently than the actual word form), and it also increases the known vocabulary: unknown words may share a cluster with known words.

9.7.1 Arabic

Arabic is similar to Hebrew in the challenges it presents for automatic parsing. Most early work on constituency parsing of Arabic focused on straightforward adaptations of Bikel's parser to Arabic, with little empirical success. [11] show that parsing accuracies of around 81% F_1 can be achieved for Arabic (assuming gold word-segmentation) by using a PCFG-LA parser with Arabic-specific unknown-word signatures. Recently, [59] report on an extensive set of experiments with several kinds of tree annotations and refinements, and report parsing accuracies of 79% F_1 using the Stanford-parser and 82% F_1 using the PCFG-LA Berkeley-parser, both when assuming gold word segmentation. The work of [59] also explored the use of lattice-parsing as suggested in Chapter 10 of this thesis, as well as earlier in [31,57], and report promising results for joint segmentation and parsing of Arabic (an F_1 score of 76% for sentences of up to 70 words). However, the best reported results for parsing Arabic when the gold word-segmentation is not known are obtained using a pipeline model in which a tagger and word-segmenter is applied prior to a manually state-split constituency parser, resulting in an F-score of 79% F_1 (for sentences of up to 70 words) [59].

9.7.2 Hebrew and relational-realizational parsing

Some related work deal directly with constituency parsing of Modern Hebrew. The work of [144] experiment with grammar refinement for Hebrew, and show that an-

notating definiteness and accusativity of constituents, together with parent annotation, improves parsing accuracy when gold word segmentation is available.

The *Relational Realizational* (RR) line of work presented in [142, 145–147] handles the constituent-order variation in Hebrew by presenting a separation between the *form* and *function* aspects of the grammar. Briefly, while plain Treebank-derived grammars have rules such as S $\rightarrow$ NP VP PP NP PP which are applied in a single step, the RR approach suggests a generative model in which the generation of flat clausal structures is decomposed into three distinct steps. First, in the *projection* step, a non-terminal generates the kinds of its children without specifying their form or the order between them, using rules of the form S $\rightarrow$ {OBJ,SBJ,PRED,COM,Adjunct}@S. Second, in the *configuration* step, an order is chosen based on a separate ordering distribution, using rules of the form
{OBJ,SBJ,PRED,COM,Adjunct}@S $\rightarrow$ SBJ@S PRED@S Adj@S OBJ@S COM@S, and third, in the *realization* step, each functional element receives a specific form, using rules of the form SBJ@S $\rightarrow$ NP or Adj@S $\rightarrow$ PP. The realization rules can encode syntactic properties which are required by the grammar for the given function – for example, a rule such as OBJ@S $\rightarrow$ NP$_{def,acc}$ captures the requirement that definite objects in Hebrew must be marked for accusativity using the את marker, and the rest of the generative process will generate the object NP according to this specified constraint. This kind of linguistically motivated separation of form and function is shown to produce models with fewer parameters and result in better parsing accuracies than plain (or head driven) PCFGs derived from the same trees.

The relational-realizational model can accommodate agreement information. It is shown in [146] that given gold-standard POS tags which include the gender and number information for individual words, RR models enriched with gender and number agreement information can provide Modern Hebrew parsing accuracies of 84% F_1 for sentences of up to 40 words, the highest reported number for Modern Hebrew parsing based on gold POS-tags and word-segmentation by the time of its publication.

While the RR framework is well motivated linguistically and appealing aesthetically, in the current work I chose to rely on the extreme markovization employed by the PCFG-LA Berkeley-parser in order to cope with the constituent order variation, and to model agreement as an external filter which is orthogonal to the grammar. The approach taken in this thesis provides state-of-the-art results for Hebrew constituency parsing. I leave the question of integrating the RR approach with the approach presented in this thesis to future work.

9.8 Chapter summary

This chapter reviewed existing constituency-parsing technology. Constituency parsing is mostly based on PCFG models. While stronger models of grammar are available, PCFGs have the benefit of admitting efficient ($O(n^3)$) parsing using the CKY algorithm. PCFG grammars can be extracted from Treebanks: the grammar rules are read from the trees, and the rule probabilities are based on smoothed Treebank counts. Trees can be annotated prior to grammar extraction. Tree-annotation adds information to each non-terminal, encoding more context and linguistic information than is available in the Treebank trees. Good tree annotation contributes to better parsing accuracies. The grammar rules in PCFG models are binarized. Binarized grammars are equivalent to non-binarized grammars. However, by markovizing the binarized rules ("forgetting" information encoded in the binarization procedure) one can come up with non-equivalent grammar which may be better than the original one in term of its generalization capabilities. The combination of tree annotation and grammar markovization results in remarkable parsing accuracies for English. Automatic category splitting (PCFG-LA) has led to further improvements in parsing accuracies. The standard metric for measuring parsing accuracy is precision and recall of labeled constituents. This metric assumes the trees of the gold and predicted parse trees share the same yield, but can be generalized to the case where the yields of the gold and predicted parse trees may differ.

Best parsing results in English are in the low nineties (90.05 for parsing Section 23 of the WSJ using the PCFG-LA Berkeley parser, and 90.8 using a two-stage parser [27, 114]). For languages other than English, accuracy results are lower. The best published results for parsing German (the TueBa-D/Z Treebank) and Arabic (the Penn Arabic Treebank, assuming gold segmentation) are 83.2% F_1 [115] and $\sim$82% F_1 [11], respectively. For Hebrew, the best published parsing results outside of this thesis are 84.1% F_1 [146] (for sentences of up to 40 words, assuming gold word-segmentation and POS-tags are available) and 77% F_1 [142] assuming gold segmentation.

Chapter 10

A Hebrew Constituency Parser

This chapter describes a system for constituency-parsing of Modern Hebrew, a language with rich morphology and a small Treebank.

The system is based on a state-of-the-art model for constituency parsing, namely, the PCFG with latent annotations (PCFG-LA) model of Petrov *et.al.* [113], as implemented in the BerkeleyParser. After evaluating the out-of-the-box performance of the BerkeleyParser on the Hebrew Treebank, I discuss some of its limitations and then go on to extend the PCFG-LA parsing model in several directions, making it more suitable for parsing Hebrew and related languages.
The system's design is motivated by the following guidelines:

- Segmentation and Parsing are closely related and are better performed jointly.
- The Treebank is too small to provide adequate lexical coverage.
- Morphological information, and in particular, morphological agreement, should help parsing.

Several aspects of the work presented in this chapter are discussed in earlier publications. [57] suggest the lattice-parsing mechanism, [58] discuss ways of interfacing a Treebank-derived PCFG-parser with an external lexicon, and [56] present experiments using the PCFG-LA Berkeley Parser. Here I provide a cohesive presentation of the entire system, as well as a more detailed description and an expanded evaluation. I also extend the previous work in several dimensions: I introduce a new method of interfacing the parser and the external lexicon, which contribute to an improved parsing accuracy, discuss the effect of various linguistically-motivated tree-annotations on parsing accuracy, and suggest to incorporate agreement information as a filter.

Setting	Tagset	F_1 (4 cycles)	F_1 (5 cycles)
Seg+POS Oracle	Core	89.7	89.5
Seg Oracle	Core	82.6	83.6
Pipeline	Core	76.3	77.2
Seg+POS Oracle	Core+Verbs	89.9	90.9
Seg Oracle	Core+Verbs	83.3	83.6
Pipeline	Core+Verbs	77.1	77.3

Table 10.1: *Baseline: out-of-the-box Berkeley-parser performance on the dev-set.*

10.1 Baseline experiments

The baseline system is an "out-of-the-box" PCFG-LA parser, as described in [113, 114] and implemented in the Berkeley-parser[1]. The parser is trained on the Modern Hebrew Treebank (see section 10.6 for the exact experimental settings) after stripping all the functional and morphological information from the non-terminals.

I test the resulting models on the development set, and consider three settings:

Seg+POS Oracle: The parser has access to the gold segmentation and POS-tag.

Seg Oracle: The parser has access to the gold segmentation, but not the POS-tags.

Pipeline: A POS-tagger is used to perform word-segmentation, which is then used as parser input.

A better tag-set Glossing over the parses revealed that the parser failed to learn the distinction between finite and non-finite verbs. The importance of this linguistic distinction for parsing is obvious, and was also noted in [76] for English and in our previous work on parsing Hebrew [57]. Finite and non-finite verbs are easily distinguishable from each other based on surface form alone. While finiteness is clearly annotated in the Treebank, it is not on the 'core" part of the POS-tags and was removed prior to training the parser. In a second set of experiments the core tagset of the parser was modified to distinguish finite verbs, infinitives and modals.[2] The original core-tagset already includes some important distinctions, such as construct from non-construct nouns.

Results and discussion Table 10.1 presents the parsing results on the development set.

With gold POS-tags and segmentation, the results are very high. Accuracy drops considerably when the parser is not given access to the gold tags: from about 90 to less

[1] http://code.google.com/p/berkeleyparser/

[2] Unlike previous work, the distinction is retained only at the POS-tag level and not propagated to the phrase level. The tag-level information is sufficient for the parser to learn the phrase-level distinctions on its own.

Tag	# Splits	Tag	# Splits
H	1	CDT	6
HAM	1	CC	7
POS	1	DT	7
REL	1	JJ	7
VB	1	VB-INF	7
AT	2	PRP	8
COM	2	CD	10
JJT	2	RB	13
QW	2	NN	16
RBR	2	NNP	17
VBMD	2	NNT	22
WDT	2	MOD	24
AGR	4	IN	26
AUX	6		

Table 10.2: *Number of learned splits per POS-category after five split-merge cycles*

than 84 F_1, indicating that the POS-tags are both informative and ambiguous. Results drop even further (from 84 to 77) in the pipeline case where the gold segmentation is not available, indicating that correct segmentation also provides valuable information to the parser and that segmentation mistakes are costly.

Enriching the tagset to distinguish modals, finite and infinite verbs proved useful, with an increase of about 1 F_1 points (absolute) after 4 split-merge-smooth cycles, and a smaller increase after 5 cycles. This stresses the importance of the core representation: the automatic learning procedure goes a long way, but it can be aided by linguistically motivated manual interventions in some cases.

10.1.1 Analyzing the Learned PCFG-LA Grammar

I begin by inspecting the splits at the part-of-speech level. Table 10.2 displays the number of splits learned for each of the parts-of-speech symbols.

Prepositions are the most heavily split, followed closely by the somewhat-generic MOD tag and the nouns.

Nouns and adjectives The noun and adjective splits are somewhat hard to decipher. Some of the groups are obvious, "things appearing after numbers", "last names", "parts-of-dates", "time related", "places", etc. Others are are much harder to decipher.

MOD For the general-modification POS-tags, most categories clearly single-out one or two words with very specific usage patterns, such as לא (no), גם (also), רק (only), אפילו (even), לשעבר (former), etc. The other categories are harder to interpret.

Verbs Finite-verbs are not split at all, even though they form an open-class category. Modal verbs are split into two groups: one of them is dominated by nine modals (אפשר,

Tag	# Splits	Tag	# Splits
FRAGQ	1	ADVP	16
INTJ	6	S	16
FRAG	7	PP	22
SQ	7	VP	22
PRN	8	PREDP	25
ADJP	14	NP	32
SBAR	14		

Table 10.3: *Number of learned splits per NT-category after five split-merge cycles*

יש, נראה, קשה, אין, ניתן, דומה, חשוב, נכון, roughly corresponding to the English *could*, *should*, *seem/appear*, *hard*, *shouldn't*, *possible*, *appear/seem*, *important*, *fitting/required*) while the second contains all the others. This is an interesting distinction, as the nine singled-out modals never take a subject, while the modals in the other group do.[3] Infinitive verbs are split into seven categories, six of which are dominated by one or two words each, while the last is a catch-all category.

Coordination and question-words Coordination words are heavily split, each of the categories dominated by one or two words, indicating different usage patterns. The question words מה (what) and מי (who) are singled out from the rest.

Gender/number agreement The verbs are not split at all, indicating that the learned grammar cannot model subject-verb agreement. Pronouns are split by type (personals, demonstrative, and subtypes of demonstratives), but not by gender and number. Noun and adjective splits are sometimes hard to decipher, but they do not exhibit any grouping based on gender or number properties, indicating that the grammar cannot model adjective-noun agreement. AGR category splits does show a clear division that follows gender and number, but it is unclear what is captured by this division as the information cannot interact with nouns, adjectives, verbs, or pronouns.

Grammar-level Splits As noted in [113], the latent state-splits learned for the grammar symbols are harder to analyze. Table 10.3 shows the number of splits learned for each grammar non-terminal. The NP category is the most heavily split, followed by predicative phrases, verb phrases, and PPs. With the exception of the FRAGQ category, all symbols are split into at least six substates. What information is encapsulated in the state splits?

One way of shedding some light on the meanings of the split-states is by using the

[3] In fact, the nine modals are very similar in characterization to the words identified in [100] as modals, while many of the modals in the other group are not necessarily considered as modal outside of the Treebank guidelines.

grammar in generation mode and by sampling word sequences from each of the states.[4] By looking at the resulting strings, one can sometimes infer the kinds of information encoded in the grammar.

NP The split-NPs encode phrase length (some splits result in very long NPs, some in very short, some in very specific one- or two-word patterns). They also encode the definiteness rules (either an NP is definite or not), the interaction between definiteness and the AT marker, and a limited interaction between definiteness and construct nouns. Other NP splits are dedicated to pronouns or to question words, or encode proper names, monetary units, and numbers.

SBAR The split-SBARs are split according to the word introducing the SBAR. In addition, some split-SBARs encode quoted and parenthetical items.

S The split-Ss differ by length. In addition, some S splits seem to be modeling verb-less sentences, variations in word order, and sentence-level coordination.

10.1.2 Limitation of PCFG-LA parsing of Modern Hebrew

The PCFG-LA baseline is a strong one, and is substantially higher than all previous reported results for Hebrew parsing in each of the setups (Seg+POS oracle, Seg Oracle, and no Oracle). However, I also identify some of its limitations, namely:

Missed splits The learning procedure is not perfect, and fails to capture some linguistically meaningful state-splits. When such splits are manually supplied (*i.e.,* the trivial split of verbal types) accuracy improves. Can we come up with further informative annotations to increase parsing performance even more?

Sensitivity to non-gold POS The substantial drop in accuracy when the POS-tags are unobserved and need to be predicted is staggering, suggests that the it is difficult for the parser to assign part of speech tags. Of the 698 part of speech errors, 314 are on words not seen in training. Can the parser's handling of unknown and infrequent words be improved?

[4]Sampling a word sequence is performed by starting at a given state (a split grammar symbol), randomly choosing a right-hand-side based on the PCFG induced distribution, expanding the state into the chosen right-hand side, and continuing recursively until we are left with only strings.

Sensitivity to non-gold segmentation The accuracy drops even further when the parser is presented with predicted segmentation. Segmentation errors are detrimental to the parser. Syntax and segmentation are closely related, and the syntactic structure should be able to guide segmentation decisions instead of being misled by erroneous ones. Can syntactic parsing and word-segmentation be performed jointly?

Not encoding grammatical agreement Finally, the learned grammar does not encode grammatical agreement. While the majority of the parser mistakes are due to the flexible constituent order or "standard" ambiguities such as coordination and PP attachment, a handful of them could be resolved using agreement information. Can notions of morphological agreement be encoded into the parsing process?

In what follows, I address these four limitations, and substantially increase the parser accuracy for the realistic case where gold segmentation and POS-tags are not available.

10.2 Manual state-splits

I experimented with several linguistically motivated state-splits which were added as tree-annotations prior to running the parser. Most of them did not help on their own and slightly degraded parser performance when combined with other splits. These include splits which were proven useful in previous work, such as marking of definite NPs, and distinguishing possessive from other PPs. I also experimented with splits based on morphological agreement features, which are discussed in section 10.5.1 below.

Overall, the learning procedure is capable of producing good splits on its own. I did, however, manage to improve upon it with the following annotation (the annotations were removed prior to evaluation).

Subject NPs Hebrew phrase order is rather flexible, and the subject can appear before or after the verb. Identifying the subject can thus help in grounding the overall structure of the sentence. The subject is also subject to agreement constraints with the verb. Klein and Manning [76] implicitly annotate subject-NPs in English using parent annotation (distinguishing NPs under S from other NPs), with good results. When applied to English, the PCFG-LA also learns to model subject NPs well. However, Hebrew's non-configurationality put both Subjects and Objects directly under S, making it much harder to learn the distinction automatically.

Explicit marking of subject NPs contributes slightly to the accuracy of the parser. Perhaps more important than the small increase in accuracy is the fact that the parser

can identify subject relatively well. In contrast, marking of object NPs did not help by itself and slightly degraded the parsing accuracy when combined with other annotations. Note, however, that Hebrew definite objects are already clearly marked using the את marker, making them an easy target for the parser.

10.3 Better lexical coverage with an external lexicon

The drop in parsing accuracy when gold core POS-tags are not available and need to be inferred by the parser is huge (from above 90 to less than 84 F_1). I attributed this drop to three related factors:

1. The POS tagging scheme in the Treebank is highly syntactic in nature: a part-of-speech is chosen to reflect the syntactic function of the given word in context. For example, demonstrative pronouns are tagged in the Treebank as adjectives when appearing in adjectival positions ("ילד זה", "this/JJ child/NN"), and a special MOD tag is used to mark non-adverbial clausal level modification (that is, modification which can be treated as adverbial but is used to modify something other than a verb). This has two effects: first, the gold-standard part-of-speech tags are highly indicative of the correct syntactic structure, and second, the tagging task under this scheme is made harder.

2. The small Treebank and large number of possible word forms (see Chapter 4) makes data-sparseness unavoidable. As noted in the error analysis, many of the POS-tagging mistakes occur on rare and OOV words. Overall, 9.5% of the word forms in the dev-set were never seen in training (on average, every sentence has 2.17 unseen word forms, even when gold segmentation is given). These numbers are much higher on non-Treebank texts. As noted in Chapter 4, guessing the correct POS tag for an unknown word based on surface-level features is difficult.

3. The relatively free word order makes the structure less indicative of correct POS-tag assignment. The parser is given a lot of flexibility in licensing many linguistic structures, and when it is not constrained by POS-tags it exploits this freedom in myriad creative ways – the tree structure is not rigid enough to effectively constrain the choice of tags for unknown words.

The large number of possible word forms make it very difficult for manually annotated corpora to provide adequate lexical coverage. The problem is even more severe with the case of the Hebrew Treebank, which is especially small. While it is big enough to

learn meaningful syntactic generalizations (as demonstrated by the high performance of the baseline system) it is far too small to learn a good lexical model (as evidenced by the drop in accuracy when gold tags are not available).

I suggest increasing the lexical coverage of the parser using an external resource, namely, a lexicon-based morphological analyzer. I further extend the utility of the analyzer with lexical tagging probabilities learned from an unannotated corpus.

10.3.1 A unified lexical probability model

I would like to use the KC Analyzer (section 4.2.2) to increase the lexical coverage of the Treebank-trained parser. That is, I would like to improve the lexical model $P(T \rightarrow W)$ of the generative parser. However, as discussed in section 4.2.4, the tagsets used by the two resources differ. How can this difference be reconciled?

One possibility is to re-tag the Treebank with the KC tagset and then train on this unified resource. In [58], we show that this procedure degrades parser performance. Instead, [58] suggest a layered generative approach, which retains the benefits of the Treebank tagging for frequent words and resorts to the KC tagset only for rare and unseen words. Under this approach, frequent words are generated from Treebank POS-tags as usual, but rare words follow a generative process in which first the Treebank tag generates a KC tag, and then the KC-tag generates the word. A sample derivation using this layered representation is presented in Figure 10.1.

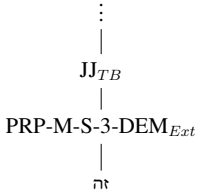


Figure 10.1: *A layered POS-tag representation*

The Treebank-to-KC tags generation probabilities represent a fuzzy, probabilistic mapping between the two resources. In [58], the estimation of these probabilities was done based on a re-tagging of the Treebank to use the KC tagset. The re-tagging process was far from trivial, and many tagging cases required extensive debates between human annotators.

Here, I present a new procedure which does not require the Treebank to be re-tagged with a new tagset. It still uses the layered representation, but instead of forcing one unique KC analysis for each location, it embraces the uncertainty and allows all of them. This is done by treating the KC-tag assignments as hidden variables, learning

the TB-KC mapping probabilities as part of the grammar training EM process, and marginalizing the KC tags out for the final tree.
The procedure is based on the following assumptions:

- We have access to trees in which the POS-tags t_{tb} are taken from a given tagset T_{TB}.
- We have additional access to an external resource (lexicon) mapping words to tags t_{ext} from a different tagset T_{Ext}.
- Probabilities involving words which are frequent in the Treebank can and should be based on Treebank counts.
- Probabilities involving less frequent words should be smoothed in with information from the external lexicon.
- Smoothing should have a greater effect on less-frequent words.
- Probabilities for unseen words should be based solely on the external lexicon.

Figure 10.2 illustrates the representation used for words which are rare or unseen in the Treebank training data.

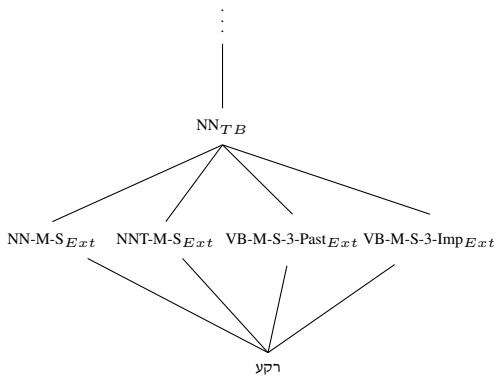


Figure 10.2: *A latent layered POS-tag representation*

The Treebank tag NN$_{TB}$ (upper level) generates the word-form רקע (lower level) by considering all the possible KC postags allowed for the word in the morphological analyzer (the middle level). The probabilities related to generating the KC postags are summed, and all the other probabilities are multiplied. The exact equations are detailed below.

While the needed quantity is the emission probability $P(T_{TB} \rightarrow W) = P(W|T_{TB})$, it is more convenient (for a reason which will be discussed later) to work with the

tagging probability $P(T_{TB}|W)$. Once the tagging probabilites $P(T_{TB}|W)$ are available, they can easily be converted to emission probabilities using Bayesian inversion, based on the relative-frequency estimates of $P(W)$ and $P(T_{TB})$ which are calculated from the Treebank:

$$\frac{P(t_{tb}|w)P(w)}{P(t_{tb})} = P(w|t_{tb}) = P(t_{tb} \rightarrow w) \tag{10.1}$$

Let us now focus on estimating the tagging probabilities $P(T_{TB}|W)$ for the cases of frequent, rare and OOV words.

For frequent words that are seen more than K times in the Treebank, we simply use Treebank-based relative-frequency estimates:[5]

$$P_{\text{tb}}(t_{tb}|w) = \frac{c(w, t_{tb})}{c(w)} \tag{10.2}$$

where $c(\cdot)$ is a counting function.

For OOV words which are not seen in the Treebank, the tagging probability is estimated using:

$$P_{\text{oov}}(t_{tb}|w) = \sum_{t_{\text{ext}} \in T_{\text{Ext}}} P(t_{\text{ext}}|w)P(t_{tb}|t_{\text{ext}}) \tag{10.3}$$

where $P(T_{\text{Ext}}|W)$ is a tagging probability using the external tagset, and $P(T_{TB}|T_{\text{Ext}})$ is a transfer probability relating the tags from the two tagsets (the estimation of these two probabilities is discussed below). What this does is assume a process in which the word is tagged by first choosing a tag according to the external lexicon, and then choosing a tag from the TB tagset based on the external one. The external tag assignments are then treated as latent variables, and are marginalized out.

Finally, for rare words that are seen only a few times in the Treebank, we interpolate the two quantities, weighted by the word's frequency in the Treebank:

$$P_{\text{rare}}(t_{tb}|w) = \frac{c(w)P_{\text{tb}}(t_{\text{tb}}|w) + P_{\text{oov}}(t_{tb}|w)}{1 + c(w)} \tag{10.4}$$

We now turn to describing the estimation of the external tagging probability $P(T_{Ext}|W)$ and the tag transfer probability $P(T_{TB}|T_{Ext})$.

[5] In practice, small amount of smoothing is added to allow tagging a word with open-class tags it wasn't seen with in the Treebank: $P_{\text{tb}}(t_{tb}|w) = (c(w, t_{tb}) + 0.0001 * P(t_{tb}))/(c(w) + 0.0001)$

Estimating $P(T_{Ext}|W)$ The tagging probability follows the morphological analyzer. The analyzer provides the possible analyses, but does not provide probabilities for them. One simple option would be to assign each possible analysis (tag) a uniform probability, and assign 0 probability for tags not allowed by the lexicon for the given word. This method is referred to as $P_{\text{unif}}(T_{Ext}|W)$. However, we know that not all the possible analyses for a given word are equally likely, and in practice, the actual tagging distribution is usually biased toward one or two of the tags. These tagging preferences can be learned in an unsupervised manner given the lexicon and a large corpus of unannotated text, using EM training of an HMM tagging model. Adler and Elhadad [5] suggest such a model for accurate tagging of Hebrew, and Adler [4] and Goldberg *et.al.* [53] extend it to provide state-of-the-art tagging accuracies for Hebrew using a smart initialization. Here, I use the pseudo-counts from the final round of EM training in this tagging model in order to compute $P_{\text{em}}(T_{\text{Ext}}|W)$. I show in section 10.6 that this unsupervised lexical probabilities estimation does indeed provide better parsing results.

Estimating $P(T_{TB}|T_{Ext})$ The tagset-transfer probabilities capture the patterns of transfer between the syntactic tagging scheme of the Treebank and the other tagging scheme of the external resource. They are estimated using Treebank counts and the tagging distribution $P(T_{Ext}|W)$:

$$P(t_{\text{tb}}|t_{\text{ext}}) = \frac{c(t_{\text{tb}}, t_{\text{ext}})}{c(t_{\text{ext}})} = \frac{\sum_w c(t_{\text{tb}}, w) P(t_{\text{ext}}|w)}{\sum_w P(t_{\text{ext}}|w)} \tag{10.5}$$

Integration into the PCFG-LA model I incorporate the estimation procedure into the training process of the PCFG-LA model. While the external tagging probabilities $P(T_{\text{Ext}}|W)$ are fixed, the other distributions are re-estimated in the EM process following each of the split, merge and smooth stages. This is done by replacing all the corpus counts $c(\cdot)$ by pseudo-counts of the same events based on marginal counts computed during the EM procedure.

Note that in the PCFG-LA model the Treebank tagset T_{TB} is gradually split, and each tag takes the form $\langle \text{tag}, \text{substate} \rangle$, where $substate$ is a latent variable indicating a specific split of the given tag. This means that the Treebank tagging probability and the tagset-transfer probabilities are also defined over these split tags. This follows naturally from the training procedure.

The main reason for using the Bayesian inversion (Eq. 10.1) instead of working with the emission probability $P(W|T)$ directly is that the emission probability is highly dependent on the vocabulary size. The Treebank estimates are based on a small vocabulary, the external lexicon estimates are based on a very large vocabulary, and a proper

combination of the two emission probabilities is not trivial. In contrast, the tagging probabilities do not depend on the vocabulary size, allowing a very simple combination. I can then base the counts for the emission probability on the Treebank vocabulary alone, and estimate $P(W)$ for words unseen in training as if they were seen once.

10.4 Joint segmentation and parsing

When applied to real text (for which the gold word-segmentation is not available), the baseline PCFG-LA parser is supplied with a word segmentation produced by a separate tagging process.[6] This seriously degrades parsing performance. A major reason for the performance drop is that the word-segmentation task and the syntactic-disambiguation task are highly related. Segmentation mistakes drive the parser toward wrong syntactic structures, and many segmentation decisions require long-distance information that is not available to a sequential process [140]. For these reasons, I claim that parsing and segmentation should be performed jointly.

Joint segmentation and parsing can be achieved using *lattice parsing*. Instead of parsing over a fixed input string, the parser operates on a lattice – a structure encoding all the possible segmentations.

10.4.1 Lattice representation

Formally, a lattice is a DAG in which all paths lead from the initial state to the end state.

For the Hebrew segmentation task, all word segmentations of a given sentence are represented using a lattice structure. Each lattice arc corresponds to a word and its corresponding POS tag, and a path through the lattice corresponds to a specific word-segmentation and POS-tagging of the sentence. This is by now a fairly standard representation for multiple morphological segmentations of Hebrew utterances [3, 4, 13, 31, 53, 56, 57]. It is also used for Arabic [59] and other languages [131].

Figure 10.3 depicts the lattice for a two-words sentence בצלם הנעים[7]. Double-circles indicate the space-delimited token boundaries. Note that in this construction arcs can never cross token boundaries. Every token is independent of the others, and the sentence lattice is in fact a concatenation of smaller lattices, one for each token. Furthermore,

[6] While the tagger also produces POS-tags assignments, we ignore them and use only the word-segmentation. This is done for two reasons: first, the tagset of the tagger is the one used by the morphological analyzer, and is not compatible with the Treebank. Second, I believe it is better for the parser to produce its own tag assignments.

[7] While Hebrew is written right-to-left, the lattice is to be read left-to-right. The words on each arc follow the Hebrew writing directions, and are written right-to-left.

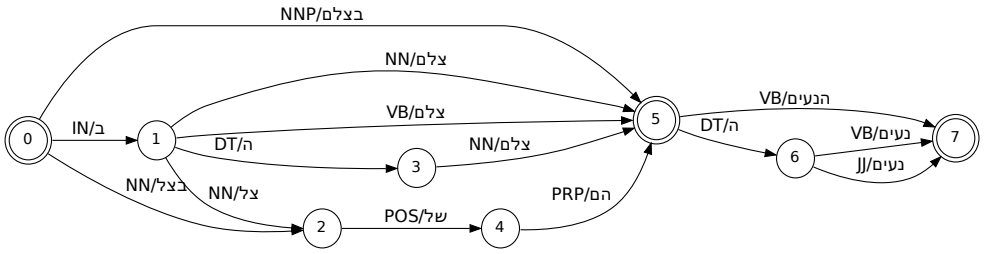


Figure 10.3: *The lattice for the Hebrew sequence* בצלם הנעים[7].

some of the arcs represent lexemes not present in the input tokens (*e.g.*, ה/DT, של/POS), however these are parts of valid analyses of the token. Segments with the same surface form but different POS tags are treated as different lexemes, and are represented as separate arcs (*e.g.*, the two arcs labeled נעים from node 6 to 7).

A similar structure is used in speech recognition. There, a lattice is used to represent the possible sentences resulting from an interpretation of an acoustic model. In speech recognition the arcs of the lattice are typically weighted in order to indicate the probability of specific transitions. Given that weights on all outgoing arcs sum up to one, weights induce a probability distribution on the lattice paths. In sequential tagging models such as [5, 14, 131] weights are assigned according to a tagging model based on linear context. For the case of parsing, context-free weighting of lattice arcs is used: each arc corresponds to a $\langle tag, word \rangle$ pair, and is weighted according to the emission distribution $P(tag \rightarrow word)$.[8]

10.4.2 Lattice parsing

The CKY parsing algorithm can be extended to accept a lattice, instead of a predefined list of tokens, as its input [25]. The CKY search then finds a tree spanning from the start-state to the end-state of the lattice, where the leaves of the tree are lattice arcs. The lattice extension of the CKY algorithm is performed by indexing lexical items according to their start- and end-states in the lattice instead of by their sentence position,

[8] In the work of [31], lattice arc weights are assigned based on aggregate quantities (forward-backward tagging marginals) derived from a discriminative CRF tagging model. This approach is problematic from a modeling perspective, as it makes each POS-tag be accounted for twice: once by the syntactic model, and once by the sequential one. In this work, a sequential tagging model is not used at all. If the use of a sequential model is desired, an alternative method for integrating of a sequence model and a syntactic model is making the models "negotiate" an agreed upon structure which maximizes the score under both models, using optimization techniques such as dual decomposition [38] which was recently introduced into natural language processing [120].

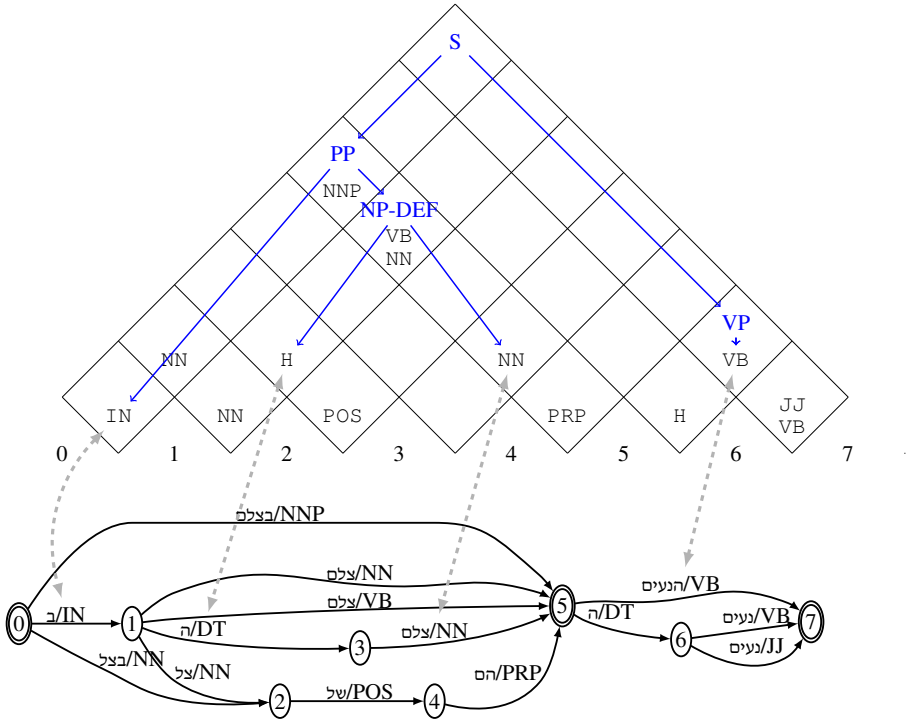


Figure 10.4: *Lattice initialization of CKY-chart*

and changing the initialization procedure of CKY to allow terminal and preterminal symbols of spans of sizes > 1. It is then relatively straightforward to modify the parsing mechanism to support this change: not giving special treatments for spans of size 1, and distinguishing lexical items from non-terminals by a specified marking instead of by their position in the chart.

Figure 10.4 shows the CKY chart for the lattice in Figure 10.3, together with an (incorrect) parse over the lattice. The chart is initialized with parts of speech corresponding to the lattice arcs. Phrase-structures are then built on top of the POS-tags (in blue). The proposed structure must span the entire chart, and correspond to a path through the lattice from the initial state (0) to the last one (7).

In training time the correct segmentation is fully observed, and the generative parser is trained as usual over the Treebank. In inference (test) time, the correct segmentation is unknown, and the decoding is applied to the segmentation lattice. The best-derivation returned by the parser forces a specific segmentation. The returned parse tree is the most probable $\langle segmentation, tree \rangle$ pair according to the grammar.[9]

I modified the PCFG-LA Berkeley parser to accept lattice input at inference time.

Lattice parsing allow to preserve the segmentation ambiguity and present it to the

[9]Note that finding the most probable *segmentation* requires summing over all the trees resulting in each segmentation: a much harder task.

parser, instead of committing to a specific segmentation prior to parsing. This way segmentation decisions are performed in the parser as part of the global search for the most probable structure, and can be affected by global syntactic considerations. I show in section 10.6 that this methodology is indeed superior to the pipeline approach.

10.5 Incorporating morphology

Inspecting the learned grammars reveals that they do not encode any knowledge of morphological-agreement: the split categories for nouns, verbs and adjectives do not group words according to any relevant morphological property such as gender or number, making it impossible for the grammar to model agreement patterns. At the same time, inspecting some of the bad parses reveals several clear cases of agreement mistakes. Can morphological agreement be incorporated in the parsing model?

10.5.1 Forcing morphologically-motivated splits

My initial attempts focused on making the PCFG-LA learning procedure picking on agreement-relevant state-splits. When neither the core tagset nor the non-terminals encode gender and number information, it is very hard for the parser to pick up on agreement patterns.[10]

I attempted to train the parser on trees which mark the agreement features (either the gender, the number or both) either on the POS-tags, the relevant constituents, or both.

Annotating agreement features on the POS-tag level made the parsing much slower, but did make the parser assign certain split categories to certain gender-number combinations, and sampling utterances from the learned grammar did indicate a notion of grammatical agreement. However, this did not improve parsing accuracy, and slightly degraded it.

When propagating the agreement features and annotating them on the constituent level, parsing accuracy dropped considerably. When inspecting the learned grammar we observe that most of the agreement-annotated constituents (*e.g.,* $\text{NP}_{\text{Masc,Plural}}$) were still fully-split, indicating that the parser picked on patterns which were orthogonal to the agreement mechanism. The pre-splitting according to agreement-features properties caused data sparseness, aided over-fitting and hurt parsing performance: the smoothing procedure of the Berkeley-parser shares some probability-mass between various splits

[10] In the external lexicon case, the external lexicon tags do encode the morphological features, making it possible in principle for the parser to learn to map certain substates to certain agreement features. This did not happen in practice. I suspect that this is because other structural factors were more powerful than the agreement ones.

of the same symbol, but was not applied in our case (no information flowed between, *e.g.*, $\text{NP}_{\text{Masc,Plural}}$ and $\text{NP}_{\text{Masc,Singular}}$). I attempted to counter this effect by changing the smoothing mechanism of the Berkeley-parser to share information also between the manually-split symbols. This brought parsing accuracy back to the initial level, but also caused the parser to, again, not model agreement very well. The reason for this is clear in hindsight: morphological agreement is an absolute concept, not a fuzzy one (things can either agree or not). Smoothing the probabilities between the different morphology-based splits licensed grammar rules which allow morphological disagreement, and made the grammar lose its discrimination power. This was then reinforced by the training process, which picked on other syntactic factors instead, and further fazed out the agreement knowledge.

A note on product-grammars In recent work, Petrov [112] showed that a committee of latent-variable grammars encoding different grammatical preferences can be combined into a product-grammar which is better than the individual ensemble members. Petrov created the ensemble by training several PCFG-LA parsers on the same data, but using different random seeds when initializing the EM starting point. I attempted to create a similar ensemble by providing the learning process with different linguistically-motivated tree annotations (*e.g.*, with and without encoding agreement features, with and without encoding definiteness, etc.). The combined parser did increase the performance level over that of the individual parsers, but an ensemble with the same number of components which was produced using the random-seeds approach produced far superior results. This reinforces the findings of Petrov (2010) who also report the ensemble creation using random initialization is exceptionally strong and outperforms other methods of ensemble creation.

10.5.2 Agreement as filter

I now turn to suggest a different approach to modeling agreement, which rests on the following principles:

- Agreement can be modeled as a set of hard (not probabilistic) constraints.
- Agreement is completely orthogonal to the other aspects of the grammar.

Based on these principles, I suggest to treat agreement as a filter, a device which can rule-out illegal parses. Under the agreement-as-filter framework, we want the parser to produce the most probable parse according to its grammar *and subject to hard agreement constraints*. This approach completely decouples the grammar from the agreement

verification mechanism. The agreement information is not modeled in the grammar and is not used to guide the search for the best parse. Instead, it is a separate process that imposes hard-constraints on the search space and rules out parts of it completely. That is, agreement is a part of the parser and not of the grammar.

Grammatical agreement is a relation between constituents. The relevant morphological features are propagated from one of the leaves up to the constituent level. When constituents are combined to form a larger constituent, their morphological features are assigned to the newly created constituent according to language specific rules (it is possible that different morphological features will be assigned by different constituents). An agreement violation occurs when two or more constituents assign conflicting features to their parent.

Implementation In the implementation, an agreement verification mechanism is manually constructed (not learned) based on a set of simple, language-dependent rules. First, I provide a set of rules to propagate the morphological agreement features from the leaves to the constituents. Then, I specify an additional set of rules to inspect local tree configuration and identify agreement violations (the Hebrew set of rules is described later, along with a concrete example). The feature-propagation mechanism works bottom-up and the agreement verification rules are very local, making it possible to integrate the filtration mechanism into a bottom-up CKY parsing algorithm (refusing to complete a constituent if it violates an agreement constraint). However, I did not pursue this route for the experiments in this thesis. Instead, I opted for an approximation in which I take the 100-best trees for each sentence, and choose the first tree that does not have an agreement violation (this is an approximation because the 100-best trees may not contain a valid tree, in which case we accept the agreement violation and choose the 1st-best tree).

Verifying the hard-constraint property I verified that the hard constraint assumption works and that the agreement verification mechanism is valid, by applying the procedure to the gold-standard trees in the training-set, and checking that (1) the propagated features agree with the manually marked ones, and (2) none of the training-set trees were filtered due to agreement violation. I did find a few cases in which the propagated features disagreed with the manually marked ones, and few gold-standard trees which the mechanism marked as containing an agreement violation. All of these cases were due to mistakes in the manual annotation.

10.5.3 The Hebrew agreement filter

Hebrew syntax requires agreement in gender, number and person. The implementation considers only the gender and number features, which are the most common. Each of the features can take one of five values as detailed in Table 10.4

Feature	Possible Values
Gender	Masculine, Feminine, Both, Unknown, NA
Number	Singular, Plural, Both, Unknown, NA

Table 10.4: *Agreement-feature values*

Masculine, Feminine, Singular and Plural are self-explanatory, and are assigned when the feature value is obvious. "NA" means that the feature is irrelevant for the given constituent (adverbs and PPs do not carry gender or number features). "Both" and "Unknown" are assigned when we are uncertain about the corresponding feature value. Both and Unknown are identical in the sense that they leave the feature value unspecified, and have the same effect on the filtration process. From a practical perspective they could be collapsed into the same category. I chose to maintain the distinction between the two cases because they have slightly different semantics. "Both" indicates that both options are possible (for example, the form ילדות is ambiguous between the plural "girls" and the singular "childhood", and the titular דר, "Dr." can refer both to males and females), while "Unknown" means that the feature value could not be computed due to a limitation of the model (for example, there is no clear rule as to the gender of a conjunction which coordinates masculine and feminine NPs, and we are currently unable to accurately infer the gender and number associated with certain complex quantifiers such as רוב ("most"). Compare: רוב הכיתה נשאר, רוב העוגה נאכלה, רוב העוגה נאכל ("*most of the class*$_{fem}$ *stayed*$_{masc}$", "*most of the cake*$_{fem}$ *was eaten*$_{fem}$", "*most of the cake*$_{fem}$ *was eaten*$_{masc}$")).

Feature values are said to *agree* if they are compatible with each other. Feminine is compatible with NA, Both and Unknown but not with Masculine. Similarly, Singular is compatible with NA, Both and Unknown, but not with Plural.

Agreement cases The system is designed to handle the following cases of morphological agreement:

NP level agreement between nouns and adjectives. ארגז תפוחים ירוקים גדולים ("box-of$_{Sg}$ apples$_{Pl}$ green$_{Pl}$ big$_{Pl}$") , ארגז תפוחים ירוקים גדול ("box-of$_{Sg}$ apples$_{Pl}$ green$_{Pl}$ big$_{Sg}$")

S level agreement between subject and verbs. אחד הילדים הלך ("[one-of the-kids]$_{Sg}$ walked$_{Sg}$")

Predicative agreement between the subject, ADJP and copular/auxiliary element.
הוא חכם ("he (is) smart$_{masc}$"), היא היתה מדהימה ("she was amazing/Fem"), but not with nouns היא היתה סמל ("she was a-symbol$_{masc}$").

Agreement between the Verb in a relativized SBAR and the realization of the Null-subject in the external NP.
הוועדה ש דנה בנושא ("the-committee$_{fem}$ which (*) discussed$_{fem}$ the-matter")

Morphological feature propagation The first step of determining agreement is propagating the relevant features from the leaves up to the constituent level.

The procedure begins by assigning each leaf gender and number features. These are assigned based either on the TB tag assigned for the word if training on gold POS-tags, or on the morphological analyzer entries for the given word (in most cases the number and gender features are easy to predict, even in cases where the core POS is not clear. In the relatively rare cases where the analyzer contains both a feminine and masculine (alt. singular and plural) analyses, feature value is marked as "both"). Words tagged by the parser as one of IN, DT, WDT, POS, H, or VB-INFINITIVE carry neither gender nor number and both features are assigned the 'NA' value.

After each leaf is assigned feature values, the features are propagated up the tree according to a set of rules such as the following (the complete set of rules is given in Table 10.5):

- If the constituent is an NP and has a Construct-noun child, it is assigned the gender of the Construct-noun.
- If the constituent is a coordinated NP (has a CC child) sets its number feature to plural.
- If the constituent is an SBAR and it has both a REL and an S children, take the gender number from the S.
- If the constituent is an S and it has VP child but no NP-Subject child, take the gender from the VP.
- If the constituent is an S and it has an NP-Subject child, assign its gender to NA.

Agreement rules Once the features are propagated from the leaves to a constituent, agreement is verified at the constituent level according to the following rules:

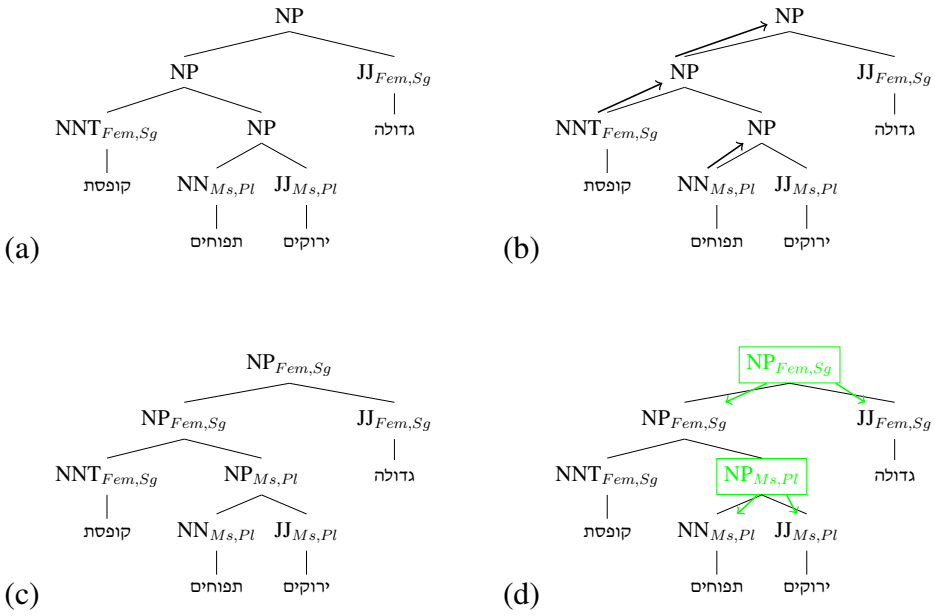


Figure 10.5: *Agreement annotation and validation example: correct tree. The sentence words translate to "box-of apples green big", literally "a big box of green apples".*

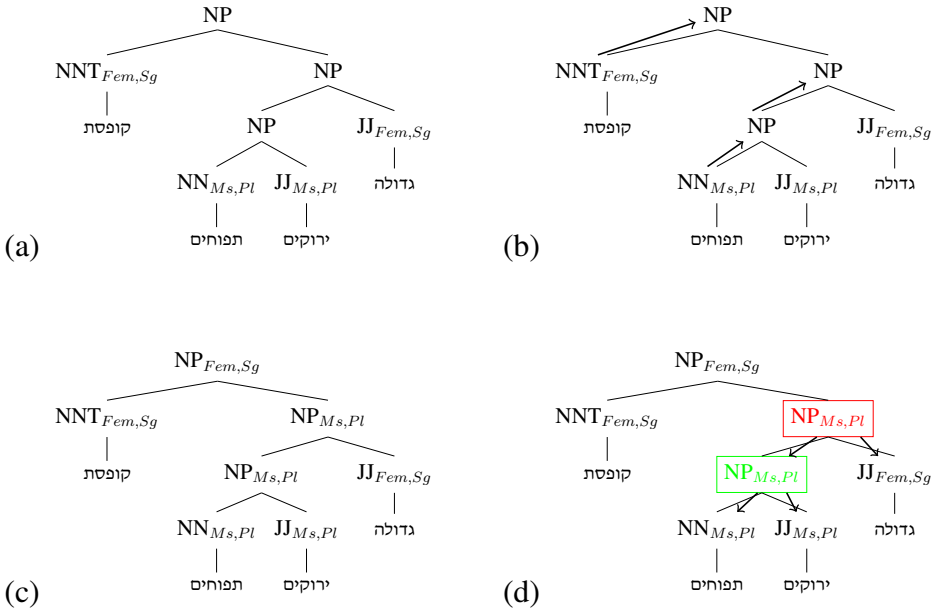


Figure 10.6: *Agreement annotation and validation example: incorrect tree, agreement violation. "box-of apples green big", literally "a big box of green apples", though the parse tree suggests the interpretation "a box of big green apples".*

Constituent	Condition	Feature Values
SBAR	has REL and S children	S.features
SBAR	otherwise	NA
PREDP	has ADJP child	ADJP.features
PREDP	has AGR child and no NP child	AGR.features
PREDP	otherwise	NA
S	has VP child and no NP-Subj child	VP.features
S	has VB child and no NP-Subj child	VB.features
S	otherwise	NA
NNPG	always	U
NP	has NNT child	NNT.features
NP	has CDT and NP children	CDT.number NP.gender
NP	is a conjunction	gender=U number=Plural
NP	has a " child	U
NP	first child is NP, second is POS	NP.features
NP	has IN child	FC.gender number=U
NP	has child with non-NA gen/num	FC.gender FC.number
NP	otherwise	NA
ADJP	has JJT child	JJT.features
ADJP	has child with non-NA gen/num	FC.gender FC.number
ADJP	otherwise	NA
VP	has VB child	VB.features
VP	has VB-Modal child	VB-Modal.features
VP	has VP child	VP.features
VP	otherwise	NA
other	always	NA

Table 10.5: *Gender and Number percolation rules. 'FC' is first child with non-NA gender/number. Rules for each constituent type are applied in order, until a condition holds. Rules for gender and number are applied independently of each other.*

NP agreement rules:

- Agreement for coordinated NPs and Possessive NPs is not checked.
- If NP has an SBAR child, all the children up to the SBAR whose type is nominal or adjectival must agree in gender and number.
- If NP has an ADJP child, all the children up to the ADJP whose type is nominal or adjectival must agree in gender and number.

S agreement rule:

- All children of S with type in {NP-Subject, VP, VB, AUX, PREDP} must agree in their gender and number features.

ADJP agreement rule:

- All children of ADJP with type in {NP, NP-Subject, NN, JJ, ADJP} must agree in their gender and number features.

An example Consider the tree in Figure 10.5a. In the first stage (Fig. 10.5b), agreement features are propagated according to the rules in Table 10.5, resulting in the annotated tree in Figure 10.5c. Agreement is then validated in Figure 10.5d (nodes in which an agreement rule applied and passed are marked in green) . In contrast, the tree in Figure 10.6a has an agreement mistake. As before, the agreement features are propagated according to the rules (Fig. 10.6b) resulting in Fig. 10.6c. Agreement validation fails at 10.6d (the node in which agreement validation was applied and failed is marked in red).

10.6 Evaluation and results

Dataset For all the experiments I use Version 2 of the Hebrew Treebank [107], with the established test-train-dev splits: Sentences 484-5740 are used for training, sentences 1-483 are the development set, and sentences 5741-6220 are used for the final test-set.

Effect of external lexicon I start by evaluating the effect of extending the parser's lexical model with an external lexicon, as described in section 10.3.1. The rare-word threshold is set to 100. I use the morphological analyzer described in section 4.2.2. I test two conditions: UNIFORM, in which the $P(T_{\text{ext}}|w)$ distribution is uniform over all the analyses suggested by the morphological analyzer for the word, and HMMBASED in

which the $P(T_{\text{ext}}|w)$ distribution is based on pseudo-counts from the final round of EM-HMM training of the semi-supervised POS-tagger described in section 4.2.3. Results are presented in Table 10.6.

Setting	Ext-Lexicon/Probs	F_1 (4 cycles)	F_1 (5 cycles)
Seg Oracle	NONE	83.13	83.39
Pipeline	NONE	75.98	76.65
Seg Oracle	UNIFORM	84.92	84.56
Pipeline	UNIFORM	77.53	77.35
Seg Oracle	HMMBASED	86.17	85.79
Pipeline	HMMBASED	78.75	78.78

Table 10.6: *Dev-set results when incorporating an external lexicon*

Incorporating the external lexicon helps both in the case where the correct segmentation is assumed to be known, as well as in the pipeline case where the segmentation is automatically induced by a sequential tagger. Incorporating the semi-supervised lexical probabilities learned over large unannotated corpora (HMMBASED) further improves the results, up to 86.1 F_1 for the gold-segmentation case and 78.7 F_1 for the pipeline case. The pipeline model still lags behind the gold-segmentation case, indicating that the correct segmentation is very informative for the parser.

Joint segmentation and parsing Having established that the external lexicon can be effectively incorporated into the parser, I turn to evaluate the method for joint segmentation and parsing. I follow the same conditions as before (UNIFORM and HMMBASED lexical probabilities), but in this set of experiments the parser is allowed to choose its preferred segmentation using the lattice-parsing methodology presented in section 10.4.2 above. The lattice is constructed according to the analyses licensed by the morphological analyzer. Table 10.7 lists the results.

Setting	Ext-Lexicon/Probs	F_1 (4 cycles)	F_1 (5 cycles)
Pipeline	UNIFORM	77.53	77.35
Lattice (Joint)	UNIFORM	80.35	80.31
Pipeline	HMMBASED	78.75	78.78
Lattice (Joint)	HMMBASED	80.91	80.46

Table 10.7: *Dev-set results when using lattice-parsing on top of an external lexicon/analyzer*

Lattice-parsing is effective, leading to an improvement of about 2-3 F_1 points over the pipeline model.

Agreement filter I now turn to add the agreement filtering on top of the lexicon-enhanced models. In this setting, the model outputs its 100-best trees for each sentence, agreement features are propagated and agreement violations are checked as described in section 10.5.3 above, and the first tree that does not contain any agreement violation

is returned as the final parse for the sentence (or the first-best tree in case that all of the output trees contain an agreement violation). Table 10.8 lists the results when agreement filtering is performed on top of parses based on gold-segmentation, and Table 10.9 lists the results when agreement filtering is performed on top of a lattice-based parsing model which does not assume gold-segmentation is available.

Setting	Ext-Lexicon/Probs	F_1 (4 cycles)	F_1 (5 cycles)
No Agreement	UNIFORM	84.92	84.56
Agreement as Filter	UNIFORM	85.30	84.52
No Agreement	HMMBASED	86.17	85.79
Agreement as Filter	HMMBASED	86.55	86.25

Table 10.8: *Dev-set results of using the agreement-filter on top of the lexicon-enhanced parser (starting from gold segmentation).*

Setting	Ext-Lexicon/Probs	F_1 (4 cycles)	F_1 (5 cycles)
No Agreement	UNIFORM	80.35	80.31
Agreement as Filter	UNIFORM	80.55	80.74
No Agreement	HMMBASED	80.91	80.46
Agreement as Filter	HMMBASED	81.04	80.72

Table 10.9: *Dev-set results of using the agreement-filter on top of the lexicon-enhanced lattice parser (parser does both segmentation and parsing).*

While the agreement filter does not hurt the parser performance, the benefits from it are very small. This is because the grammar is strong enough to produce fairly accurate structures, which have very few agreement mistakes to begin with. In addition, fixing an agreement mistake does not necessarily mean fixing the entire parse – in some cases it is very easy for the parser to fix the agreement mistake and still produce an incorrect parse for other parts of the structure. The agreement filter is useful in overcoming agreement errors and providing better parses, but its total effect on the parsing score is small.

10.7 The final model

Finally, I evaluate the best performing model on the test set. Table 10.10 presents the results of parsing the test-set while incorporating the external lexicon and using the HmmBased probabilities, for a grammar trained for four split-merge iterations. This grammar is applied both to the gold-segmentation case and to the realistic case where segmentation and parsing are performed jointly using lattice-parsing. I also test the effectiveness of the agreement-filter in both situations.

Agreement information does not hurt performance, but contributes very little to the final accuracy – also on the test sentences, the parser makes very few agreement mistakes to begin with.

Setting	Model	F_1 (4 cycles)
Gold Segmentation	HmmBased External Lexicon	85.67
	+ Agreement	85.70
Lattice-parsing	HmmBased External Lexicon	76.87
	+ Agreement	76.95

Table 10.10: *Test-set results of the best-performing models*

Consistent with previous reports [142], the test-set is somewhat harder than the development set. With gold-segmentation, the models achieve accuracies of 85.70% F_1. In the realistic scenario in which the segmentation is induced by the parser, the accuracies are around 76.9% F_1. I verified that the HmmBased lexical probabilities outperform the Uniform probabilities also on the test-set (the F_1 scores when using uniform lexical probabilities are 84.06 and 76.30 for the gold and induced segmentations, respectively). These are the best reported result for parsing the test-set of the Hebrew Treebank.

10.8 Summary

This chapter presented experiments on Hebrew Constituency Parsing based on the PCFG-LA methodology of Petrov *et.al.* [113]. The PCFG-LA model performs well out-of-the-box, especially when the gold POS-tags are available to the parser. However, it is possible to improve the learned grammar by specifying some manual state-splits, specifically distinguishing between modal, finite and infinitive verbs, and explicit marking of subject-NPs.

Parsing accuracies drop considerably when the gold POS-tags are not available, and drop even further when using non-gold segmentation. A large part of the drop when the gold POS-tags are not available is due to the large percentage of lexical events which are unseen or seen only a few times in the training set. This drop can be mitigated by extending the lexical coverage of the parser using an external lexical resource such as a wide-coverage morphological analyzer for mapping lexical items to their possible POS-tags. The POS-tagging schemes assumed by the Treebank and the morphological analyzer need not be compatible with each other: I present a method for bridging the POS-tags differences between the two resources. The morphological analyzer does not provide lexical probabilities. Parsing accuracies can be further improved by using lexical probabilities which are derived in a semi-supervised fashion based on the morphological analyzer and a large corpus of unannotated text.

The correct token-segmentation is very important for achieving high-quality parses, and when the gold segmentation is not available, parsing results drop considerably. It is better to let the parser induce its preferred segmentation in interaction with the pars-

ing process rather than to use a segmentation based on an external sequence model in a pipeline fashion. The joint induction of both the syntactic structure and the token-segmentation can be performed by representing the possible segmentations in lattice structure, and using lattice-parsing. Joint parsing and segmentation is shown to outperform the pipeline approach. However, the parsing accuracies with non-gold segmentation are still far below the accuracies when the gold-segmentation is assumed to be known, and accurate parsing with non-gold segmentation remains a challenging open research problem.

The learned PCFG-LA grammar is not capable of modeling agreement information. I considered methods of using morphological agreement information to improve parsing accuracy. I propose modelling agreement information as a filtration process which is orthogonal to the grammar used for parsing. The approach works in the sense that, in contrast to other methods of using agreement information, it does not degrade parsing accuracy and even improves it slightly. However, the benefit from the agreement filtering is small: with the strong grammar induced by the PCFG-LA training procedure, the parser makes very few agreement mistakes to begin with. Modeling morphological agreement is probably more useful in syntactic generation than in syntactic parsing. I expect the filtration approach I propose in this chapter to be proven useful for tasks involving syntactic generation, such as syntax-based machine translation into a morphologically rich language.

Overall, I presented four enhancements to the PCFG-LA mechanism in order to adapt it to parsing Hebrew: the introduction of manual, linguistically-motivated state-splits, extending the lexical coverage of the parser using an external morphological analyzer, performing segmentation and parsing jointly using a lattice parser, and incorporating agreement information in a filtering framework. Together, these enhancements result in the best published results for Hebrew Constituency Parsing to date.

Chapter 11

Conclusions and Future Work

This dissertation describes my attempts at creating computational systems for the automatic syntactic analysis of Modern Hebrew text. The immediate contribution of the work is the availability of concrete software systems that can automatically analyze Hebrew text and produce dependency- and constituency-based syntactic representations for Hebrew sentences, with good accuracy. A secondary concrete product is the Hebrew Dependency Treebank, described in Chapter 5. These resources are central building blocks in automatic language processing of Modern Hebrew text, and can be used by researchers in the humanities (literature, social sciences, history), linguistics, and researchers and practitioners in the computational fields of natural language processing, text mining and information extraction.

Beyond concrete deliverables, the dissertation also explores the process of creating a parser for a language with rich morphology and only few annotated resources. The following themes are central to the overall approach I pursued:

Utilizing existing methods: When possible, it is best to utilize existing methods. However, direct application of a method designed for language A may not work optimally for language B. One should consider the specificities of the language being analyzed, and adapt the method to the language at hand.

For the case of Hebrew constituency parsing (Chapter 10), I built upon PCFG-LA [113], a robust, mostly language independent parser. I extended the parser to not rely solely on the Treebank for lexical coverage, and instead use an external, wide coverage lexicon. The available lexicon followed a different annotation scheme than the one employed in the Hebrew Treebank, and I created a model to work around this discrepancy. In addition, I borrowed the lattice-parsing technique from the speech-processing community [25], adapted it to deal with Hebrew token-segmentation, and integrated it into the PCFG-LA parsing process. These two extensions, together with a transformation of

the input trees that help the parser training process capture important linguistic distinctions (separating finite verbs, infinitive and modals, as well as distinguishing subject noun-phrases from the rest of the noun-phrases), result in a substantial increase in accuracy for parsing Hebrew compared to the original, out-of-the-box PCFG-LA model.

For the case of dependency parsing (chapters 7 and 8), I present a novel parsing algorithm, which works well for both English and Hebrew (as well as other languages, as explored by independent researchers). The algorithm is based on a left-to-right transition-based parser, but extends it by replacing the left-to-right processing order with an easy-to-hard processing order, which allows the use of a richer feature set. Research in this direction stemmed from the realization that a machine-learning-based dependency parser for Hebrew requires kinds of features that can not be effectively represented and utilized by existing dependency parsing methods. These kinds of features turned out to be useful for parsing English as well (though an independent group of researchers [139] came up with a specialized feature set which is better suited for parsing English, presenting further evidence for the need to tailor systems for specific languages). In addition, I also present an extended feature set which is based on Hebrew-specific morphological agreement patterns. This feature set improves the parser's accuracy on Hebrew even further. The existing MST2 and ARCSTN parsers, while not as accurate for parsing Hebrew as the proposed EASYFIRST parser, also perform fairly well, while producing markedly different mistakes. The strengths of these three different models can be combined using an ensemble-method, producing even more accurate parses.

Not relying on solely on Treebank data The small size of the Hebrew Treebank may be adequate for learning a reasonable syntactic model, but the Treebank is far too small for learning an adequate lexical model. Making use of an external resource for providing lexical information is important both for dependency and constituency parsing.

For dependency parsing, a large amount of lexical knowledge is injected into the training process by training the tagger on a version of the Treebank in which each word is assigned an automatic POS-tag and several other morphological properties. The parser learns to rely on these automatically assigned POS-tags and morphological information (which are also available in prediction time) when deciding on the correct parse. The assignment of POS-tags is robust, and is performed using a tagger that was trained based on a wide-coverage lexicon-based morphological analyzer and large quantities of unannotated text. Training the parser on automatically tagged data is crucial for parser performance, as evidenced in Chapter 8.

For constituency parsing, I take a more direct approach: the lexical tagging proba-

bilities which are used by the parser are estimated based on the Treebank only for words occurring many times in the training portion of the Treebank. For words which do not occur many times (or do not occur at all) in the training set, the lexical probabilities are assigned using a mixture of the Treebank-based estimate (if available) and an estimate based on a wide-coverage morphological analyzer and a large amount of unannotated text. Using the morphological-analyzer's-based estimates substantially improves accuracy (Chapter 10).

Deferring ambiguity resolution The dependency parsing algorithm presented in Chapter 7 and the constituency parsing algorithm described in Chapter 10 are very different from each other. Aside from producing two different kinds of linguistic representations, they also differ in their overall approach: the dependency parsing algorithm is greedy and does not perform any search (but relies on an expressive feature set for each of its decisions), while the constituency parsing algorithm performs an exhaustive search (but is restricted to relying on a minimal feature set). However, within their respective modes of operation, the two systems share the same philosophy with regard to ambiguity resolution – they try not to resolve ambiguities until they absolutely must do so.

The EASYFIRST parser is a deterministic transition-based parser, that must perform an action at each stage of its operation and cannot backtrack, undo the consequences of previously taken action, or abstain from taking any action. As natural language is inherently ambiguous, some of the parser's actions must resolve an ambiguity, and then the parser will then be bound to its decision. However, the easy-first training objective attempts to defer taking hard ambiguity resolutions decisions as much as possible, by starting from the least ambiguous decisions and hoping that the resolution of the less ambiguous decisions will help in reducing the ambiguity of the more ambiguous ones. The training process is designed in such a way as to make the parser attempt and identify hard ambiguous decisions, and defer their resolutions as much as possible. The result of this policy is a parser that performs as well as or better than even search-based dependency parsers on the UAS metric, and greatly outperforms them on the exact-match metric.

The PCFG-LA-based Hebrew parser, on the other hand, is search-based. It explicitly compares all the possible structures, and outputs the best one, thus performing its entire ambiguity resolution at once. The ambiguity resolution delaying strategy is apparent in the application of the parser to Hebrew: the lattice-parsing approach defers the resolution of the ambiguous token segmentation. The stream of words that should be parsed is ambiguous, and depends on the token segmentation. Instead of committing to one segmentation prior to parsing as is done in the pipeline approach, the lattice-based

encoding allows passing the segmentation ambiguity as input to the parser, and the lattice-parsing mechanism solves both the segmentation and the syntactic resolutions jointly.

When extending the lexical probability model of the constituency parser with information from the external lexicon, there is uncertainty, both at training and inference times, with respect to the correct lexical tag attached to each word in the present or predicted parse trees. As the choice regarding this particular ambiguity will not be visible in the final parse tree (the parser's output is a tree with the Treebank tags, and does not contain the external-lexicon tags), the parsing process chooses not to resolve this ambiguity at all. Instead, the parser regards all the allowed external tags as possible, and its final prediction marginalizes them out, *i.e.*, it is based on the assumption that all the possibilities are correct.

Morphology and parsing One hypothesis that motivated the work on this thesis was that morphological agreement information can provide hints which are useful for parse disambiguation, and that such information should be incorporated in the parsing process and help in improving the parsing accuracy.

This turned out to be only partially true. While the use of morphological agreement information did improve parsing accuracy, the gains were minimal. The main reason for the small increase is that the parsers were good enough to not make many agreement mistakes to begin with. While there are clear cases where morphological information could be used to resolve ambiguities, it turns out that most of these cases could be resolved also based on other structural factors, which to a large extent remain a mystery for me, but which were nevertheless identified and utilized by the learning components of the various parsing algorithms. While agreement information can be used to resolve some attachment ambiguities, it is also highly redundant. Overall, it seems that agreement information helps transition-based systems a bit more than it helps search-based systems. A possible explanation for this is that transition-based systems learn to rely on agreement information to guide some local decisions, while the search-based systems are capable of utilize more global structural constraints that make the agreement-supplied hints more redundant.

One area where morphological agreement is crucial is in machine-translation, and, more broadly in natural language generation. When attempting to generate new sentences, we must pay attention to morphological agreement constraints. While agreement constraints are hard to encode effectively in a PCFG grammar, the approach presented in Chapter 10, of modeling them as manually defined filters which are part of the decoding process and orthogonal to the grammar, is a natural fit for target-side syntax-

based machine translation.

11.1 Open Issues

The work described in this thesis highlighted the need for further work in the following areas.

More use of external resources Using lexical data external to the Treebank substantially improves parsing accuracy, but the final parsing accuracies can still be improved. The methods presented in this thesis only scratch the surface in using external lexical resources to improve parsing, and the effective representation and use of lexical knowledge for resolving syntactic ambiguities is still an open research question. Lexical resources such as verb valency and subcategorization lexicons, selectional-preferences lists, dictionaries of multi-word expressions, and semantic ontologies can provide valuable information for disambiguating syntactic ambiguities. Such lexical resources can be either created manually or, as demonstrated in recent research [108, 119, 133], acquired from large quantities of automatically parsed text using machine-learning clustering methods. Some preliminary reports show that statistics from large quantities of automatically annotated text can indeed improve parsing accuracies [12, 28, 80, 148, 155], but the methods impose a heavy performance penalty, while the actual improvement obtained remains small. A good method for incorporating Treebank-external information in order to improve parsing is greatly needed.

Better handling of token segmentation Ambiguous word-segmentation remains a serious source of errors for both dependency and constituency Hebrew parsers. The syntactic structure of the sentence and the correct word-segmentation are closely linked with each other, and segmentation mistakes are detrimental to syntactic parsing accuracy. How can word segmentation be improved? Further work is needed to identify and quantify the aspects of the syntactic structure that are most affected by incorrect word segmentation, and the kinds of word segmentation that can be best disambiguated by taking possible syntactic structures into account. An additional area for future investigation is the joint prediction of syntactic structure and word segmentation. The lattice-parsing mechanism described in Chapter 10 is more effective than the pipeline model, but its use restricts both the segmentation and the syntactic modules to use relatively small amount of contextual information. It is desired to have joint-prediction methods which are more expressive in terms of the kinds of information they allow to incorporate into the predictions of both components, and which are efficient enough to

be incorporated into fast discriminative dependency parsers such as those described in Chapters 6 and 8. A promising direction into these kinds of models is the incorporation of the dual-decomposition technique recently popularized in [82, 120] with a dynamic-programming dependency parsing technique such as the one presented in [66].

Confidence-estimates on the produced structures English parsers provide accuracies of about 90% UAS, while Hebrew parsers provide accuracies of about 85% UAS. These are high numbers, yet they still mean that the parsers are likely to provide an incorrect analysis for at least one out of ten words. Moreover, the mistakes are not uniform across sentences: a parser may assign a correct structure to all the words in a given sentence, yet fail miserably on others. Most research effort into parsing is directed at improving these accuracy numbers. Very little research is directed at trying to *predict* these accuracy numbers. Currently, parsers are equally confident about all of their predictions. There is no way for a parser to distinguish between words it is likely to get correct and words it is likely to get wrong. Similarly, a parser can not identify sentences it is likely to err on. It is critical to incorporate various notions of *confidence* into parsing algorithms. Such confidence may come in the form of a numeric score attached to complete analyses, or by allowing the parser to output only partial (yet reliable) structures. Recent results from psycholinguistics indicate that humans do not form complete analyses of every sentence they process, but instead settle for a more coarse-grained representation, which they expand when needed [46]. I would like to investigate ways in which automatic syntactic parsers can perform in a similar fashion. The EASYFIRST parsing algorithm may be a good first-step in this direction.

Using syntax Finally, the newly available syntactic analysis tools should be put to use in other, higher-level tasks such as semantic analysis of Hebrew text, and syntax-based machine translation *into* Hebrew and similar languages.

Bibliography

[1] Anne Abeillé, Lionel Clément, and François Toussenel. Building a Treebank for French. In *Treebanks : Building and Using Parsed Corpora*. Springer, 2003.

[2] Steven Abney. Parsing by Chunks. *Principle-Based Parsing: Computation and Psycholinguistics*, 1991.

[3] Meni Adler. Hidden Markov Model for Hebrew Part-of-Speech Tagging. Master's thesis, Ben-Gurion University of the Negev, 2001.

[4] Meni Adler. *Hebrew Morphological Disambiguation: An Unsupervised Stochastic Word-based Approach*. PhD thesis, Ben-Gurion University of the Negev, 2007.

[5] Meni Adler and Michael Elhadad. An unsupervised morpheme-based HMM for Hebrew morphological disambiguation. In *Proc. of COLING/ACL*. Association for Computational Linguistics, 2006.

[6] Meni Adler, Yoav Goldberg, David Gabay, and Michael Elhadad. Unsupervised lexicon-based resolution of unknown words for full morphological analysis. In *Proc. of ACL*, 2008.

[7] Meni Adler, Yael Netzer, David Gabay, Yoav Goldberg, and Michael Elhadad. Tagging a Hebrew corpus: The case of participles. In *Proc. of LREC 2008*, 2008.

[8] Bharat Ram Ambati, Samar Husain, Sambhav Jain, Dipti Misra Sharma, and Rajeev Sangal. Two methods to incorporate 'local morphosyntactic' features in Hindi dependency parsing. In *Proc. of the NAACL/HLT Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010)*, 2010.

[9] Bharat Ram Ambati, Samar Husain, Rajeev Sangal, and Joakim Nivre. On the role of morphosyntactic features in Hindi dependency parsing. In *Proc. of the NAACL/HLT Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010)*, 2010.

[10] Giuseppe Attardi and Felice Dell'Orletta. Reverse revision and linear tree combination for dependency parsing. In *Proc. of HLT/NAACL*. Association for Computational Linguistics, June 2009.

[11] Mohammed Attia, Jennifer Foster, Deirdre Hogan, Joseph Le Roux, Lamia Tounsi, and Josef van Genabith. Handling unknown words in statistical latent-variable parsing models for Arabic, English and French. In *Proc. of the NAACL/HLT Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010)*, 2010.

[12] Mohit Bansal and Dan Klein. Web-scale features for full-scale parsing. In *Proc. of ACL*, 2011.

[13] Roy Bar-Haim, Khalil Sima'an, and Yoad Winter. Choosing an optimal architecture for segmentation and POS-tagging of Modern Hebrew. In *Proc. of the ACL Workshop on Computational Approaches to Semitic Languages*. Association for Computational Linguistics, June 2005.

[14] Roy Bar-Haim, Khalil Sima'an, and Yoad Winter. Part-of-speech tagging of modern Hebrew text. *Natural Language Engineering*, 14(2), 2008.

[15] Regina Barzilay, Kathleen R. McKeown, and Michael Elhadad. Information fusion in the context of multi-document summarization. In *Proc. of ACL*, 1999.

[16] Kepa Bengoetxea and Koldo Gojenola. Application of different techniques to dependency parsing of Basque. In *Proc. of the NAACL/HLT Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010)*, 2010.

[17] Daniel M. Bikel. Design of a multi-lingual, parallel-processing statistical parsing engine. In *Proc. of HLT*, 2002.

[18] Eric Brill and Grace Ngai. Man vs. Machine: a case study in base noun phrase learning. In *Proc. of ACL*, 1999.

[19] Sabine Buchholz and Erwin Marsi. CoNLL-X shared task on multilingual dependency parsing. In *Proc. of CoNLL*, 2006.

[20] Shu Cai, David Chiang, and Yoav Goldberg. Language-independent parsing with empty elements. In *Proc. of ACL (short-paper)*, 2011.

[21] Marie Candito and Benoit Crabbé. Improving generative statistical parsing with semi-supervised word clustering. In *Proc. of IWPT'09*, 2009.

[22] Marie Candito, Benoit Crabbé, and Djamé Seddah. On statistical parsing of French with supervised and semi-supervised strategies. In *EACL 2009 Workshop Grammatical inference for Computational Linguistics*, 2009.

[23] Xavier Carreras. Experiments with a higher-order projective dependency parser. In *Proc. of CoNLL Shared Task, EMNLP-CoNLL*, 2007.

[24] John Carroll, Guido Minnen, Yvonne Canning, Siobhan Devlin, and John Tait. Practical simplification of english newspaper text to assist aphasic readers. In *In Proc. of AAAI-98 Workshop on Integrating Artificial Intelligence and Assistive Technology*, 1998.

[25] J. Chappelier, M. Rajman, R. Aragues, and A. Rozenknop. Lattice Parsing for Speech Recognition. In *In Sixth Conference sur le Traitement Automatique du Langage Naturel (TANL'99)*, 1999.

[26] Eugene Charniak. A maximum-entropy-inspired parser. In *Proc. of the 1st Annual Meeting of the North American Chapter of the ACL (NAACL)*, 2000.

[27] Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proc. of ACL*, 2005.

[28] Wenliang Chen, Jun'ichi Kazama, Kiyotaka Uchimoto, and Kentaro Torisawa. Improving dependency parsing with subtrees from auto-parsed data. In *Proc. of EMNLP*, 2009.

[29] Y. J. Chu and T. H. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14, 1965.

[30] J. Cocke and J. T. Schwartz. Programming languages and their compilers: Preliminary notes. Technical report, Courant Institute of Mathematical Sciences, New York University, 1970.

[31] Shay B. Cohen and Noah A. Smith. Joint morphological and syntactic disambiguation. In *Proc. of EMNLP-CoNLL*, 2007.

[32] Michael Collins. Three Generative, Lexicalized Models for Statistical Parsing. In *Proc. of ACL*, 1997.

[33] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

[34] Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP*, 2002.

[35] Michael Collins, Jan Hajič, Lance Ramshaw, and Christoph Tillmann. A statistical parser for Czech. In *Proc. of ACL*, 1999.

[36] Anna Corazza, Alberto Lavelli, Giogio Satta, and Roberto Zanoli. Analyzing an Italian Treebank with state-of-the-art statistical parsers. In *Proc. of the TLT*, 2004.

[37] Benoit Crabbé and Marie Candito. Expériences d'analyse syntaxique statistique du français. In *Actes de la 15ème Conférence sur le Traitement Automatique des Langues Naturelles (TALN'08)*, 2008.

[38] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8, 1960.

[39] Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. Generating typed dependency parses from phrase structure parses. In *Proc. of LREC*, 2006.

[40] Markus Dreyer, David A. Smith, and Noah A. Smith. Vine parsing and minimum risk reranking for speed and precision. In *Proc. of CoNLL*, 2006.

[41] Amit Dubey. What to do when lexicalization fails: parsing German with suffix analysis and smoothing. In *Proc. of ACL*, 2005.

[42] Amit Dubey and Frank Keller. Probabilistic parsing for German using sister-head dependencies. In *In Proc. of ACL*, 2003.

[43] Jack R. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B, 1967.

[44] Jason Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proc. of COLING*, 1996.

[45] Jason Eisner and Noah A. Smith. Parsing with soft and hard constraints on dependency length. In *Proc. of IWPT*, 2005.

[46] Fernanda Ferreira and Nikole D. Patson. The good enough approach to language comprehension. *Language and Linguistics Compass*, 1, 2007.

[47] Gaifman, H. Dependency systems and phrase-structure systems. *Information and Control*, 8, 1965.

[48] Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. Scalable inference and training of context-rich syntactic translation models. In *Proc. COLING/ACL*, July 2006.

[49] Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. What's in a translation rule? In *Proc. of HLT-NAACL*, 2004.

[50] JP Gee and F. Grosjean. Performance structures: a psycholinguistic and linguistic appraisal. *Cognitive psychology(Print)*, 15(4), 1983.

[51] Lewis Glinert. *The Grammar of Modern Hebrew*. Cambridge University Press, 1989.

[52] Yoav Goldberg, Meni Adler, and Michael Elhadad. Noun phrase chunking in Hebrew: Influence of lexical and morphological features. In *Proc. of COLING/ACL*, 2006.

[53] Yoav Goldberg, Meni Adler, and Michael Elhadad. EM Can find pretty good HMM POS-Taggers (when given a good start). In *Proc. of ACL*, 2008.

[54] Yoav Goldberg and Michael Elhadad. Easy-first dependency parsing of Modern Hebrew. In *Proc. of the NAACL/HLT Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010)*, 2010.

[55] Yoav Goldberg and Michael Elhadad. An efficient algorithm for easy-first non-directional dependency parsing. In *Proc. of NAACL*, 2010.

[56] Yoav Goldberg and Michael Elhadad. Joint Hebrew segmentation and parsing using a PCFGLA lattice parser. In *Proc. of ACL (short-paper)*, 2011.

[57] Yoav Goldberg and Reut Tsarfaty. A single generative model for joint morphological segmentation and syntactic parsing. In *Proc. of ACL*, 2008.

[58] Yoav Golderg, Reut Tsarfaty, Meni Adler, and Michael Elhadad. Enhancing unlexicalized parsing performance using a wide coverage lexicon, fuzzy tag-set mapping, and EM-HMM-based lexical probabilities. In *Proc. of EACL*, 2009.

[59] Spence Green and Christopher Manning. Better Arabic parsing: Baselines, evaluations, and analysis. In *Proc. of COLING*, 2010.

[60] BGU Computational Linguistics Group. Hebrew morphological tagging guidelines. Technical report, Ben Gurion University of the Negev, 2008.

[61] Noemie Guthmann, Yuval Krymolowski, Adi Milea, and Yoad Winter. Automatic annotation of morpho-syntactic dependencies in a Modern Hebrew Treebank. In *Proc. of TLT*, 2009.

[62] Vasileios Hatzivassiloglou, Judith L. Klavans, Melissa L. Holcombe, Regina Barzilay, Min-Yen Kan, and Kathleen R. McKeown. Simfinder: A flexible clustering tool for summarization. In *Workshop on Automatic Summarization, NAACL*, 2001.

[63] Liang Huang, Wenbin Jiang, and Qun Liu. Bilingually-constrained (monolingual) shift-reduce parsing. In *Proc. of EMNLP*, 2009.

[64] Liang Huang, Kevin Knight, and Aravind Joshi. Statistical syntax-directed translation with extended domain of locality. In *Proc. of AMTA*, 2006.

[65] Liang Huang and Haitao Mi. Efficient Incremental Decoding for Tree-to-String Translation. In *Proc. of EMNLP*, 2010.

[66] Liang Huang and Kenji Sagae. Dynamic programming for linear-time incremental parsing. In *Proc. of ACL*, July 2010.

[67] Zhongqiang Huang and Mary Harper. Self-training PCFG grammars with latent annotations across languages. In *Proc. of the EMNLP*. Association for Computational Linguistics, 2009.

[68] Alon Itai and Shuly Wintner. Language resources for Hebrew. *Language Resources and Evaluation*, 42(1), March 2008.

[69] Wenbin Jiang, Liang Huang, and Qun Liu. Automatic adaptation of annotation standards: Chinese word segmentation and POS tagging – a case study. In *Proc. of ACL/AFNLP*, August 2009.

[70] Hongyan Jing. Sentence reduction for automatic text summarization. In *Proc. of ANLC*, 2000.

[71] Hongyan Jing and Kathleen R. McKeown. Cut and paste based text summarization. In *Proc. of NAACL*, 2000.

[72] Richard Johansson and Pierre Nugues. Extended constituent-to-dependency conversion for English. In *Proc. of NODALIDA*, 2007.

[73] Mark Johnson. PCFG models of linguistic tree representations. *Computational Linguistics*, 24, 1998.

[74] Mark Johnson. Transforming projective bilexical dependency grammars into efficiently-parsable CFGs with unfold-fold. In *Proc. of ACL*, June 2007.

[75] T. Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. *Scientific Report AFCRL-65-758, Air Force Cambridge Research Lab*, 10, 1965.

[76] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proc. of ACL*, July 2003.

[77] Kevin Knight and Daniel Marcu. Summarization beyond sentence extraction: a probabilistic approach to sentence compression. *Artificial Intelligence*, 139:91–107, July 2002.

[78] Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, 2010.

[79] Philipp Koehn, Franz Josef Och, and Daniel Marcu. Statistical Phrase-based Translation. In *Proc. of NAACL*, 2003.

[80] Terry Koo, Xavier Carreras, and Michael Collins. Simple semi-supervised dependency parsing. In *Proc. of ACL*, 2008.

[81] Terry Koo and Michael Collins. Efficient third-order dependency parsers. In *Proc. of ACL*, 2010.

[82] Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. Dual decomposition for parsing with non-projective head automata. In *Proc. of EMNLP*, 2010.

[83] Sandra Kübler, Ryan T. McDonald, and Joakim Nivre. *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2009.

[84] Marco Kuhlmann and Joakim Nivre. Mildly non-projective dependency structures. In *Proc. of COLING/ACL*. Association for Computational Linguistics, July 2006.

[85] John Lee, Jason Naradowsky, and David A. Smith. A discriminative model for joint morphological disambiguation and dependency parsing. In *Proc. of ACL/HLT*, 2011.

[86] Yang Liu, Qun Liu, , and Shouxun Lin. Tree-to-string alignment template for statistical machine translation. In *Proc. of COLING/ACL*, 2006.

[87] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[88] Andre Martins, Noah Smith, and Eric Xing. Concise integer linear programming formulations for dependency parsing. In *Proc. ACL/AFNLP*, 2009.

[89] Yuval Marton, Nizar Habash, and Owen Rambow. Improving Arabic dependency parsing with lexical and inflectional morphological features. In *Proc. of the NAACL/HLT Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010)*, 2010.

[90] Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Probabilistic CFG with latent annotations. In *Proc. of ACL*, 2005.

[91] Ryan Mcdonald. Discriminative sentence compression with soft syntactic constraints. In *Proc. of EACL*, 2006.

[92] Ryan McDonald. *Discriminative Training and Spanning Tree Algorithms for Dependency Parsing*. PhD thesis, University of Pennsylvania, 2006.

[93] Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *Proc. of ACL*, 2005.

[94] Ryan McDonald and Joakim Nivre. Characterizing the errors of data-driven dependency parsing models. In *Proc. of EMNLP*, 2007.

[95] Ryan McDonald and Fernando Pereira. Online learning of approximate dependency parsing algorithms. In *Proc. of EACL*, 2006.

[96] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. Non-projective dependency parsing using spanning tree algorithms. In *Proc. of EMNLP*, 2005.

[97] Ryan McDonald and Giorgio Satta. On the complexity of non-projective data-driven dependency parsing. In *Proc. of IWPT*, 2007.

[98] Mel'čuk, I. *Dependency Syntax: Theory and Practice*. State University of New York Press, 1988.

[99] Tetsuji Nakagawa. Multilingual dependency parsing using global features. In *Proc. of EMNLP-CoNLL*, 2007.

[100] Yael Netzer, Meni Adler, David Gabay, and Michael Elhadad. Can you tag the modal? you should! In *ACL07 Workshop on Computational Approaches to Semitic Languages*, 2007.

[101] Joakim Nivre. Incrementality in deterministic dependency parsing. In *Incremental Parsing: Bringing Engineering and Cognition Together, ACL-Workshop*, 2004.

[102] Joakim Nivre. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4), December 2008.

[103] Joakim Nivre, Johan Hall, Sandra Kübler, Ryan Mcdonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. The CoNLL 2007 shared task on dependency parsing. In *Proc. of EMNLP-CoNLL*, 2007.

[104] Joakim Nivre, Johan Hall, and Jens Nillson. MaltParser: A data-driven parser-generator for dependency parsing. In *Proc. of LREC*, 2006.

[105] Joakim Nivre and Ryan McDonald. Integrating graph-based and transition-based dependency parsers. In *Proc. of ACL*, 2008.

[106] Nivre J. Dependency grammar and dependency parsing. Technical Report MSI 05133, Vxj University: School of Mathematics and Systems Engineering, 2005.

[107] Adi Milea Noemie Guthmann, Yuval Krymolowski and Yoad Winter. Automatic annotation of morpho-syntactic dependencies in a Modern Hebrew Treebank. In *Proc. of TLT*, 2009.

[108] Diarmuid Ó Séaghdha. Latent variable models of selectional preference. In *Proc. of ACL*, 2010.

[109] Owen Rambow. The Simple Truth about Dependency and Phrase Structure Representations: An Opinion Piece. In *Proc. of NAACL*, 2010.

[110] Maria Teresa Pazienza, editor. *Information Extraction: Towards Scalable, Adaptable Systems*. Springer-Verlag, 1999.

[111] Slav Petrov. *Coarse-to-Fine Natural Language Processing*. PhD thesis, University of California at Bekeley, 2009.

[112] Slav Petrov. Products of random latent variable grammars. In *Proc. of NAACL*, 2010.

[113] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proc. of ACL*, 2006.

[114] Slav Petrov and Dan Klein. Improved inference for unlexicalized parsing. In *Proc. of NAACL*, 2007.

[115] Slav Petrov and Dan Klein. Parsing German with latent variable grammars. In *Proc. of the ACL Workshop on Parsing German*, 2008.

[116] Matt Post and Dainel Gildea. Weight pushing and binarization for fixed-grammar parsing. In *Proc. of IWPT*, 2009.

[117] Detlef Prescher. Inducing head-driven PCFGs with latent heads: Refining a tree-bank grammar for parsing. In *Proc. of ECML*, 2005.

[118] Sebastian Riedel and James Clarke. Incremental integer linear programming for non-projective dependency parsing. In *Proc. of EMNLP*, 2006.

[119] Alan Ritter, Mausam, and Oren Etzioni. A latent dirichlet allocation method for selectional preferences. In *Proc. of ACL*, 2010.

[120] Alexander M Rush, David Sontag, Michael Collins, and Tommi Jaakkola. On dual decomposition and linear programming relaxations for natural language processing. In *Proc. of EMNLP*, 2010.

[121] Kenji Sagae and Alon Lavie. A best-first probabilistic shift-reduce parser. In *Proc. of ACL*, 2006.

[122] Kenji Sagae and Alon Lavie. Parser combination by reparsing. In *Proc. of HLT-NAACL*, 2006.

[123] Federico Sangati and Chiara Mazza. An English dependency Treebank à la Tesnière. In *Proc. of TLT8*, 2009.

[124] Anoop Sarkar and Daniel Zeman. Automatic extraction of subcategorization frames for Czech. In *Proc. of CoLING*, 2000.

[125] Djamé Seddah, Marie Candito, and Benoit Crabbé. Cross parser evaluation and tagset variation: A French Treebank study. In *Proc. of IWPT*, 2009.

[126] Djamé Seddah, Grzegorz Chrupała, Ozlem Cetinoglu, Josef van Genabith, and Marie Candito. Lemmatization and statistical lexicalized parsing of morphologically-rich languages. In *Proc. of the NAACL/HLT Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010)*, 2010.

[127] Libin Shen and Aravind K. Joshi. LTAG dependency parsing with bidirectional incremental construction. In *Proc. of EMNLP*, 2008.

[128] Libin Shen, Giorgio Satta, and Aravind K. Joshi. Guided learning for bidirectional sequence classification. In *Proc. of ACL*, 2007.

[129] Khalil Sima'an, Alon Itai, Yoad Winter, Alon Altman, and Noa Nativ. Building a Tree-Bank of Modern Hebrew text. *Traitement Automatique des Langues*, 42(2), 2001.

[130] D. Sleator and D Temperley. Parsing English with a link grammar. In *Proc. of IWPT*, 1993.

[131] Noah A. Smith, David A. Smith, and Roy W. Tromble. Context-based morphological disambiguation with random fields. In *Proc. of EMNLP*, October 2005.

[132] Rion Snow, Dan Jurafsky, and Andrew Y. Ng. Learning syntactic patterns for automatic hypernym discovery. In *Proc. of NIPS*. MIT Press, 2005.

[133] Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. Semantic taxonomy induction from heterogenous evidence. In *Proc. of ACL*, 2006.

[134] Xinying Song, Shilin Ding, and Chin-Yew Lin. Better binarization for the CKY parsing. In *Proc. of EMNLP*, 2008.

[135] Mihai Surdeanu and Christopher D. Manning. Ensemble models for dependency parsing: Cheap and good? In *Proc. of NAACL*, 2010.

[136] Tesnière, L. *Èlèments de syntaxe structurale*. Klincksieck, 1959.

[137] Ivan Titov and James Henderson. Fast and robust multilingual dependency parsing with a generative latent variable model. In *Proc. of EMNLP-CoNLL*, 2007.

[138] André Filipe Torres Martins, Dipanjan Das, Noah A. Smith, and Eric P. Xing. Stacking dependency parsers. In *Proc. of EMNLP*, 2008.

[139] Stephen Tratz and Eduard Hovy. A fast, effective, non-projective, semantically-enriched parser. In *Proc. of EMNLP*, 2011.

[140] Reut Tsarfaty. Integrated Morphological and Syntactic Disambiguation for Modern Hebrew. In *Proc. of ACL-SRW*, 2006.

[141] Reut Tsarfaty. The Interplay of Syntax and Morphology in Building Parsing Models for Modern Hebrew. In *Proc. of ESSLI Student Session*, 2006.

[142] Reut Tsarfaty. *Relational-Realizational Parsing*. PhD thesis, ILLC Dissertation Series, University of Amsterdam, 2010.

[143] Reut Tsarfaty, Djame Seddah, Yoav Goldberg, Sandra Kubler, Marie Candito, Jennifer Foster, Yannick Versley, Ines Rehbein, and Lamia Tounsi. Statistical parsing of morphologically rich languages: What, how and whither. In *Proc. of NAACL Workshop on Statistical Parsing of Morphologically Rich Languages*, 2010.

[144] Reut Tsarfaty and Khalil Sima'an. Three-dimensional parametrization for parsing morphologically rich languages. In *Proc. of IWPT*, 2007.

[145] Reut Tsarfaty and Khalil Sima'an. Relational-realizational parsing. In *Proc. of CoLING*, 2008.

[146] Reut Tsarfaty and Khalil Sima'an. Modeling morphosyntactic agreement in constituency-based parsing of Modern Hebrew. In *Proc. of the NAACL/HLT Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2010)*, 2010.

[147] Reut Tsarfaty, Khalil Sima'an, and Remko Scha. An alternative to head-driven approaches for parsing a (relatively) free word order language. In *Proc. of EMNLP*, 2009.

[148] Gertjan van Noord. Using self-trained bilexical preferences to improve disambiguation accuracy. In *Proc. of IWPT*, 2007.

[149] Wei Wang, Jonathan May, Kevin Knight, and Daniel Marcu. Re-structuring, re-labeling, and re-aligning for syntax-based statistical machine translation. *Computational Linguistics*, 2010.

[150] Dekai Wu. Stochastic Inversion Transduction Grammars and Bilingual Parsing of Parallel Corpora. *Computational Linguistics*, 1997.

[151] Hiroyasu Yamada and Yuji Matsumoto. Statistical dependency analysis with support vector machines. In *Proc. of IWPT*, 2003.

[152] Roman Yangarber, Ralph Grishman, Pasi Tapanainen, and Silja Huttunen. Unsupervised discovery of scenario-level patterns for information extraction. In *Proc. of the sixth conference on Applied natural language processing*. Morgan Kaufmann Publishers Inc., 2000.

[153] D.H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10, 1967.

[154] Yue Zhang and Stephen Clark. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proc. of EMNLP*, 2008.

[155] Guangyou Zhou, Jun Zhao, Kang Liu, and Li Cai. Exploiting web-derived selectional preference to improve statistical dependency parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 1556–1565, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

תקציר

שפה מורכבת ממילים המצורפות זו לזו ליצירת משפטים. אף כי מילים בודדות מסוגלות לבטא מגוון רב של משמעויות, צירופן של המילים למשפטים הוא זה שמאפשר תקשורת יעילה והמשגה של רעיונות מורכבים. כשמילים מצורפות לכדי משפט, אופן הצירוף נשלט על ידי אוסף של חוקים, המהווים את התחביר (*syntax* או *grammar*) של השפה. על המשפטים לעמוד באילוצים מבניים המוגדרים על ידי התחביר, והמבנה של המשפט הוא הקובע את משמעותו. תזה זו עוסקת בניתוח תחבירי (*parsing*), כלומר קביעת המבנה התחבירי של משפטים בשפה. הגישה היא מונחית־נתונים – אלגוריתם למידה עובר תהליך של אימון בהתבסס על רשימה של משפטים והניתוח התחבירי הידוע שלהם (בנק עצים, או *Treebank*), ותוצאתו של תהליך האימון היא מנתח תחבירי (פרסר, *parser*) המסוגל לקבוע מבנה תחבירי נכון למשפטים חדשים, שלאלגוריתם הלומד לא היתה גישה אליהם.

בעבודה זו, אני מתמקד בניתוח תחבירי אוטומטי של עברית מודרנית, שפה שמית בעלת מורפולוגיה עשירה ויצרנית, סדר מילים חופשי יחסית ובנק־עצים קטן. המטרה המרכזית של העבודה הינה יצירה של מערכת חישובית המסוגלת לבצע ניתוח תחבירי אוטומטי של טקסטים בעברית מודרנית.

חלק נכבד מהספרות העדכנית בנושא ניתוח תחבירי אוטומטי מוקדש לניתוח תחבירי של השפה האנגלית, שהינה בעלת מורפולוגיה פשוטה יחסית, סדר מילים קבוע למדי, וכן קיים עבורה בנק־עצים גדול. מערכות ניתוח תחבירי מונחה־נתונים בשפה האנגלית מסוגלות לנתח טקסט חדשותי בדיוק של כ־90% (על פי אמת המידה המקובלת להערכת ניתוח תחבירי). ברם, במעבר לשפות בעלות מורפולוגיות עשירות יותר וסדרי מילים קבועים פחות, אלגוריתמי הניתוח שפותחו עבור השפה האנגלית מפגינים ירידה חדה ברמת הדיוק שלהם.

על מנת להתגבר על ירידה זו ברמת הדיוק, אני מתמקד במאפיינים שבהם עברית שונה מאנגלית. באנגלית, סדר המילים הוא קבוע יחסית, בעוד שבשפות אחרות סדר המילים הוא גמיש יותר (בעברית, לדוגמא, יכול נושא המשפט להופיע לפני הפועל או אחריו). בשפות בהן סדר המילים גמיש יותר, משמעות המשפט מובעת תוך שימוש באלמנטים מבניים אחרים, דוגמת הטיות של מילים או מיליות יעודיות (למשל המילית 'את' בעברית). אלמנטים אלו מכונים מורפולוגיה. היחידות הלקסיקליות (מילים) באנגלית תמיד מופרדות זו מזו על ידי רווח. בעברית (כמו גם בערבית), מרבית המילים מופרדות על ידי רווחים, אך מיליות יחס רבות לא מופיעות לבדן, כי אם נצמדות למילה הבאה בטקסט, ויוצרות יחידה אחת שניתן לעתים לקרוא באופנים שונים – כמילה בודדת או כרצף של שתי מילים (שמים, ש־מים).

עולות מספר שאלות: כיצד ניתן לפצות על גודלו הקטן של בנק העצים על ידי שימוש במשאבים קיימים ומקורות מידע אחרים? האם ניתן להשתמש במידע המורפולוגי כדי לשפר את הדיוק של הניתוח התחבירי? כיצד לטפל בנושא החלוקה למילים (המיליות הצמודות למילה שאחריהן ויוצרות בכך יחידה מרובת משמעויות)? בהתבסס על שאלות אלו, התזה נוגעת בנקודות הבאות:

הפרדה בין מידע לקסיקלי ומידע תחבירי: אני טוען כי כמות המידע התחבירי הדרוש למערכת תחבירית הינה קטנה דיה כדי להלמד באופן אפקטיבי גם מתוך בנק־עצים קטן יחסית. בניגוד לכך, המידע הלקסיקלי הדרוש הינו רחב היקף בהרבה, ואל לנו לשער כי בנק־עצים מנותח ידנית (גדול ככל שיהיה) יוכל לספק כיסוי לקסיקלי רחב מספיק. במקום זאת, עלינו לפעול למציאת דרכים לשילוב של מידע לקסיקלי חיצוני לבנק העצים אל תוך תהליך הניתוח התחבירי.

שימוש במידע מורפולוגי לשיפור הניתוח התחבירי: המורפולוגיה מספקת מידע רב שיכול לסייע להפגת רב־משמעות התחבירית ובחירה בניתוח הנכון. יש לקחת מידע זה בחשבון כאשר מתכננים מערכת ניתוח תחבירי אוטומטי לשפה מסויימת, ולתכנן את רכיב הניתוח התחבירי כך שיוכל לעשות שימוש במידע זה.

דחיית הטיפול באי־וודאות: לעיתים, הקלט למנתח התחבירי יכול להיות לא־ודאי. כאשר הדבר ניתן לביצוע מבחינה חישובית, ראוי לטפל באי־ודאות זו כחלק מתהליך הניתוח התחבירי ולא בתהליך יעודי נפרד המקדים את הניתוח התחבירי. באופן כללי יותר, עדיף לדחות את ההכרעה לגבי מקרים לא־ודאיים ככל הניתן, ולטפל בהחלטות הוודאיות יותר קודם להחלטות הוודאיות פחות.

התרומה המרכזית של עבודה זו הנה מערכות המסוגלות לספק ניתוח תחבירי של טקסט חופשי בשפה העברית המודרנית. בפרט, אני מציג מנתח תחבירי מסוג מנתח־פסוקיות (*constituency parser*) ומנתח תחבירי מסוג מנתח־תלויות (*dependency parser*). שתי המערכות זמינות כמערכות תוכנה בקוד פתוח, ומנתחות טקסט עברי ברמת דיוק של 77% (עבור מבנה פסוקיות) ו־ 79.5% (עבור מבנה תלויות) כשהקלט לניתוח הינו טקסט חופשי, או רמת דיוק של 85% (עבור שתי המערכות) כשהקלט לניתוח מכיל הפרדה נכונה של מיליות היחס מהמילים שאחריהן. תרומה נוספת של העבודה הנה יצירת מדריך לניתוח תחבירי מונחה תלויות של עברית, וכן בנק־עצים המכיל משפטים עבריים המנותחים במבני־תלויות, שנוצר על ידי המרה של בנק העצים המכיל עצים במבני פסוקיות.

מערכות הניתוח התחבירי לעברית שפותחו בעבודה זו נשענות על הפיתוחים האלגוריתמיים והמתודולוגיים הבאים:

עבור ניתוח למבנה תלויות, אני מציע את אלגוריתם EASYFIRST – אלגוריתם ניתוח תחבירי יעיל בעל סיבוכיות ריצה ($O(n \log n)$). זהו אלגוריתם חמדן הפועל בשיטת bottom up, ומבוסס על העקרון שהחלטות קלות צריכות להתבצע לפני החלטות קשות יותר. עבור השפה העברית, האלגוריתם משיג את תוצאות הניתוח המדוייקות ביותר. עבור השפה האנגלית, האלגוריתם משיג רמת דיוק השווה לזו של אלגוריתמים איטיים בהרבה, המתבססים על חיפוש מקיף. בנוסף, האלגוריתם מאפשר שימוש בסוגי מידע ואינדיקטורים רבים בביצוע כל החלטה, ואני משתמש ביכולת זו לקידוד מידע אודות התאמה־מורפולוגית במין ובמספר. עבודה זו היא בין הראשונות להראות כי מידע התאם־מורפולוגי מביא לשיפור ברמת הדיוק של הניתוח התחבירי.

עבור ניתוח למבנה פסוקית, אני מדגים כי ניתן לשפר את ביצועי המערכת על ידי שימוש במשאבי שפה החיצוניים לבנק העצים, ובפרט על ידי שימוש במנתח־מורפולוגי מבוסס לקסיקון. אני מציג מודל חישובי המאפשר קישור בין המנתח המורפולוגי החיצוני למנתח התחבירי המבוסס על בנק העצים, גם במקרה השכיח בו המידע בבנק העצים לא תואם מבחינת סכמת הניתוח שלו למידע במנתח המורפולוגי. כמו כן, אני מראה כי החלוקה למילים והניתוח התחבירי יכולות להתבצע בו־זמנית בתהליך יחיד על ידי שימוש בטכניקה של CKY lattice parsing. ביצוע שתי המשימות בו־זמנית הינו אפקטיבי, ומשיג תוצאות טובות במידה ניכרת משימוש בשני מודלים נפרדים. בנוסף, אני מציע למדל התאם תחבירי במין ובמספר במנתח תחבירי מבוסס פסוקיות כמנגנון סינון שאינו תלוי בחוקי הגזירה של המנתח התחבירי, ואני מציג מימוש של שיטה זו. המנתח התחבירי מלכתחילה לא מבצע טעויות התאם רבות, אך מנגנון הסינון הנו אפקטיבי בתיקון השגיאות המועטות שכן קיימות.

מילות מפתח: תחביר, מורפולוגיה, ניתוח תחבירי, מבנה־תלויות, מבנה־פסוקיות, עברית מודרנית, עיבוד טקסט.

למתעניינים
ולאלו שלא

העבודה נעשתה בהדרכת פרופ' מיכאל אלחדד

במחלקה למדעי המחשב
בפקולטה למדעי הטבע

ניתוח תחבירי אוטומטי
של טקסט בעברית מודרנית

מחקר לשם מילוי חלקי של הדרישות לקבלת תואר "דוקטור לפילוסופיה"

מאת

יואב גולדברג

הוגש לסינאט אוניברסיטת בן־גוריון בנגב

חתימת המחבר: _____

אישור המנחה: _____

אישור דיקן בית הספר ללימודי מחקר מתקדמים ע"ש קרייטמן : _____

תמוז תשע"א

באר שבע

ניתוח תחבירי אוטומטי
של טקסט בעברית מודרנית

מחקר לשם מילוי חלקי של הדרישות לקבלת תואר "דוקטור לפילוסופיה"

מאת

יואב גולדברג

הוגש לסינאט אוניברסיטת בן־גוריון בנגב

תמוז תשע"א

באר שבע