# A finite-state based morphological analyzer for Hebrew

Shlomo Yona

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE MASTER DEGREE

University of Haifa

Faculty of Social Science

Department of Computer Science

November, 2004

# A finite-state based morphological analyzer for Hebrew

By: Shlomo Yona

Supervised By: Dr. Shuly Wintner

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE MASTER DEGREE

University of Haifa

Faculty of Social Science

Department of Computer Science

November, 2004

Approved by: ________________________________ Date:________

(supervisor)

Approved by: ________________________________ Date:________

(Chairman of M.A Committee)

# Acknowledgment

I would like to thank the Ministry of Science and Technology for their financial support through the *Knowledge Center for Processing Hebrew*[1].

[1]http://mila.cs.technion.ac.il

# Contents

# A finite-state based morphological analyzer for Hebrew

Shlomo Yona

## Abstract

Morphological analysis is an important component in many natural language processing tasks. Existing morphological analyzers for Hebrew are either limited or proprietary. We developed a morphological analyzer for undotted Hebrew words that is based on finite-state linguistically motivated rules and a broad coverage lexicon. The lexicon contains base forms of words and linguistic attributes that are used by the rules to allow analysis and generation of Hebrew words. The current set of rules comprehensively covers the morphological phenomena that are observable in contemporary Hebrew texts. Our analyzer produces output for over 90% of the tokens observed in daily newspapers.

# List of Figures

# Chapter 1

# Introduction

## 1.1 Morphological analysis

Morphology studies the structure of words. Morphological analysis is the computational process which provides information about the structure of a given surface word. An analysis can produce morphological and morpho-syntactic features such as the root, tense, person, number etc. A morphological analyzer should separate and identify the component morphemes of the input word, labeling them with sufficient information to be useful further processing. Morphological generation is the inverse process of generating a surface word given a morphological analysis. A very simplified version of morphological analysis is called Stemming. Stemming is a process of obtaining a stem and affixes from a lexeme, where the stem is the objective of this operation. Its main use is as part of a term normalization process that is usually done when setting up Information Retrieval systems. This is an easier and simpler problem than morphological analysis, as the morphological properties of the stripped affixes are ignored. In many languages, in particular in Semitic languages, surface words have more than one morphological analysis. Morphological disambiguation is the process of choosing the correct morphological analysis of a surface word, taking its context into account.

Hebrew, like other Semitic languages, has rich and complex morphology. The major word formation machinery is root-and-pattern, where roots are sequences of three (typically) or more consonants, called *radicals*, and patterns are sequences of vowels and, sometimes, also consonants, with "slots" into which the root's consonants are being inserted. Words are created by *interdigitating* roots into patterns: the first radical is inserted into the first consonantal slot of the pattern, the second radical fills the second slot and the third fills the last slot. Inflectional morphology is highly productive and consists mostly of suffixes, but sometimes of prefixes or circumfixes. In general, inflectional morphology can be assumed to be concatenative, but derivational morphology is certainly non-concatenative.

As an example of root-and-pattern morphology, consider the Hebrew roots ג'ד'ל' and ר'ש'מ' and the patterns הCCCה and CיCוC, where the 'C's indicate the slots. When the roots combine with these patterns the resulting lexemes are הגדלה, גידול, הרשמה, רישום, respectively. After the root combines with the pattern, some morpho-phonological alternations take place, which may be non-trivial: for example, the התCCCות pattern triggers assimilation when the first consonant of the root is ת or ד: thus, ד'ר'ש'+התCCCות yields הדרשות. The same pattern triggers metathesis when the first radical is ס or ש: ס'ד'ר'+התCCCות yields הסתדרות rather than the expected התסדרות. Frequently, root consonants such as ו or י are altogether missing from the resulting form. Other *weak* paradigms include roots whose first radical is נ and roots whose second and third radicals are identical. Thus, the roots ק'ו'מ', ג'נ'נ', נ'פ'ל' and י'צ'ג', when combining with the הCCCה pattern, yield the seemingly similar lexemes הקמה, הגנה, הפלה and הצגה, respectively.

These matters are complicated further due to two sources: first, the standard Hebrew orthography leaves most of the vowels unspecified. It does not explicate *[a]* and *[e]* vowels, does not distinguish between *[o]* and *[u]* vowels and leaves many of the *[i]* vowels unspecified. Furthermore, the single letter ו is used both for the vowels *[o]* and *[u]* and for the consonant *[v]*, whereas י is similarly used both for the vowel *[i]* and for the consonant *[y]*. On top of that, the script dictates that many particles, including prepositions, conjunctions and the definite article, attach to the words which immediately follows them. Thus, a form such as מהגר can be read as a lexeme ("immigrant"), as מ+הגר "from Hagar"or even as מ+ה+גר "from the foreigner". Note that there is no deterministic way to tell whether the first מ of the form is part of the pattern, the root or a prefixing particle (the preposition מ "from").

An added complexity stems from the fact that there exist two main standards for the Hebrew script: one in which vocalization diacritics, known as *niqqud* (*dots*), decorate the words, and another in which the dots are missing, and other characters represent some, but not all of the vowels. Most of the texts in Hebrew are of the latter kind; unfortunately, different authors use different conventions for the undotted script. Thus, the same word can be written in more than one way, sometimes even within the same document. This fact adds significantly to the degree of ambiguity.

A lexeme can be inflected to and derived to many word forms. The lexeme שולחן (a table), for example, is the basis of the entire possessive inflection : שולחני (my table), שולחנך (your table, singular, masculine or feminine), שולחננו (our table), שולחנו (his table), שולחנה (her table), שולחנם (their table, masculine), שולחנן (their table, feminine), שולחנכם (your table, plural, masculine) and שולחנכן (your table, plural, feminine). The base form can alter in construct state (שולחן כתיבה - a desk). The morphological richness is even greater in the verb system.

In light of this morphological richness and the high degree of ambiguity, morphological analysis is a challenging problem. A naïve solution would be to list all word forms, including inflections, in the lexicon. This is not a practical task to perform manually due to the number of possible inflections of each word (dozens). It is also not a scalable solution since words that are missing from the lexicon obtain no analysis at all. On the other hand, formulating the morphological processes result in smaller lexica and might provide generalizations that allow (partial) analysis of unseen words.

## 1.2 Finite-state technology

Morphological rules need to be expressed, preferably in a manner which is both human and machine readable and maintainable. *Finite-state technology* offers the ability to describe natural language morphology in some calculus which can express most of the morphological processes and simulate to a satisfactory extent the remaining few (such as interdigitation and reduplications). Finite-state technology also offers a very compact and fast means of storing and using morphological rules.

One of the fundamental results of formal language theory (Kleene, 1956) is the demonstration that finite-state languages are precisely the set of languages that can be described by a *regular-expression.* A regular-expression denotes a set of strings (a language) or a set of string pairs (a relation). It can be compiled into a finite-state network that compactly encodes the corresponding language or relation that may well be infinite.

The language of regular-expressions includes the common set operators of boolean logic and operators that are specific to strings such as concatenation. It follows from Kleene's theorem that for each of the regular-expression operators for finite-state languages there is a corresponding operation that applies to finite-state networks and produces a network for the resulting language. Any regular-expression can in principle be compiled into a single network. A *finite-state network* is a term we use both for simple automata that define regular languages and for transducers (automata with output) which define regular relations. A finite-state network for a complex language can be built by first constructing a regular expression that describes the language in terms of set operations and then compiling that regular expression into a network. This is in general easier than constructing a complex network directly, and in fact it is the only practical way for all but the most trivial languages and relations. The advantages of using finite-state networks include time complexity that is linear in the length of a string that is processed, and the ability to *reverse* the morphological analyzer and obtain a morphological generator with no extra effort.

Several calculi provide more operators in addition to the three operators needed for formulating regular expressions. There are also computer programs which translate expressions in such calculi into networks. For example, *xfst* (Beesley and Karttunen, 2003) is a language consisting of many operators, which allow easy and natural description of most of the morphological processes of natural languages. Xfst is built on top of a software library that provides algorithms for creating automata from regular expressions and contains both classical operations such as union or composition and also new algorithms such as replacement or local sequentialization.

Closure properties of regular languages, the ability to lookup an analysis or generate a surface word in linear time in the length of the string, the compact storage of information and the natural way morphological rules can be expressed using regular-expressions make finite-state technology particularly appealing for NLP applications.

## 1.3 An introduction to xfst

*Xfst* is a programming language for regular expressions, which can be compiled into finite-state networks. The language of regular expressions is a formal language, similar to formulas of Boolean logic. It has a simple syntax but the expressions can be arbitrarily complex. Like formulas of Boolean logic, regular expressions denote sets.

The simplest kind of regular expression consists of a single symbol. For example, the expression `a` denotes the set that contains the string "a" and nothing else. A special kind of string, the empty string, is expressed with an empty pair of double quotes: "". A set of strings is called a *language*. The expression above, `a`, denotes the language consisting of just the string "a". As in set theory, a set consisting of pairs of strings is called a *relation*. The terms *regular language* and *regular relation* refer to sets that can be described by a regular expression.

The simplest kind of regular expression that denotes a relation is a *symbol pair*, such as `a:b`. It denotes the set consisting of the ordered pair ⟨"a", "b"⟩. A regular relation may always be viewed as a mapping between two regular languages. In this case, the relation is simply the crossproduct of the languages denoted by the expressions `a` and `b`.

In order to distinguish the two languages that are involved in a regular relation, we call the first one the *upper* and the second one the *lower* language of the relation. Correspondingly, in the pair `a:b`, the first symbol, `a`, is called the upper symbol and the second one, `b`, the lower symbol.

Complex regular expressions can be built up from simpler ones by means of regular expression *operators*. A regular expression can be enclosed in square brackets without changing its interpretation. For example, `[a | b] and [a] | [b]` describe the same language as `a | b`.

An empty pair of brackets, `[]`, is interpreted as denoting the empty-string language. The *universal language* contains all strings of any length including the empty string. `0` is a symbol that represents the empty string “”. `?` is a symbol that represents any symbol that occurs in the same regular expression and any unknown symbol. The 0 symbol is also used as the upper or the lower symbol of a symbol pair. The expression `0:a` maps ”” to “a”, and vice versa. The inverse relation is denoted by `a:0`.

The escape character, `%`, eliminates any special meaning of the following character. Thus `|`, `0`, `?`, etc. can be used as ordinary symbols by prefixing them with `%`. For example, `%0` represents the string “0” rather than the epsilon symbol; `%|` is the vertical bar itself, as opposed to the the union operator `|`. The ordinary percent sign may be expressed as `%%`.

The print name of a symbol can contain more than one character. For example, `%+V` represents the string “+V”. Sometimes it is easier to write characters of words together; for notational convenience, the `{text}` single character brace construct will take each character, between the braces, as a single-character symbol. Thus, `{abc}` is equivalent to `[a b c]`. Because xfst intentionally ignores the distinction between a language and its identity relation, a pair of identical symbols may be collapsed into a single symbol. Instead of `a:a` we may simply write `a`.

In addition to concatenation, union, and minus, xfst includes many other regular expression operators. The list given here is representative but not exhaustive (see Beesley and Karttunen (2003), chapters 2 and 3 for more details). Some of the operators below can be combined with all expressions regardless of whether they denote languages or relations. Concatenation and union are of this kind. Other operators presuppose that the component expression is of a particular type. For example, complementation, intersection, and minus can be used only with expressions that denote a simple language because regular relations are not closed with respect to these operations. Composition presupposes that the component expressions denote relations. For each of the operators described below, we specify what kind of expressions it operates on.

**~A** complement. The set of all strings that are not in the language `A`. For example, `~a` contains "", "aa", and "aaa" but not "a". The component expression, `A`, cannot denote a relation. The complement of the null language, `~\?`, is the universal language, and vice versa.

**A+** iteration (Kleene plus). The language or relation `A` concatenated with itself any number of times, including zero times. `A+` includes `[A]`, `[A A]`, `[A A A]`, and so on ad infinitum. `?+` is the language of all nonempty strings.

**A*** Kleene star. The union of `A+` with the empty-string language. `A*` is equivalent to `(A+)`. `?*` denotes the universal language.

**$A** containment. The set of strings or string pairs containing at least one instance of `A` as a subpart. For example, "bac" is a string in the language denoted by `$a`, speaking informally, "bac" is in `$a`. Here `A` can denote a relation. `$A` is equivalent to `[?* A ?*]`. `$[]` denotes the universal language.

**A B** concatenation of the languages or relations `A` and `B`.

**A |B** union of the languages or relations `A` and `B`. The set of strings or pairs of strings that are in `A` or in `B` or in both.

**A .P. B** upper side priority union. The result of `[A .P. B]` is a union of `A` and `B`, except that whenever `A` and `B` have the same string on the upper side, the path in `A` has precedence over the path in `B`. It is interesting to note that `.P.` performs poorly when executed on large networks. The effect of `.P.` can be obtained using other operations. The `[replace .P. overGen]` expression is implemented in xfst using:

```
[[overGen.u & ~replace.u] .o. overGen] | replace].
```

Unsurprisingly, using this alternative performs as poorly as the `.P.` operator itself. However, the equivalent

```
[[overGen.u - replace.u] .o. overGen] | replace]
```

performs efficiently. We therefore use the latter for priority union.

**A & B** intersection of the languages `A` and `B`. The set of strings that are both in `A` and in `B`. `A` and `B` must be simple languages, not relations. For example `[a | b] & [b | c]` consists of the string "b".

**A - B** `A` minus `B`. The set of strings that are in the language `A` but not in the language `B`. `A` and `B` must be simple languages, not relations. `[A - B]` is equivalent to `[A & ~B]`. Note also that `\a` could be defined as to `[? - a]`.

**A .x. B** crossproduct of the languages `A` and `B`. It denotes a regular relation that maps every string in `A` (upper side) to all strings in `B` (lower side), and vice versa. Thus `[a .x. [b | c]]` is equivalent to `[a:b | a:c]`. `A` and `B` must be simple languages, not relations.

**A .o. B** composition of the relations `A` and `B`. The intersection of the lower language of `A` and the upper language of `B` works like a filter here. For example, the expression `[a:b | b:c] .o. [a:x | b:y]` is equivalent to `[a:y]`. The `b:c` and `a:x` pairs do not contribute anything to the result because neither "c" nor "a" is in the intersection of `[b | c]` (the lower language of `A`) and `[a | b]` (the upper language of `B`).

**A ->B** replacement of the language `A` by the language `B`. This denotes a relation that consists of pairs of strings that are identical except that every instance of `A` in the upper-side string corresponds to an instance of `B` in the lower-side string. For example, `[a -> b]` pairs "b" with "b" (no change) and "aba" with "bbb" (replacing both "a"s by "b"s). Replacement may be constrained by a specific context. For example, `[A -> B || L _ R]` makes the `A -> B` replacement only if the `A` string is preceded by a string from `L` and followed by a string from `R`.

**A ->B ||L _ R** conditional replacement of the language `A` by the language `B`. This is like the relation `[A -> B]` except that any instance of `A` in the upper string corresponds to an instance of `B` in the lower string only when it is preceded by an instance of `L` and followed by an instance of `R`. Other instances of `A` in the upper string remain unchanged. `L` is defined as *left context* and `R` as *right context*. `A`, `B`, `L`, and `R` must all denote simple languages, not relations. The `||` indicates that both contexts refer to the upper string.

**.#.** boundary marker. In the conditional replacement a special boundary marker, `.#.` can be used to refer to the beginning or to the end of a string. In the left context, the boundary marker signals the beginning of the string; in the right context it means the end of the string.

**define VARIABLE regularExpression;** the `define` command is used to assign the finite-state network that is represented by `regularExpression` to the variable `VARIABLE`. We use this command to separately compile finite-state networks for parts of the morphological analyzer.

## 1.4 Computational approaches to Hebrew morphology

While the original Two-Level formulation (Koskenniemi, 1983) of finite-state technology for morphology was not particularly well suited to Semitic languages (Lavie et al., 1988; Lavie, 1989), modifications of the Two-Level paradigm and more advanced finite-state implementations have been applied successfully to a variety of Semitic languages, including Ancient Akkadian (Kataja and Koskenniemi, 1988), Syriac (Kiraz, 2000) and Arabic. Beesley in a number of works (Beesley, 1996; Beesley, 1998a; Beesley, 1998b; Beesley, 2001; Beesley and Karttunen, 2000) describes a finite-state morphological analyzer of modern standard Arabic that can be tested on the Internet. The morphological knowledge was assembled by linguists manually over a long period of time. This Arabic system handles both inflectional and derivational morphology, including interdigitation processes typical to Semitic languages.

Work on computational Hebrew morphology can be roughly divided into two categories: morphological analysis and morphological disambiguation. A comprehensive and thorough survey of computational linguistics for Hebrew was done by Wintner (2004), on which the following discussion is based. We concentrate on morphological analysis and do not discuss disambiguation below.

The first computational system for Hebrew morphological analysis was probably built by Choueka in the early sixties (Shapira and Choueka, 1964). Naturally, this system is very basic, yet its accomplishments are impressive: the article describes both a full morphological analysis of the noun system and an implementation of the analysis for preparing a concordance and a citation list. In a subsequent article Choueka (1966) describes the part handling verbs.

This work was an introduction to a large scale project dealing with various aspects of computational linguistics, natural language processing and information retrieval. Algorithms for morphological analysis were further developed (Attar et al., 1978) in order to support such projects. Algorithms were developed for automatic generation of all possible inflections and derivations of every Hebrew lexical entry, including those generated by adding morphemes such as prepositions, conjunctions and so on. A file containing all possible Hebrew words was built, which was used for the generation and analysis morphological systems for Hebrew (but not modern Hebrew).

Later, Choueka developed for the third time a Hebrew morphological analysis system, which was a part of the *RAV-MILIM* project (Choueka, 1990). This system was configured for modern Hebrew, and was used in two main applications: a vocalization software (which adds the missing vowels from the words and practically disambiguates the meaning) and a practical dictionary which also contains morphological analysis. These applications soon turned commercial, and therefore, the system, probably the most extensive and comprehensive system yet for Hebrew, was never made available for computational linguistics researchers. This work was briefly surveyed in (Choueka, 1993).

A commercial system, AVGAD, was developed at IBM Haifa Research (Bentur et al., 1992; Bentur, Angel, and Segev, 1992). It was based on a 25000 entry dictionary, from which "hundreds of thousands" of Hebrew words can be recognized. The system was integrated into word processors mainly used for spelling correction. AVGAD is an outdated system, with an incomplete dictionary. The system cannot be further developed as it is commercial and unavailable to the research community.

A morphological analyzer built by Segal (1997) produces all the possible analyses for a given lexeme. Although this system is available to the research community, its dictionary is limited (based on AVGAD with a few words added) and the morphological rules are an inseparable part of the computer program, which makes them very hard to maintain. The analyses are sometimes superficial or incorrect due to inability to fully handle prefix particles and a guesser which picks the Proper Name category for all unfamiliar surface forms.

A different approach to Hebrew morphology was introduced by Ornan. Based on his observations of the difficulties and limitations posed on computational processing by the Hebrew orthography, Ornan suggested an alternative script for the language, based on the Latin alphabet plus several symbols which appear on a computer keyboard: this is the *phonemic script* (Ornan, 1986; Ornan, 1994). The fact that the vast majority of Hebrew texts available in electronic format is not represented in phonemic script practically rules out any usage of it. There are, however systems, such as Ornan and Katz (1995) which convert Hebrew script to the phonemic script, but they require morphological disambiguation and the degree of accuracy of this conversion is not mentioned.

Another approach which generates all possible inflections of lexicon items is used by the HSpell project.[1] The goal of this project is to offer a practical spell checker for Hebrew. The main effort of the project is the construction and expansion of a broad coverage lexicon of verbs, nouns and adjectives, which is then automatically inflated into a TRIE containing all possible inflections of the lexicon items. HSpell also offers a basic morphological analyzer which returns some basic linguistic information for a lexicon item. HSpell contains a grammar describing many possible ways of attachment of prefix particles to verbs, nouns and adjectives.

## 1.5 Research goals

Many natural language processing tasks, such as machine translation, require some processing of words in order to obtain their base forms, morphotactics, part-of-speech and other linguistic information. Currently available systems for morphological analysis of Hebrew suffer from

[1]http://www.ivrix.org.il/projects/spell-checker/

one or more of the following problems:

1. Morphological rules are, for the most part, expressed in terms easily manipulated by the programmers and hence become part of the system. Separation of the morphological rules (including morphotactics and morphophonology) from the morphological processing program (analysis and generation) is important for practical and theoretical reasons. First, formalization of the rules is of theoretical linguistic value as it contributes to a better understanding of the morphological, morphophonological and orthographical processes which occur in a natural language. Secondly, a computational system which separates linguistic rules from the processing program based on the rules will be more modular, and therefore easier to maintain, update, and expand.

2. Most available systems are capable of analysis only. The reversibility property of finite-state transducers is extremely useful for implementation of a morphological system, as it is capable of doing both morphological analysis and morphological generation.

3. As described in Wintner (2004), existing morphological analyzers for Hebrew are either limited or proprietary.

Our goal is to develop a system that performs morphological analysis (and generation) of undotted Modern Hebrew, with broad coverage and based on finite-state linguistically motivated rules. Our system includes a broad coverage modular lexicon represented in XML, which is compiled into a finite-state network. As far as we know, no morphological analyzer for Hebrew is based on finite-state technology.

The design of the morphological analyzer is divided into two main components: a lexicon and a set of rules. Both parts are implemented in xfst. The lexicon lists base forms (lexemes/stems), with some additional linguistic information. The rules implement inflectional morphology, morphological alternations and orthographic issues. The rules and the lexicon are compiled together into the morphological system, which is represented as a finite-state network. An overview of the system is presented in chapter 2, whereas chapters 3 and 4 detail the lexicon and the rules, respectively.

# Chapter 2

# Overview of the system

The system consists of two main components: a *lexicon* represented in XML, and a set of finite-state rules, implemented in xfst. We discuss the structure of the lexicon in section 2.1 and its conversion to xfst in section 2.2. The structure of the rules is explained in section 2.3.

## 2.1 Lexicon structure

This lexicon is currently used as a tool for morphological analysis software, and can interface with other natural language processing tools for Hebrew. The lexicon is represented in Extensible Markup Language[1] (XML), which is a set of language rules that have been defined to describe and structure data. The usage of XML supports standardization, allows a format that is both human and machine readable, and supports interoperability with other applications.

The lexicon contains a *metadata* section and an *items* section. The metadata section contains information about the lexicon (such as: name, version, revision, contact information of maintainers), its sources (name of the source, and a link to it), and other administrative information such as licenses and copyrights. In addition, it defines a transliteration that maps every Hebrew letter to a Latin equivalent. The items section is a list of lexical entries, each with a base form that is the form used in conventional dictionaries. The base form of

[1]http://www.w3.org/XML/

nouns and adjectives is the absolute singular masculine, and for verbs it is the third person singular masculine, past tense.

All lexicon items are specified for the following attributes:

**id** a string which uniquely identifies the item.

**undotted** the undotted Hebrew script spelling of the item (according to the formal strict recommendations of the Academy of the Hebrew Language (Cohen, 1996), unless otherwise stated in the *script* attribute).

**transliterated** a Latin transliteration of the undotted form.

**dotted** the dotted Hebrew representation of the base form.

**script** *formal*, *colloquial*, *typo* or *slang*. The default is *formal*.

In addition, every lexicon item belongs to a *part of speech* category. Eleven parts of speech are defined: *adjective*, *adverb*, *conjunction*, *interjection*, *negation*, *noun*, *preposition*, *pronoun*, *properName*, *quantifier* and *verb*. The part of speech of a lexicon item determines its additional attributes (discussed in detail in chapter 3). Note that some lexicon attributes are not used by the morphological analyzer, but are meaningful for other applications. Furthermore, lexicon attributes are of two types:

**lexical attributes** attributes that represent lexical features such as *gender* and *number*

**support attributes** attributes that represent information used for derivation and inflection processes such as *plural* (the suffix to use for the plural form of nouns and adjectives) and *dual* (can the dual form be derived from the base form, in nouns).

Figure 2.1 exemplifies a lexicon item represented in XML. Note that there are no additional attributes specified for this item.

Irregular forms, slang and colloquial spellings can be specified in the lexicon using three XML elements: *add*, *remove* and *replace*, which are listed along with the part of speech elements of lexicon items.

```
<item id="4807" transliterated="bli" undotted="בלי">
 <negation />
</item>
```

Figure 2.1: A lexicon item represented in XML

**add** allows to add a word form. For example, the noun חכמה is also commonly spelled חוכמה, so the additional spelling is specified in the lexicon, next to the standard spelling, using *add* as demonstrated in figure 2.2.

**replace** allows to replace what might be an incorrect inflection of a base form with the correct form. For example, the noun בוקר in plural is בקרים and not the default בוקרים. This is accounted for by using *replace*, as shown in figure 2.3.

**remove** allows to remove inflections. For example, the verb יכול does not have imperative inflections. Figure 2.4 shows a fragment of its lexicon item and how *remove* elements are used to express that.

The attributes of the *add*, *remove* and *replace* elements differ according to the part of speech of the lexicon item.

```
<item id="12876" transliterated="xkmh" undotted="חכמה">
 <noun gender="feminine" number="singular" plural="wt" deverbal="true">
  <add gender="feminine" number="singular" plural="wt" script="colloquial"
       transliterated="xwkmh" undotted="חוכמה"/>
 </noun>
</item>
```

Figure 2.2: A lexicon item for חכמה, exemplifying the addition of a colloquial spelling using an *add* element

Sometimes the base form which is specified in the lexicon is not the most convenient one for generating the inflection paradigm. For example, the preposition עם is a base form, whose entire inflection paradigm is much simpler if עימ is used as the base used for inflection

```
<item id="5805" transliterated="bwqr" undotted="בוקר">
  <noun gender="masculine" number="singular" plural="im">
    <replace gender="masculine" number="plural"
             transliterated="bqrim" undotted="בקרים"
             inflectConstruct="true" inflectPossessive="true" />
  </noun>
</item>
```

Figure 2.3: A lexicon item for בוקר, which exemplifies the usage of the *replace* element

( עימי, עימך, and so on). While *add*, *remove* and *replace* are useful for marking a particular inflected form, using them for specifying all the inflected forms that result from a different inflection base is cumbersome. For such cases we use a mechanism based on an additional attribute, `inflectionBase`. Items with `inflectionBase` attributes are not only processed like other items, but are also included in an additional "inflectionBase" lexicon, which includes the alternative base. During the construction of the morphological analyzer for every part of speech category, the inflected forms that use the "ordinary" lexicon are being *priority unioned* (Beesley and Karttunen (2003), pages 300-307) with the inflected forms that use the "inflectionBase" lexicon, so that the latter override the former. Figure 2.5 lists the lexicon item for the preposition עם, whose `inflectionBase` is עימ.

Items containing *add*, *remove* and *replace* elements are included in the "ordinary" lexicon without the *add*, *remove* and *replace* elements, which are included in additional lexicons according to their element name. The "ordinary" lexicon is used for the "basic" morphological analyzer (after incorporating the `inflectionBase` forms as explained above). Morphological analyzers are built using the same set of rules for *add*, *remove* and *replace* lexicons. The final morphological analyzer is created after the forms generated by the *remove* morphological analyzer are subtracted from the basic morphological analyzer (by a minus operation on the finite-state networks: $basic - remove$). Then the forms generated by the *add* morphological analyzer are added (by the union operation on the finite-state networks: $(basic - remove) \cup add$), and then the forms generated by the *replace* morphological an-

```
<item id="12257" transliterated="ikwl" undotted="יכול">
  <verb binyan="Pa'al" inflectionPattern="1" root="יכל">
    <remove pgn="2p/M/Sg" tense="imperative" transliterated="ikwl"
                                                            undotted="יכול"/>
    <remove pgn="2p/F/Sg" tense="imperative" transliterated="ikli"
                                                            undotted="יכלי"/>
    <remove pgn="2p/M/Pl" tense="imperative" transliterated="iklw"
                                                            undotted="יכלו"/>
    <remove pgn="2p/F/Pl" tense="imperative" transliterated="ikwlnh"
                                                            undotted="יכולנה"/>
    . . .
  </verb>
</item>
```

Figure 2.4: A fragment of the lexicon item for **יכול**, which exemplifies removal of non-existing forms using *remove*

alyzer are being priority unioned (by applying priority union on the finite-state networks: $(replace.P.((basic - remove) \cup add)))$. This final finite-state network contains only and all the valid inflected forms.

The lexicon was initially populated with a small number of words (about 200) used for development of the morphological analyzer. Then, about 3000 nouns and adjectives were automatically added by adopting a subset of the HSpell lexicon. Over 3500 verbs were added from Zdaka (1974). Remaining entries were added manually by a lexicographer using a graphical user interface with the support of the *Knowledge Center for Processing Hebrew*[2]. This effort resulted in a broad-coverage lexicon: we currently have near 20,000 entries, with a near-complete coverage of all categories except proper names.

[2]http://mila.cs.technion.ac.il

```
<item id="16003" transliterated="ym" undotted="עם">
 <preposition inflectionBase="עמי">
  <add personGenderNumber="1p/MF/Sg" script="formal" transliterated="yimdi"
       undotted="עימדי"/>
 </preposition>
</item>
```

Figure 2.5: A lexicon item in XML representations, that exemplifies the usage of *inflectionBase*

```
[[ nounTag idTag {13829} undottedTag {אח} transliteratedTag {ax}
            genderTag %+masculine numberTag %+singular pluralTag %+im
            scriptTag %+formal
] .x. {אחי} ]
```

Figure 2.6: A lexicon item in xfst representation, exemplifying the usage of *inflectionBase*

## 2.2 From XML to xfst

The lexicon is represented in XML, while the morphological analyzer is implemented in xfst. Therefore, the lexicon has to be converted to xfst. The xfst representation is a file containing one xfst *define* statement that is a union of items. Every lexicon item is a concatenation of strings, each with an *upper side* (lexical information) and a *lower side* (the base or bases for the inflections). As a lexicon item is processed by the rules, the upper side is manipulated to reflect the attributes of the inflected form that is being created on the lower side.

Programming in xfst is different from programming in high level languages. While xfst rules are very expressive, and enable a true implementation of some linguistic phenomena, it is frequently necessary to specify, within the rules, information that is used mainly for "book-keeping". Due to the limited memory of finite-state networks, such information is encoded in *tags*, which are multi-character symbols attached to strings. The tags can be manipulated by the rules and thus propagate information among rules. For example, noun lexicon items include the number attribute, so rules which apply to plural nouns only can make use of this information: `$[numberTag %+plural] .o. nouns`. once all linguistic processing is complete, “book-keeping” tags are filtered out.

The upper (lexical) side of lexicon items has the following form: a tag with the part of speech, then an id tag, followed by the id of the item, an undotted tag followed by the undotted string of the item, then a transliterated tag, followed by the transliterated string of the item (this information is created by the conversion process if not found in the XML representation), then the remaining attributes of the item, that are used by the morphological analyzer or that are required in the output.

The lower (surface) side of lexicon items consists of the base form. The base form is usually the same as the `undotted` attribute, but when `inflectionBase` is set, it is used instead. Figure 2.7 lists the xfst representation of the word בלי whose XML lexical entry was listed in figure 2.1.

```
[ [ negationTag idTag {7321} undottedTag {בלי} transliteratedTag {bli}
                  scriptTag %+formal ] .x. {בלי} ]
```

Figure 2.7: The lexicon item of בלי in xfst

Verbs specify the base form of every tense on the lower side, as there is no single base for all tenses. The verb bases are determined by combining a root and a pattern. Given a root, the appropriate pattern for generating the base for every tense is determined by an inflection pattern. Every inflection pattern is associated with a procedure that determines how the root letters are used to form the bases. For example, inflection pattern 23 (binyan Pi’el) defines that the past tense base is formed by inserting י between the first and second letters of the root (the root ש'ל'מ' yields the past tense base שילם). Figure 2.8 demonstrates the lexicon item for the verb שילם in XML, and figure 2.9 lists its xfst representation. Notice that the lexical side contains additional attributes (to those listed in the XML representation), which are set to their default values. The surface side includes the inflection pattern, and the bases for the tenses.

```
<item id="14827" transliterated="eilm" undotted="שילם">
 <verb binyan="Pi'el" inflectionPattern="23" root="שלמ" valence="transitive" />
</item>
```

Figure 2.8: The lexicon item of the verb שילם in XML

```
[
  [
    verbTag idTag {14827} undottedTag {שילם} transliteratedTag {eilm}
    rootTag {שלמ} binyanTag %+Pi'el pgnTag %+NONE inflectionPatternTag {23}
    scriptTag %+formal tenseTag %+NONE valenceTag %+unspecified
  ] .x. [
    ip23Tag originTag {שלמ} bklmTag {שלמ} pastTag {שילמ}
    presentTag {שלמ} futureTag {שלמ} imperativeTag {שלמ}
  ]
]
```

Figure 2.9: The lexicon item of the verb שילם in xfst

## 2.3 The format of the rules

Our rules support surface forms that are made of zero or more prefix particles, followed by a (possibly inflected) lexicon item. Figure 2.10 lists the high-level organization of the analyzer. The variable `inflectedWord` is a union of all the possible inflections of all lexical items. Similarly, `prefixalParticles` is the language of all the possible combinations of prefixes. When the two are combined, they yield a language of all possible surface forms, vastly over-generating. On the upper side of this language a prefix particles filter is composed, which enforces linguistically motivated constraints on the possible combinations of prefixes with inflected forms, reducing the number of possible combinations. On top of this another filter is composed, which handles "cosmetic" changes, such as removing "book-keeping" tags. A similar filter is applied to the the lower side. Details about rules specific to each and every part of speech category are specified in chapter 4.

```
 tagAffixesFilter
 .o.
 prefixalParticlesFilters
 .o.
 [ prefixalParticles inflectedWord ]
 .o.
 removeTagsFilter
];
```

Figure 2.10: A high level view of the analyzer

There are specific rules for every part of speech category, which are applied to the appropriate lexicons. A variable that contains all the base forms and all the inflected forms is defined and assigned for every part of speech. The finite-state networks that are compiled from the regular expressions assigned to these variables make morphological analyzers for every part of speech (these do not include idiosyncrasies nor prefix particles). These part of speech dependent morphological analyzers are too verbose on the lexical side, as they contain all the information that was listed in the lexicon items. As most of the uses of the morphological analyzer require only some of the information, filters are applied to the lexical side to remove the unneeded information.

In general, rules are formed according to the following template:

```
optionalPrefixation [
lexicalSideAlternations
.o.
extractionRules
.o.
setOfwords
.o.
surfaceSideAlternations
] optionalSuffixation
```

The `setOfwords` may denote a lexicon, such as `nouns` lexicon, or a subset of a lexicon with some inflections applied to it, such as `masculinePluralNoun`. Optional affixation is applied to the upper side (adjust lexical information, such as the properties of a possessive suffix), to the lower side (such as the affixes of future tense inflections in verbs), or both. Extraction rules are used to restrict the set of words that the rule is applied to. Alternation rules are used to correct the surface side (e.g., ם is removed from plural nouns in construct form), and to update the lexical side (e.g., gender tag is changed from masculine to feminine when the feminine inflection is created from the masculine form).

# Chapter 3

# The lexicon

## 3.1 Adjectives

Adjectives inflect regularly, with few exceptions. The base form is the absolute singular masculine. The base form is used to generate the feminine form (using one of the ה, ת or ית suffixes), the masculine plural (using the ים suffix) and the feminine plural (using the ות suffix). An additional dimension is status, which can be absolute or construct.

Adjective lexicon items have the following attributes:

**gender**

**number**

**feminine** feminine form suffix: ה, ת or ית.

**plural** plural form suffix, ים for masculine and ות for feminine.

Figure 3.1 lists the lexicon entry of the adjective עילאי: its feminine form is obtained by adding the ת suffix (`feminine="t"`). Note that most of the information is not included as it is assumed by default (that the lexicon item is masculine, singular and so on), but the feminine suffix is explicitly defined. This lexicon entry yields עילאי, עילאית, עילאיים, עילאיות, as well as additional forms including status inflections, using a set of rules that are described in section 4.1.

```
<item id="13852" transliterated="yilai" undotted="עילאי">
 <adjective feminine="t" />
</item>
```

Figure 3.1: A lexicon item for עילאי

Consider now the adjective חיפאי, whose plural masculine form, חיפאיים, occurs in an additional form, חיפאים. Similarly, חיפני is a valid alternative form in addition to חיפאי. These additional forms are handled via *add* elements, as shown in figure 3.2. The *add* element for חיפאים includes the number and gender in addition to the undotted spelling, while the *add* element for חיפני also includes attributes for specifying the feminine and plural suffixes (since they are not the default values). By default, a masculine singular form that is listed using *add* will be used as an additional base for inflections. Note that the additional plural form for the extra base, חיפני, also occurs with a single letter י instead of two in the plural suffix, as the additional *add* element shows.

```
<item id="10193" transliterated="xipai" undotted="חיפאי">
 <adjective feminine="t">
  <add gender="masculine" number="plural" transliterated="xipaim" undotted="חיפאים"/>
  <add feminine="h" gender="masculine" number="singular" transliterated="xipni"
       undotted="חיפני"/>
  <add gender="masculine" number="plural" transliterated="xipnim" undotted="חיפנים"/>
 </adjective>
</item>
```

Figure 3.2: A lexicon item for חיפאי with additional forms

## 3.2 Nouns

The basic form of nouns is the absolute singular masculine form. Hebrew has grammatical gender, and the gender of nouns that denote animate entities coincides with their natural gender. Such nouns usually have both masculine and feminine forms: {כלב, כלבה} and {שוטר, שוטרת}. Where a feminine inflection of a masculine form exists, the masculine is used as the base form. Nouns regularly inflect for number, but some nouns have only a plural or only a singular form. Nouns can be inflected for status and can take possessive suffixes.

Nouns have the following attributes:

**gender** can be `masculine`, `feminine` or `masculine and feminine`

**number** can be `singular`, `plural` and `dual`

**feminine** the suffix for the feminine form: ה, ת or ית.

**plural** the suffix for the plural form, this is usually ים for masculine and ות for feminine.

**deverbal** is this a deverbal noun?

**acronym** is this item an acronym?

**direction** does this noun contain the direction letter ה (E.g., הביתה)?

While the `gender` and `nuber` attributes are lexical, all the rest are support attributes.

The default noun gender is masculine and the default number is singular. As a default there is no feminine inflection and the plural suffix is set to ים. These defaults can be overridden by assigning a different value. The acronym, direction, dual and definiteness properties are set by default to `"false"`.

Figure 3.3 demonstrates a deverbal noun. Figure 3.4 demonstrates a masculine noun with a plural suffix ות, which is usually used for feminine plurals. Another example for a feminine noun with a masculine plural suffix is demonstrated in figure 3.5, where the *replace* operator is used to correct the plural form which is idiosyncratic: שיניים instead of שנות that would have been the default. Figure 3.6 exemplifies how an alternative plural form, סיסמאות, is added to the lexicon item of סיסמה, in addition to the default plural form סיסמות.

```
<item id="10757" script="formal" transliterated="hpgnh" undotted="הפגנה">
 <noun gender="feminine" number="singular" plural="wt" deverbal="true" />
</item>
```

Figure 3.3: A lexicon item for the deverbal noun הפגנה

```
<item id="5044" script="formal" transliterated="ewlxn" undotted="שולחן">
 <noun gender="masculine" number="singular" plural="wt" />
</item>
```

Figure 3.4: A lexicon item for the noun שולחן

```
<item id="13555" script="formal" transliterated="en" undotted="שן">
 <noun gender="feminine" number="singular" plural="im" >
  <add construct="true" gender="feminine" inflectConstruct="true"
        inflectPossessive="true" number="dual and plural" transliterated="eini"
        undotted="שיני"/>
  <replace gender="feminine" inflectConstruct="true" inflectPossessive="true"
            number="plural" transliterated="einiim" undotted="שיניים"/>
 </noun>
</item>
```

Figure 3.5: A lexicon item for the noun שן

```
<item id="10" transliterated="sismh" undotted="סיסמה">
 <noun gender="feminine" number="singular" plural="wt">
  <add gender="feminine" number="plural"
        inflectConstruct="true" inflectPossessive="true"
        transliterated="sismawt" undotted="סיסמאות"/>
 </noun>
</item>
```

Figure 3.6: A lexicon item for the noun סיסמה

## 3.3 Verbs

The base form of verbs is the third person, singular masculine past tense form. This is the standard form used in dictionaries and hence is what our morphological analyzer produces. Verbs inflect for number, gender, person and tense, and "transitive" verbs can take accusative suffixes. The attributes used for verbs are:

**inflectionPattern** a number that corresponds to one or more paradigms in Barkali (2000).

**root**

***binyan*** the *binyan* is deduced from the inflection pattern.

**valence** setting this attribute to "transitive" enables the accusative inflections to be generated.

The inflection patterns that are used by the morphological analyzer are based on the paradigms listed in Barkali (2000). Barkali's paradigms are designed for dotted Hebrew, whereas our morphological analyzer processes undotted script, and therefore many of the paradigms collapse into a smaller set (61) of what we call *inflection patterns*. Inflection patterns determine the entire inflection paradigm of a verb base (which is viewed as a combination of a root and a *binyan*), across all the possible values of number, gender, person and tense. For example, inflection pattern `23` groups together verbs whose inflections behave similarly to that of the verb **שילם**. The inflection pattern determines the entire paradigm, which is similar also for **סיפסר** or **איתר**, among others. In contrast, a verb like **דימה**, which shares the same *binyan* with **שילם**, is assigned to a different inflection pattern because its inflection is different due to its "weak" root. Clearly, inflection patterns are more specific than *binyanim*, but all the verbs in the same inflection pattern share the same *binyan*.

Unlike other parts of speech, the inflection bases of verbs are not listed in their lexicon items. Instead, the bases are created during the translation of the XML lexicon into xfst by interdigitating roots with patterns. The patterns that are used are determined by the `inflectionPattern` attribute. A verb has more than one base for inflection, and the xfst representation of a verb lexicon item includes on the surface side a base for every tense. The tenses are bare-infinitive, infinitive, past, present, future and imperative. Figure 3.7 lists the XML lexicon item of the verb הוציא. Figure 3.8 demonstrates the lexicon entry for הוציא in xfst format.

```
  <item id="3374" script="formal" transliterated="hwcia" undotted="הוציא">
    <verb binyan="Hif'il" inflectionPattern="49" root="יצא" valence="transitive" />
  </item>
```

Figure 3.7: A lexicon item for the verb הוציא

```
[
 [
  verbTag idTag {16557} undottedTag {הוציא} transliteratedTag {hwcia}
  rootTag {יצא} binyanTag %+Hif'il inflectionPatternTag {49}
  scriptTag %+formal valenceTag %+transitive
 ] .x. [
  ip49Tag originTag {הוציא} bklmTag {הוציא} pastTag {הוציא}
  presentTag {וציא} futureTag {וציא} imperativeTag {הוציא}
 ]
]
```

Figure 3.8: The lexicon item הוציא in the verbs lexicon in xfst format

## 3.4 Other parts of speech

### 3.4.1 Adverb

Some of the adverbs inflect for person/gender/number (for example, לאט: לאיטי, לאיטך, לאיטו, לאיטנו, and so on). Some adverbs are interrogatives. Adverbs have the following attributes in the lexicon:

**interrogative** is the adverb an interrogative?

**inflect** does the adverb inflect for person/gender/number?

**position** position of the adverb with respect to the entity it modifies (possible values are: `pre`, `post`, or `pre and post`).

**modifiesVerb** does the adverb modify a verb?

**modifiesAdjective** does the adverb modify an adjective?

**modifiesAdverb** does the adverb modify an adverb?

Many adverbs are derived from nouns, but as this is not systematic (derivational morphology rather than inflectional morphology) we do not support such derivations automatically. Figure 3.9 lists the lexicon item for the adverb **היטב**.

```
<item id="7225" script="formal" transliterated="hivb" undotted="היטב">
 <adverb inflect="false" interrogative="false" modifiesVerb="true" position="post"/>
</item>
```

Figure 3.9: A lexicon item for **היטב**

### 3.4.2 Conjunction and negation

We distinguish between three types of conjunctions: coordinating, subordinating and relativizing. Figure 3.10 exemplifies all three types.

The lexicon does not provide any special attributes for negation words: they are simply marked as "negation". Figure 3.11 exemplifies the negation **בלי**.

### 3.4.3 Interjection

Interjections do not usually inflect, but a few do (for person/gender/number). The default for inflection is `"false"` for no inflection. Figure 3.12 exemplifies the non-inflecting interjection **לעזאזל** and the inflecting interjection **אשרי**. Note the *replace* operator which is used to replace the invalid inflection **אשריי** with the proper surface form **אשרי**.

```
<item id="11548" script="formal" transliterated="lpikk" undotted="לפיכך">
  <conjunction type="coordinating"/>
</item>
<item id="9634" script="formal" transliterated="ki" undotted="כי">
  <conjunction type="subordinating"/>
</item>
<item id="9688" script="formal" transliterated="aer" undotted="אשר">
  <conjunction type="relativizing"/>
</item>
```

Figure 3.10: Lexicon items for the coordinating conjunction לפיכך, the subordinating conjunction כי and the relativizing conjunction אשר

```
<item id="4807" script="formal" transliterated="bli" undotted="בלי">
 <negation />
</item>
```

Figure 3.11: A lexicon item for the negation בלי

### 3.4.4 Preposition

All prepositions inflect for person/gender/number using pronomial suffixes. Idiosyncrasies are handled by the `inflectionBase` attribute where the base changes or the *add remove* and *replace* elements for less systematic exceptions. Prepositions in the lexicon may have the following attribute:

**case** the case can be either accusative (only for the preposition את) or possessive (for של), but normally neither, so the default is `"unspecified"`.

We assume that prepositions cannot take the prepositional prefixes ({ב', כ', ל', מ'}), so a preposition such as אחרי is considered a different preposition from מאחרי or לאחרי, and so we assign to each its own lexicon item.

Figure 3.13 demonstrates the lexicon item of עבור and לפני. The latter exemplifies usage of *replace* elements that correct parts of the inflection: לפני instead of לפניי, לפניהם instead

```
<item id="14321" script="formal" transliterated="lyzazl" undotted="לעזאזל">
 <interjection inflect="false"/>
</item>
<item id="5423" script="formal" transliterated="aeri" undotted="אשרי">
 <interjection inflect="true">
  <replace personGenderNumber="1p/MF/Sg" script="formal" transliterated="aeri"
           undotted="אשרי"/>
 </interjection>
</item>
```

Figure 3.12: Lexicon items for the interjections לעזאזל and אשרי

of לפנים and לפניהן instead of לפנין.

### 3.4.5 Pronoun

We distinguish between the following types of pronouns:

**personal** such as אני and אתם.

**demonstrative** such as אותו and אלה.

**interrogative** (relativizer) such as מי and מהם.

**impersonal** such as מישהו and אלמוני.

Pronouns have the following attributes: `gender`, `number`, `person` and `type`. Figure 3.14 demonstrates a lexicon item of a pronoun.

### 3.4.6 Proper name

We consider names of people, of locations, of organizations, of products etc. as types of proper names. Proper names scarcely take the definite article. Figure 3.15 demonstrates a lexicon item of a proper name. Proper names have the attributes `gender` and `number` as well as:

**type** can be `person` (such as שלמה), `location` (such as ירושלים), `organization` (such as הטכניון), `product` (such as במבה) and `dateTime` (such as תשרי).

```
<item id="85" script="formal" transliterated="ybwr" undotted="עבור">
 <preposition case="unspecified" inflect="true"/>
</item>
<item id="1467" script="formal" transliterated="lpni" undotted="לפני">
 <preposition case="unspecified" inflect="true">
  <replace personGenderNumber="1p/MF/Sg" script="formal"
           transliterated="lpni" undotted="לפני"/>
  <replace personGenderNumber="3p/M/Pl" script="formal"
           transliterated="lpnihm" undotted="לפניהם"/>
  <replace personGenderNumber="3p/F/Pl" script="formal"
           transliterated="lpnihn" undotted="לפניהן"/>
 </preposition>
</item>
```

Figure 3.13: Lexicon items for the prepositions עבור and לפני

```
<item id="6224" script="formal" transliterated="kzh" undotted="כזה">
 <pronoun gender="masculine" number="singular" person="3" type="demonstrative"/>
</item>
```

Figure 3.14: The lexicon item of the pronoun כזה

**definiteness** does the proper name allow the definite article to prefix it?

**direction** for example ירושלימה, which is equivalent to לירושלים (to Jerusalem).

### 3.4.7 Quantifier

We distinguish among the following types of quantifiers:

**numeral ordinal** such as ראשון, שני, שלישית....

**numeral cardinal** such as אחת, שתיים, שלוש....

**numeral fractional** such as חצי, שליש....

```
<item id="1454" script="formal" transliterated="elmh" undotted="שלמה">
 <properName gender="masculine" number="singular" type="person"/>
</item>
```

Figure 3.15: The lexicon item of the proper name שלמה

**gematria** this is used mostly for representing years in Hebrew dates, e.g., תשס"ד, but can also be used elsewhere, e.g., ל"ו in ל"ו צדיקים.

**non-numeral** such as מרבית.

Quantifiers have the `gender`, `number` and `type` attributes. Numerals have feminine and masculine forms and can inflect for status. The `inflect` attribute is used to signal whether or not the lexical item inflects for person/gender/number. Figure 3.16 demonstrates a lexicon item of a quantifier.

```
<item id="7052" script="formal" transliterated="kpliim" undotted="כפליים">
 <quantifier gender="unspecified" number="dual" inflect="false" inflection-
Base="" type="non-numeral"/>
</item>
```

Figure 3.16: The lexicon item of the quantifier כפליים

# Chapter 4

# The rules

In this chapter we present the set of rules used by the morphological analyzer, i.e., the implementation of linguistic structures in xfst. The system includes hundreds of rules and obviously not all of them can be listed here[1]. We present a small subset of the rules, that exemplifies the principles that govern the overall organization of the system. The linguistic information was collected from several sources (Schwarzwald, 2001; Schwarzwald, 2002; Alon, 1995; Barkali, 2000; Zdaka, 1974; Cohen, 1996; Ornan, 2003).

Many of the categories can combine with pronomial suffixes which function as possessive in the case of nouns, accusatives in the case of verbs, prepositional objects in the case of prepositions, etc. As the form of pronomial suffixes in Hebrew is independent of their function, we treat them uniformly. The suffixes are defined as transducers. For example, the pronomial suffix for first person, singular yields `[pronomialSuffix]+1p/MF/Sg` on the upper side and י on the lower side.

## 4.1 Adjectives

The base form of adjectives is the singular masculine absolute form. Other forms are generated from this base form. Figure 4.1 demonstrates how the base forms are extracted from the adjective lexicon. The adjective lexicon is a union of all lexicon items whose part of

[1]The source code is available at http://cl.haifa.ac.il/projects/hebmorph/index.html

speech is `adjective`. The variable `adjective` is the union of all adjective lexicon items, as generated by the process that translates the XML lexicon into xfst. A filter allows only lexicon items containing the string `numberTag +singular` (i.e., items in singular form) in their upper side to be extracted. On top of the number filter there is a gender filter that allows only items containing a masculine gender tag to be extracted. The result is a variable `masculineSingularAdjective` that denotes all the masculine singular adjectives in the lexicon.

```
define masculineSingularAdjective [
 $[genderTag [%+masculine| %+masculine% and% feminine]]
 .o.
 $[numberTag %+singular]
 .o.
 adjective
];
```

Figure 4.1: Singular masculine adjectives

The feminine singular form is generated from the masculine singular by adding a suffix. This suffix can be either ה, or ת or ית. Of course, there might be idiosyncratic forms that have no masculine singular form, and do have a feminine singular form, for example הרה (pregnant). Therefore, as figure 4.2 shows, a singular feminine adjective is either extracted as is from the lexicon or generated from a singular masculine with the appropriate suffix.

The rule `[ %+feminine <- ? || genderTag _ ]`, as figure 4.3 exemplifies, updates the gender attribute to `feminine` for the inflected feminine forms.

Figure 4.4 shows how the suffix ה is used in the inflection. Notice that the default is not to add an additional ה if the masculine adjective already terminates with it, as in מורה → מורה (teacher). This means that exceptions to this default, such as גבוה → גבוהה (tall), are being improperly treated. All the over-generated forms are being handled and corrected at a later stage, as explained in section 2.1. Figure 4.5 shows how the inflection using the suffix ת (handling of the ית suffix is similar).

```
define feminineSingularAdjective [
 [
  $[genderTag [%+feminine| %+masculine% and% feminine]]
  .o.
  adjective
 ]|
 masculineSingularAdjective2feminineSingularAdjective
];
```

Figure 4.2: A singular feminine adjective is either extracted from the lexicon or inflected from the singular masculine

The construct form for masculine singulars is the same as the absolute form, so only the lexical side is being updated by setting the construct tag to be `constructTag +true`. For singular feminine forms the same applies, except that final ה changes to ת as figure 4.6 shows. Notice that lexicon items that are marked with `inflectConstruct="false"` are excluded.

All adjectives can also be nouns, and adjectives with possessive suffixes are implicitly nouns. Therefore, we allow adjectives to take the possessive suffixes, but the part of speech category is changed from `adjective` to `noun`. The possessive inflections are created by adding the appropriate suffix (according to person, number and gender) to the construct form. Lexicon items whose `inflectPossessive` attribute is `"false"` are not being processed, and the same holds for lexicon items that already have a possessive suffix (this happens when listing exceptions using the *add*, *remove* or *replace* operators), as figure 4.7 demonstrates. Note how the rule template mentioned in section 2.3 is applied here. The possessive inflection for singular feminine is similar.

The plural forms are based on the singular masculine form. Plural masculine forms take the ים suffix, whereas the plural feminine suffix is ות. Figure 4.8 shows how the plural form with the ים suffix is obtained. Figure 4.9 shows that plural masculine adjectives are either generated from the base or extracted directly from the lexicon (exceptions that have the plural form specified). For the plural feminine forms a similar set of rules is used, except that

```
define masculineSingularAdjective2feminineSingularAdjective [
 [ %+feminine <- ? || genderTag _ ]
 .o.
 [
  masculineSingularAdjective2feminineHSingularAdjective |
  masculineSingularAdjective2feminineTSingularAdjective |
  masculineSingularAdjective2feminineITSingularAdjective
 ]
];
```

Figure 4.3: Transforming the gender tag from masculine to feminine.

the suffix is ות and a final ה in singular form is elided before adding the ות suffix.

The construct form of feminine plurals is identical to the absolute form, but the construct form of masculine plurals differs from the absolute masculine plural as a result of reducing the plural suffix from ים to י. The possessive inflections for the plural forms are based on the construct plural forms, adding the plural possessive suffixes. The possessive suffixes for plural are similar to those of the singular, with an additional letter י. So, the possessive suffix for the second person, masculine, plural will be יכם, and not כם (the singular possessive suffix).

Lexicon items that specify the `inflectionBase` attribute should be inflected using that base and not using the default base. As the rules apply to all the items, there is some overgeneration. Lexicon items that have an `inflectionBase` specified are inflected properly using special rules. Figure 4.10 shows the usage of priority union for handling the `inflectionBase`.

Figure 4.11 shows the combination of all types of adjectives. Note that possessive inflections convert the adjectives into nouns.

```
define masculineSingularAdjective2feminineHSingularAdjective [
 [
  [
   $[feminineTag %+h]
   .o.
   masculineSingularAdjective
  ]
  [ 0 .x. addedHE ]
 ]
 .o.
 [ addedHE -> 0 || HE _ .#. ]
 .o.
 [ addedHE -> HE ]
];
```

Figure 4.4: Converting a singular masculine adjective into a singular feminine with the suffix ה

```
define masculineSingularAdjective2feminineTSingularAdjective [
 [
  $[feminineTag %+t]
  .o.
  masculineSingularAdjective
 ]
 [ 0 .x. TAV ]
];
```

Figure 4.5: Converting a singular masculine adjective into a singular feminine with the suffix ת

```
define constructFeminineSingularAdjective [
 ~$[inflectConstructTag %+false]
 .o.
 [
  feminineSingularAdjective
  [ [constructTag %+true] .x. 0 ]
 ]
 .o.
 [ HE -> TAV || _ .#. ]
];
```

Figure 4.6: Singular feminine construct

```
define possessiveSuffixMasculineSingularAdjective [
 [
  [ 0 <- possessiveSuffixTag %+unspecified ]
  .o.
  [ %+false <- %+true || constructTag _ ]
  .o.
  ~$[possessiveSuffixTag]
  .o.
  ~$[inflectPossessiveTag %+false]
  .o.
  constructMasculineSingularAdjective
 ]
 singulaPossessiveSuffixes
];
```

Figure 4.7: The possessive inflection for singular masculine adjectives

```
define pluralIMAdjective [
 [
  [ %+plural <- %+singular || numberTag _ ]
  .o.
  $[pluralTag %+im]
  .o.
  masculineSingularAdjective
  .o.
  [ HE -> 0 || _ .#. ]
 ]
 [
  0 .x.  [ YOD FINALMEM ]
 ]
];
```

Figure 4.8: Plural inflection with ים suffix

```
define masculinePluralAdjective [
 pluralIMAdjective |
 [
  $[genderTag [%+masculine| %+masculine% and% feminine]]
  .o.
  $[numberTag [%+plural|singular% and% plural]]
  .o.
  adjective
 ]
];
```

Figure 4.9: Plural masculine adjectives

```
define possessiveSuffixSingularAdjectivePriorirtyUnioned [
 [
  [
    possessiveSuffixSingularAdjective.u -
    possessiveSuffixSingularAdjectiveByinflectionBase.u
  ]
  .o. possessiveSuffixSingularAdjective
 ] |
 possessiveSuffixSingularAdjectiveByinflectionBase
];
```

Figure 4.10: Singular possessive inflections undergoing priority union

```
define Adjective [
 masculineSingularAdjective |
 feminineSingularAdjective |
 constructMasculineSingularAdjective |
 constructFeminineSingularAdjective |
 masculinePluralAdjective |
 femininePluralAdjective |
 constructMasculinePluralAdjective |
 constructFemininePluralAdjective |
 [
  [ nounTag <- adjectiveTag ] !! possessives are nouns
  .o.
  [
   possessiveSuffixSingularAdjectivePriorirtyUnioned |
   possessiveSuffixPluralAdjectivePriorirtyUnioned
  ]
 ]
];
```

Figure 4.11: Adjectives

## 4.2 Nouns

Nouns inflect for gender, number and status and take possessive suffixes. The base form is the singular absolute. The rules we implement support masculine and feminine, singular, dual and plural, construct and absolute forms, with or without possessive suffixes as figure 4.12 shows.

```
define Noun [
 [
  masculineSingularNoun |
  feminineSingularNoun |
  masculineDualNoun |
  feminineDualNoun |
  constructMasculineSingularNoun |
  constructFeminineSingularNoun |
  masculinePluralNoun |
  femininePluralNoun |
  constructMasculinePluralNoun |
  constructFemininePluralNoun |
  [
   possessiveSuffixSingularNounPriorirtyUnioned |
   possessiveSuffixPluralNounPriorirtyUnioned
  ]
 ]
];
```

Figure 4.12: Nouns

The handling of nouns is very similar to that of adjectives, but includes more special cases, as nouns do not inflect as regularly as adjectives. For example, masculine plurals by default take the ים suffix, while feminine plurals take the ות suffix, but there are several exceptions to both defaults.

Singular feminine nouns are either specified in the lexicon or are inflected from the singular masculine. When the singular feminine is inflected from the singular masculine the suffix ה or ת or ית is added. Figure 4.13 shows how feminine singular nouns are processed. Notice that the appropriate feminine suffix is marked in the lexicon and this hint is used when the feminine form is generated.

```
define feminineSingularNoun [
 [ 0 <- constructTag [%+false|%+unspecified]]
 .o.
 ~$[constructTag %+true]
 .o.
 [ [
   $[genderTag [%+feminine|%+masculine% and% feminine]]
   .o.
   $[numberTag %+singular]
   .o.
   noun
  ] | [
   [%+feminine <- %+masculine || genderTag _ ]
   .o.
   [
    [[ $[feminineTag %+h] .o. masculineSingularNoun] [0 .x. HE]] |
    [[ $[feminineTag %+t] .o. masculineSingularNoun] [0 .x. TAV]] |
    [[ $[feminineTag %+it] .o. masculineSingularNoun] [0 .x. [YOD TAV]]]
   ] ] ] ];
```

Figure 4.13: Singular feminine nouns

Dual nouns are listed in the lexicon and therefore are directly extracted. Construct and possessive inflections of nouns are similar to the case of adjectives. The handling of plurals is more complicated in nouns than in adjectives, as there are more possible suffixes and as the ים and the ות suffixes cannot be predicted by gender. Figure 4.14 shows how plurals (regardless of gender) with ים suffix are processed. Only singular nouns with the appropriate suffix are extracted from the nouns lexicon. On the lower side, final letter ה, if it exists, is removed, and the ים suffix is concatenated. On the upper side the number tag is updated from singular to plural.

```
define pluralIMNoun [ [
  [ %+plural <- %+singular || numberTag _ ]
  .o.
  $[numberTag %+singular]
  .o.
  $[pluralTag %+im]
  .o.
  noun
  .o.
  [ HE -> 0 || _ .#. ]
 ]
 [ 0 .x.  [ YOD FINALMEM ] ]
];
```

Figure 4.14: Plural nouns with ים suffix

Figure 4.15 shows how the plural nouns with the ות suffix are processed. On the lower side some conditional alternations are performed before the ות suffix is added. Alternation rule #1 ensures that nouns such as שנייה (a second) are properly inflected to שניות (seconds) and not to שנייות*. Notice that final letter ה is removed as rule #4 suggests. Alternation rule #2 is used for ensuring that a singular noun such as משאית (a truck) is properly inflected to its plural form משאיות. Alternation rule #3 ensures that singular nouns such as סמכות (authority) are properly inflected into their plural form סמכויות. Of course, exceptional cases such as חנית (a spear) that should be properly inflected as חניתות and not as חניות*, are handled separately (see section 2.1). The יות and the אות suffixes are less frequent and more regular than the previously mentioned plural suffixes, and are handled by concatenating the suffix to the singular form. Figure 4.16 describes plural nouns: they are either extracted from

```
define pluralWTNoun [
 [
  [ %+plural <- %+singular || numberTag _ ]
  .o.
  $[numberTag %+singular]
  .o.
  $[pluralTag %+wt]
  .o.
  noun
  .o.
  [ YOD YOD HE -> YOD HE || _ .#. ] !! [1]
  .o.
  [ ALEF YOD TAV -> ALEF YOD || _ .#. ] !! [2]
  .o.
  [ VAV TAV -> VAV YOD || _ .#. ] !! [3]
  .o.
  [ [HE|TAV] -> 0 || _ .#. ] !! [4]
 ]
 [ 0 .x. [VAV TAV] ]
];
```

Figure 4.15: Plural nouns with ות suffix

the lexicon or inflected using the rules listed above. Figure 4.17 describes plural feminine nouns, either extracted from the lexicon, or inflected using the rules above, or inflected from the singular masculine forms (in cases where they can inflect as feminine).

```
define masculinePluralNoun [
 [ 0 <- constructTag [%+false|%+unspecified]]
 .o.
 ~$[constructTag %+true]
 .o.
 $[genderTag [%+masculine| %+masculine% and% feminine]]
 .o.
 [
  pluralIMNoun  |
  pluralWTNoun  |
  pluralIWTNoun |
  pluralAWTNoun ! just in case...
  [
   $[numberTag %+plural]
   .o.
   noun
  ]
 ]
];
```

Figure 4.16: Plural masculine nouns

```
define femininePluralNoun [
 [ 0 <- constructTag [%+false|%+unspecified]]
 .o.
 ~$[constructTag %+true]
 .o.
 $[genderTag [%+feminine| %+masculine% and% feminine]]
 .o.
 [
  [ ! handles the forms generated from the singular masculine lexicon item
   [
    [ %+feminine <- %+masculine || genderTag _ ]
    .o.
    [ %+plural<- %+singular || numberTag _ ]
    .o.
    $[[feminineTag %+it]|[feminineTag %+t]|[feminineTag %+h]]
    .o.
    $[numberTag %+singular]
    .o.
    $[genderTag %+masculine]
    .o.
    noun
    .o.
    [ HE -> 0 || _ .#. ]
   ]
   [ 0 .x. [VAV TAV] ]
  ] |
  [ $[numberTag %+plural] .o.  noun ! extract plural nouns from the lexicon ] |
  pluralIMNoun  | pluralWTNoun  | pluralIWTNoun | pluralAWTNoun
 ]
];
```

Figure 4.17: Plural feminine nouns

## 4.3 Verbs

Inflected verb forms are obtained by adding affixes to a base form. As explained in section 3.3, the surface side of the xfst representation of verb lexicon items includes an inflection base for every tense which is third person masculine singular. The rules for verbs only handle the affixation and do not have to handle interdigitation. Rules are applied to verbs according to their inflection pattern.

Verbs inflect in the bare-infinitive (one form + 10 inflections for accusative suffixes), the infinitive (4 forms), past tense (10 forms), present (4 forms), future (10 forms) and imperative (4 forms). So for every combination of root and inflection pattern $11+4+10+4+10+4 = 43$ inflected forms are created (more, when alternative script rules are involved). We do not support accusative suffixes for tenses other than the bare-infinitive due to their rareness, but such support can be easily added when required.

Initially, all verb inflections are generated according to all the verb paradigms, regardless of their inflection pattern. This stage is performed by sets of rules, each called a *class*. Different classes handle different variations of inflection. For example, different rules are applied to verbs with נ as their first root letter, depending on the paradigm: הנשמתי (נ'ש'מ') vs. הפלתי (נ'פ'ל'). Every inflection pattern has its own configuration of class rules. For example, inflection pattern 1 uses only rules from class 1, but inflection pattern 40 uses rules from class 1, class 3, class 4 and class 5. Inflection pattern rules just pick the appropriate class rules for every inflection in every tense, so they behave as a "menu" of class rules. We demonstrate rules used for generating some of the inflected forms of the verb שילם whose lexicon item was listed in figure 2.8, in XML, and in figure 2.9 in xfst.

Figure 4.18 shows how past tense first person plural forms are obtained for verbs with inflection pattern 23, e.g., שילם. The variable `class1Past1pMFPl` represents all verbs inflected for past tense first person plural using *class1* rules. The `$[inflectionPatternTag {23}]` filter extracts only verbs that are tagged with the required inflection pattern.

Figure 4.19 lists the class 1 rule for inflecting verbs for past tense first person plural form. The variable `pastBase` represents past tense bases of all verbs in the lexicon, as shown

```
define ip23Past1pMFPl [
 $[inflectionPatternTag {23}]
 .o.
 class1Past1pMFPl
];
```

Figure 4.18: Rule for creating the past tense, 1st person, plural inflection for ip23

by figure 4.20. The filters composed on top of the variable set the tense to `past` and the inflection to `+1p/MF/Pl`, on the lexical side. On the surface side the appropriate suffix נו is concatenated, thus generating the string שילמנו on the surface side.

```
define class1Past1pMFPl [
 [
   [ %+1p%/MF%/Pl <- ? || pgnTag _ ] .o. [ %+past <- ? || tenseTag _ ]
  .o.
  pastBase
 ]
 [ 0 .x.  [NUN VAV] ]
];
```

Figure 4.19: Class 1 rule for inflecting verbs for past tense, first person, plural

As another example, figure 4.21 lists the class 2 rules used for the weak inflections where the last root letter נ is assimilated into the suffix נו, as in שמנו, for the root {ש'מ'נ'}. Other inflections for the various tenses and classes are performed in a similar way. The 61 inflection patterns use rule sets from 11 classes, where each class has distinct rules for every combination of tense, gender, number and person. The entire set of rules for verb inflections constitutes a substantial part of the morphological analyzer, but the organization delineated above allows for a modular way of specifying the rules which facilitates relatively easy development and maintenance.

```
define pastBase [
 Verbs
 .o.
 ~$[pastTag %+NONE]
 .o.
 [ ?+ pastTag -> 0 || .#. _ ]
 .o.
 [ presentTag ?+ -> 0 || _ .#. ]
];
```

Figure 4.20: Rule extracting the past tense base from the lexicon items

```
define class2Past1pMFPl [
 [
  [%+1p%/MF%/Pl <- ? || pgnTag _ ] .o. [ %+past <- ? || tenseTag _ ]
  .o.
  pastBase .o.  [NUN -> 0 || _ .#.]
 ]
 [ 0 .x.  [NUN VAV] ]
];
```

Figure 4.21: Class 2 rule for inflecting verbs for past tense, first person, plural

## 4.4 Other parts of speech

Conjunctions, negations, proper names and pronouns do not inflect, and are therefore extracted directly from the lexicon with no rules applying to them.

Adverbs, interjections and prepositions may or may not take pronomial suffixes, depending on the value of the `inflectTag` attribute. Figure 4.22 shows the rule that adds the pronomial suffixes to adverbs whose `inflectTag` is set to `true`. Interjections and prepositions are treated similarly.

```
define adverbInflections [
 [
  $[inflectTag %+true]
  .o.
  adverb
 ] singularPronomialSuffixes
];
```

Figure 4.22: Adverbs with pronomial suffixes

Figure 4.23 shows the variable `Adverb` that contains adverbs extracted from the lexicon plus those that were generated by adding pronomial suffixes to their base forms.

```
define Adverb [
 [ adverb | adverbInflections ]
];
```

Figure 4.23: Adverbs

Quantifiers that combine with pronomial suffixes are handled similarly to nouns and adjectives (and are not further elaborated here). Other types of non-lexical quantifiers include gematria, numbers and inflected numerals. Gematria assigns a numeric value to each letter, which is used in practice in Jewish dates and in some cases for numbering. Figure 4.24 shows the regular expression for handling gematria. The `finalLetter` filters (upper and lower) are used for allowing final and ordinary version of letters appear as the last letter in the expression (for example, to allow תש"ן and תש"נ). A gematria expression can be made of one letter followed by a single quote symbol, or by an expression with two, three, four or five letters. Expressions with more than one letter contain a double quote symbol in the penultimate position, and this is handled by the two `doubleQuote` filters (upper and lower).

```
define gematriaNumbers [
 finalLetterUpper
 .o.
 [
  [oneLetterGematria ’] |
      [
       doubleQuoteUpper
       .o.
       [ twoLetterGematria | threeLetterGematria |
        fourLetterGematria | fiveLetterGematria ]
       .o.
       doubleQuoteLower
      ]
 ]
 .o.
 finalLetterLower
];
```

Figure 4.24: Gematria

Supported numbers follow the expressions in figure 4.25. They describe integers, numbers with a decimal point, and integers that have a comma that delimits every three digits.

```
define ordinaryNumbers [
 [ (%+|%-) DIGIT+ ] |
 [ ((%+|%-) DIGIT+) %. DIGIT+ ]
];
define numbers [
 [quantifierTag undottedTag]:0
 [
  ordinaryNumbers |
  [ (%+|%-) DIGIT^{1,2} [%, DIGIT^3]+]
 ]
 [typeTag %+literal% number]:0
];
```

Figure 4.25: Numbers

## 4.5 Prefixal particles

Several particles which are free morphemes in other languages are expressed in Hebrew as prefixes, which orthographically combine with the words that immediately follow them. Figure 4.26 lists the various prefix particles, specifying their surface form and their morpho-syntactic function.

| | *morpho-syntactic function* | *surface form* |
|---|---|---|
| 1 | preposition | ב |
| 2 | preposition + definite article | ב |
| 3 | relativizer | ה |
| 4 | interrogative | ה |
| 5 | definite article | ה |
| 6 | tense inversion | ו |
| 7 | conjunction | ו |
| 8 | preposition | כ |
| 9 | preposition + definite article | כ |
| 10 | adverb | כ |
| 11 | temporal subordinating conjunction | כש |
| 12 | preposition | ל |
| 13 | preposition + definite article | ל |
| 14 | temporal subordinating conjunction | לכש |
| 15 | preposition | מ |
| 16 | temporal subordinating conjunction | מש |
| 17 | subordinating conjunction | מש |
| 18 | relativizer | ש |
| 19 | subordinating conjunction | ש |

Figure 4.26: Prefix particles and their morpho-syntactic functions

Some of the prefix particles are rarely used in Modern Hebrew: The tense inversion ו is obsolete and the interrogative ה is gradually disappearing. The temporal subordinating conjunction מש is less frequent than the other two temporal subordinating conjunctions.

Several prefix particles can combine with a single word. For example, ו+ל+בנה (and to her son). This leads to ambiguity as the prefixes can, sometimes, be part of the stem, for example ולבנה can also be read as "and a brick". In this section we discuss the possible combinations of prefixal particles.

Shapira and Choueka (1964) dealt with the problem of removing prefixes by manually building a list of 45 possible combinations of prefixes, found in a test text. The method of building the list is not mentioned except for a constraint that a combination of prefixes cannot be longer than four letters. Alon (1995) mentions in general a few possible combinations of prefixes, but does not attempt to describe the phenomena in full. Schwarzwald (2001) generally states that certain constrains hold between the prefixes and the category they combine with. For verbs, nouns and adjectives she provides a grammar. Schwarzwald (2002), pages 139-144, describes several possible combinations of the prefixes in a discussion about morphotactics, including constraints that restrict certain combinations with nouns, verbs and particles, and determines that the length of the longest combination of prefixes is 4 letters (similar to the result of Shapira and Choueka). The most comprehensive treatment of prefixes and their valid combinations was implemented in HSpell[2]. The system includes a list of possible combinations of prefix particles and uses some syntactically motivated rules in order to decide when a combination can be prefixed to a word according to the word's part of speech and some attributes such as status and tense. The list produced by HSpell probably covers well the performance in Modern Hebrew written texts although it does not attempt to cover the competence. Moreover, the production of the list and the rules that constrain the possible combinations are encoded in a computer program and thus are not easily accessible for linguists.

We maintain a list of several dozens of prefix sequences that were observed in a corpus of over ten million words (a few additional combinations are commonly heard in speech, but did not appear in the corpus). Our morphological analyzer incorporates this list. A sequence of prefixes may or may not combine with a given word, based on syntactically motivated

[2]http://www.ivrix.org.il/projects/spell-checker/

constraints which refer to the part of speech of the word and to the function of the *last* prefix in the sequence. The following constraints therefore examine the last prefix only. They are implemented by applying a set of rules on the upper side of the morphological analyzer's finite-state network. For example, the rule `~$[ prepositionTag ?+ verbTag ]` prevents prepositions from combining with verbs.

### 4.5.1 Prepositions: {ב', כ', ל', מ'}

- Prepositional prefixes can prefix nouns, adjectives, numerals, proper names, quantifiers and some of the pronouns.
- Prepositional prefixes do not combine with verbs unless the verb is in infinitive form. Note that all present tense verbs in Hebrew are ambiguous and can be considered also nouns or adjectives. If such an ambiguous word is prefixes by a preposition, the ambiguity is reduced and the word is marked as being either a noun or an adjective, but *not* a present tense verb.

### 4.5.2 Definite article: {ה'}

- The definite article can prefix nouns, adjectives and numerals unless they are in construct form or have a possessive suffix.
- The definite article does not prefix proper names in general. Some exceptions exist (mostly locations and sometimes names of organizations).
- The definite article does not prefix nouns, adjectives and numerals that already have the definiteness property, or the direction property (ה' המגמה).
- The definite article, when combined with a personal pronoun, causes the pronoun to become a demonstrative pronoun.

### 4.5.3 Relativizer: {ה'}

- The relativizer ה' occurs only before a verb in the present tense.

### 4.5.4 Conjunction: {ו'}

- The conjunction ו' can prefix any Hebrew word, excluding the conjunction ו' itself and a few others (אבל, או, שכן, אפוא, דהיינו, כלומר, ברם).

### 4.5.5 Adverb: {כ'}

- The adverb כ' can prefix numerals and time related terms, such as שעה, שבועיים and so on.

### 4.5.6 Temporal subordinating conjunctions: {כש־, לכש־, מש־}

- The subordinating conjunctions do not prefix conjunctions, interjections and interrogatives.

### 4.5.7 Subordinating conjunction: {ש'} and the relativizer {ש'}

- ש' do not prefix verbs in imperative form, interrogatives and conjunctions.

# Chapter 5

# Conclusion

We described a broad-coverage finite-state morphological analyzer for undotted Hebrew texts. The analyzer consists of two main components: a lexicon and a set of rules. The current underlying lexicon includes over 19,000 items. The average number of inflected forms for a lexicon item is 30. Counting also forms that include prefix particles, the morphological analyzer supports over 45 million surface forms. Due to the usage of finite-state technology the morphological analyzer can also be used for generation and the time complexity of morphological analysis and generation is linear in the length of the string. In practice, we can analyze about one hundred words per second using a contemporary workstation.

Evaluation of the analyzer requires a measure of:

- the precentage of tokens that receive the correct analysis in context (vs. the number of tokens that are not analyzed at all or lack the correct analysis in context)
- the precentage of wrong analyses (linguistically wrong, relardless of context)
- how many correct analyses have not been output for tokens

These measures can be performed using a reliable broad-coverage morphological analyzer, or by having a human informant annotate a text manually. The former option was not possible as we have no access to such a tool. The latter option is very hard and demanding and can be done on relatively small texts. We have manually annotated 1024 tokens (from a total of 2312 tokens) from a newspaper article [1]. 925 of the tokens (90.332%) received the correct analysis in context, leaving 99 tokens (9.668%) with either no analysis at all or without the correct analysis in context. 7 tokens (0.6836%) had an incorrect analysis among the analyses. There was no attempt at measuring how many of the tokens received all possible correct analyses. The vast majority of unanalyzed tokens were proper names that were not listed in the lexicon. Then there were collocations, such as, תל אביב, הקיבוץ המאוחד, ובעלי חיים, מבית־מדרשם, לבני־אדם, הקדוש־ברוך־הוא, and several Aramic expressions, such as, בתולא־שפירתא. Some of the collocations were proper names. Then there were words that have a different spelling than the one listed in the lexicon, such as, לראיין vs. לראין. Then there were a few nouns and adjectives that were missing from the lexicon. Then there were mis-spelling, such as י"ם as an abbreviation for ירושלים, or בניי instead of בני. There were also a few cases of unrecognized punctuation or words that contained vowels, which were not recognized by the analyzer.

We have also extracted the most frequent tokens that have not been assigned any analysis by the morphological analyzer using large corpora [2] and realized that almost all of these cases result in proper names missing from the lexicon or alternative spellings (that do not conform to a standard) of words that are covered by the lexicon. Applying Named Entity Recognition techniques can help dealing with the peoper names coverage, whereas alternative spellings remain a problem at this point.

Following are some output examples that demonstrate the morphological analyzer. Figures 5.1, 5.2, 5.3 and 5.4 demonstrate the output of the words כשמהראש, שבתה, תסתלק and מאחר, respectively.

```
    [+temporalSubConj]כש[+preposition]מ[+definiteArticle]ה
[+noun][+id]12328[+undotted]ראש[+transliterated]rae
[+gender]+masculine[+number]+singular[+script]+formal[+construct]+false
```

Figure 5.1: Analyses for כשמהראש

[1] see: http://www.ynet.co.il/articles/0,7340,L-3023878,00.html taken from the books section of YNET

[2] over twenty three million tokens in total; see: http://mila.cs.technion.ac.il/website/english/resources/corpora/index.html

The morphological analyzer is wrapped by a script that allows it to receive tokenized text in XML and output a morphologically annotated representation (also in XML). The XML representation facilitates different *views* of the data, for example, a *user friendly* representation of the output. A demonstration of these capabilities is available on-line at `http://cl.haifa.ac.il/projects/hebmorph/fsma.html`.

The analyzer is now used for a number of projects, including as a front end for a Hebrew to English machine translation system (Lavie et al., 2004). It is routinely tested on a variety of texts, and tokens with zero analyses are being inspected manually. Other than that, maintenance consists only of fixing bug reports. We are able to automate only the process of determining which tokens in a given text did not get any analysis, whereas ensuring that all and only correct analyses are generated can only be performed manually, and therefore the ability to apply these tests on large amounts of text is limited.

This work can be extended in a number of directions. While the set of rules is relatively stable, the lexicon must be constantly expanded (mostly with proper names). Automatic expansion of the lexicon using machine learning methods is an attractive direction due to the time intensive and costly process of manually performing lexicographic work. The lexicon can also be further augmented to include definitions and/or translations. Work is currently underway for incorporating Hebrew-English word translations into the lexicon.

The morphological analyzer produces all the possible analyses of each word independently of its context. A natural continuation of our work would be a module for disambiguation. Existing works on part-of-speech tagging and morphological disambiguation in Hebrew (Segal, 2000; Adler, 2001; Bar-Haim, 2005) leave much room for further research. Incorporating state-of-the-art machine learning techniques for morphological disambiguation to the output produced by the analyzer will generate an optimal system which is broad-coverage, effective and accurate. Such work is expected to begin in the near future.

```
    [+noun] [+id]13251[+undotted]שבת[+transliterated]ebt
[+gender]+feminine[+number]+singular[+script]+formal[+construct]+false
[+possessiveSuffix]+3p/F/Sg
    [+verb] [+id]7446[+undotted]שבה[+transliterated]ebh
[+root]שבה[+binyan]+Pa'al[+person/gender/number]+3p/F/Sg[+script]+formal
[+tense]+past
    [+verb] [+id]17274[+undotted]שבת[+transliterated]ebt
[+root]שבת[+binyan]+Pa'al[+person/gender/number]+3p/F/Sg[+script]+formal
[+tense]+past
    [+relativizer/subordinatingConjunction]ש[+preposition]ב
[+noun] [+id]18422[+undotted]תה[+transliterated]th
[+gender]+masculine[+number]+singular[+script]+formal[+construct]+true
    [+relativizer/subordinatingConjunction]ש[+preposition]ב
[+noun] [+id]18422[+undotted]תה[+transliterated]th
[+gender]+masculine[+number]+singular[+script]+formal[+construct]+false
    [+relativizer/subordinatingConjunction]ש[+preposition]ב[+definiteArticle]
[+noun] [+id]18422[+undotted]תה[+transliterated]th
[+gender]+masculine[+number]+singular[+script]+formal[+construct]+false
    [+relativizer/subordinatingConjunction]ש
[+noun] [+id]3320[+undotted]בתה[+transliterated]bth
[+gender]+feminine[+number]+singular[+script]+formal[+construct]+false
    [+relativizer/subordinatingConjunction]ש
[+noun] [+id]17868[+undotted]בת[+transliterated]bt
[+gender]+feminine[+number]+singular[+script]+formal[+construct]+false
[+possessiveSuffix]+3p/F/Sg
```

Figure 5.2: Analyses for **שבתה**

[+verb] [+id] 4355 [+undotted] הסתלק [+transliterated] hstlq
[+root] סלק [+binyan] +Hitpa'el [+person/gender/number] +2p/M/Sg [+script] +formal
[+tense] +future

[+verb] [+id] 4355 [+undotted] הסתלק [+transliterated] hstlq
[+root] סלק [+binyan] +Hitpa'el [+person/gender/number] +3p/F/Sg [+script] +formal
[+tense] +future

Figure 5.3: Analyses for תסתלק

[+verb] [+id] 5147 [+undotted] איחר [+transliterated] aixr
[+root] ארח [+binyan] +Pi'el [+person/gender/number] +123p/M/Sg [+script] +formal
[+tense] +beinoni

[+preposition] [+id] 826 [+undotted] מאחר [+transliterated] maxr [+script] +formal

[+conjunction] [+id] 15745 [+undotted] מאחר [+transliterated] maxr
[+type] +coordinating [+script] +formal

[+preposition] מ [+adjective] [+id] 4407 [+undotted] אחר [+transliterated] axr
[+gender] +masculine [+number] +singular [+script] +formal [+construct] +false

[+preposition] מ [+adjective] [+id] 4407 [+undotted] אחר [+transliterated] axr
[+gender] +masculine [+number] +singular [+script] +formal [+construct] +true

Figure 5.4: Analyses for מאחר

## References

Adler, Meni. 2001. Hidden markov model for hebrew part-of-speech tagging. Master's thesis, Ben Gurion University, Israel.

Alon, Emmanuel. 1995. *Unvocalized Hebrew Writing: The Structure of Hebrew Words in Syntactic Context.* Ben-Gurion University of the Negev Press. In Hebrew.

Attar, R., Y. Choueka, N. Dershowitz, and A. S. Fraenkel. 1978. KEDMA - linguistic tools for retrieval systems. *Journal of the Association for Computing Machinery*, 25(1):52–66, January.

Bar-Haim, Roy. 2005. Part-of-speech tagging for hebrew and other semitic languages. Master's thesis, Computer Science Department, Technion, Haifa, Israel.

Barkali, Shaul. 2000. *luax ha-pe&lim ha-shalem.* Rubin Mass Ltd., Jerusalem, 51 edition. In Hebrew.

Beesley, Kenneth R. 1996. Arabic finite-state morphological analysis and generation. In *Proceedings of COLING-96, the 16th International Conference on Computational Linguistics*, Copenhagen.

Beesley, Kenneth R. 1998a. Arabic morphological analysis on the internet. In *Proceedings of the 6th International Conference and Exhibition on Multi-lingual Computing*, Cambridge, April.

Beesley, Kenneth R. 1998b. Arabic morphology using only finite-state operations. In Michael Rosner, editor, *Proceedings of the Workshop on Computational Approaches to Semitic languages*, pages 50–57, Montreal, Quebec, August. COLING-ACL'98.

Beesley, Kenneth R. 2001. Finite-state morphological analysis and generation of Arabic at Xerox Research: Status and plans in 2001. In *ACL Workshop on Arabic Language Processing: Status and Perspective.*, pages 1–8, Toulouse, France, July.

Beesley, Kenneth R. and Lauri Karttunen. 2000. Finite-state non-concatenative morphotactics. In *Proceedings of the fifth workshop of the ACL special interest group in computational phonology, SIGPHON-2000*, Luxembourg, August.

Beesley, Kenneth R. and Lauri Karttunen. 2003. *Finite-State Morphology.* CSLI.

Bentur, Esther, Aviella Angel, and Danit Segev. 1992. Computerized analysis of Hebrew words. *Hebrew Linguistics*, 36:33–38, December. In Hebrew.

Bentur, Esther, Aviella Angel, Danit Segev, and Alon Lavie. 1992. Analysis and generation of the nouns inflection in Hebrew. In Uzzi Ornan, Gideon Arieli, and Edit Doron, editors, *Hebrew Computational Linguistics*. Ministry of Science and Technology, chapter 3, pages 36–38. In Hebrew.

Choueka, Yaacov. 1966. Computers and grammar: mechnical analysis of Hebrew verbs. In *Proceedings of the annual conference of the Israeli Association for Information Processing*, pages 49–66, Rehovot. In Hebrew.

Choueka, Yaacov. 1990. MLIM - a system for full, exact, on-line grammatical analysis of Modern Hebrew. In Yehuda Eizenberg, editor, *Proceedings of the Annual Conference on Computers in Education*, page 63, Tel Aviv, April. In Hebrew.

Choueka, Yaacov. 1993. Response to “Computerized analysis of Hebrew words”. *Hebrew Linguistics*, 37:87, December. in Hebrew.

Cohen, Haim A. 1996. klalei ha-ktiv xasar ha-niqqud. *leshonenu la&am*, special edition, May. In Hebrew.

Kataja, Laura and Kimmo Koskenniemi. 1988. Finite-state description of Semitic morphology: A case study of Ancient Akkadian. In *COLING*, pages 313–315.

Kiraz, George Anton. 2000. Multitiered nonlinear morphology using multitape finite automata: a case study on Syriac and Arabic. *Computational Linguistics*, 26(1):77–105, March.

Kleene, S. C. 1956. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press.

Koskenniemi, Kimmo. 1983. *Two-Level Morphology: a general computational model for word-form recognition and production*. The department of general linguistics, University of Helsinki.

Lavie, Alon. 1989. Two-level morphology for Hebrew. Master's thesis, Technion, Haifa, Israel. In Hebrew.

Lavie, Alon, Alon Itai, Uzzi Ornan, and Mori Rimon. 1988. On the applicability of two-level morphology to the inflection of Hebrew verbs. In *Proceedings of the International Conference of the ALLC*, Jerusalem, Israel.

Lavie, Alon, Shuly Wintner, Yaniv Eytani, Erik Peterson, and Katharina Probst. 2004. Rapid prototyping of a transfer-based Hebrew-to-English machine translation system. In *Proceedings of TMI-2004: The 10th International Conference on Theoretical and Methodological Issues in Machine Translation*, Baltimore, MD, October.

Ornan, Uzzi. 1986. Phonemic script: A central vehicle for processing natural language – the case of Hebrew. Technical Report 88.181, IBM Research Center, Haifa, Israel.

Ornan, Uzzi. 1994. Basic concepts in "Romanization" of scripts. Technical Report LCL 94-5, Laboratory for Computational Linguistics, Technion, Haifa, Israel, March.

Ornan, Uzzi. 2003. *ha-mila ha-axarona*. Haifa university. In Hebrew.

Ornan, Uzzi and Michael Katz. 1995. A new program for Hebrew index based on the Phonemic Script. Technical Report LCL 94-7, Laboratory for Computational Linguistics, Technion, Haifa, Israel, July.

Schwarzwald, Ora. 2001. *Moden Hebrew*, volume 127 of *Languages of the World/Materials*. LINCOM EUROPA.

Schwarzwald, Ora (Rodrigu). 2002. *Studies in Hebrew Morphology.* The Open University of Israel.

Segal, Erel. 1997. Morphological analyzer for undotted Hebrew words. Unpublished manuscript.

Segal, Erel. 2000. Hebrew morphological analyzer for hebrew undotted texts. Master's thesis, Computer Science Department, Technion, Haifa, Israel.

Shapira, Meir and Yaacov Choueka. 1964. Mechanographic analysis of Hebrew morphology: possibilities and achievements. *Leshonenu*, 28(4):354–372. In Hebrew.

Wintner, Shuly. 2004. Hebrew computational linguistics: past and future. *Artificial Intelligence Review*, 21:113–138.

Zdaka, Itzhak. 1974. *luxot ha-po&al.* Kiryath Sepher Ltd., Jerusalem. In Hebrew.