Learning to Identify Semitic Roots

Ezra Daya

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE MASTER DEGREE

University of Haifa

Faculty of Social Sciences

Department of Computer Science

November, 2005

Learning to Identify Semitic Roots

By: Ezra Daya

Supervised By: Dr. Shuly Wintner

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE MASTER DEGREE

University of Haifa

Faculty of Social Sciences

Department of Computer Science

November, 2005

Approved by: _____ Date:_____

(supervisor)

Approved by: _____ Date:_____

(Chairman of M.A. Committee)

Acknowledgments

My foremost thank goes to my thesis advisor Dr. Shuly Wintner for his encouragement, support, and invaluable guidance that made this work possible.

I would like to thank Prof. Dan Roth for his insights and suggestions on various aspects through this work.

I would also like to thank The Caesarea Edmond Benjamin de Rothschild Foundation Institute for Interdisciplinary Applications of Computer Science for their financial support.

Last but not least, I would like to express my deepest love and gratitude to my wife, Miriam, for her endless support and understanding during the course of this thesis.

Contents

Learning to Identify Semitic Roots

Ezra Daya

Abstract

The standard account of word-formation processes in Semitic languages describes words as combinations of two morphemes: a *root* and a *pattern*. The root consists of consonants only, by default three (although longer roots are known), called *radicals*. The pattern is a combination of vowels and, possibly, consonants too, with 'slots' into which the root consonants can be 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.

Identifying the root of a given word is an important task. Although existing morphological analyzers for Hebrew only provide a lexeme (which is a combination of a root and a pattern), for other Semitic languages, notably Arabic, the root is an essential part of any morphological analysis simply because traditional dictionaries are organized by root, rather than by lexeme. Furthermore, roots are known to carry some meaning, albeit vague. We believe that this information can be useful for computational applications.

We present a machine learning approach, augmented by limited linguistic knowledge, to the problem of identifying the roots of Semitic words. To the best of our knowledge, this is the first application of machine learning to this problem. While there exist programs which can extract the root of words in Arabic and Hebrew, they are all dependent on labor intensive construction of large-scale lexicons which are components of full-scale morphological analyzers. The challenge of our work is to automate this process, avoiding the bottleneck of having to laboriously list the root and pattern of each lexeme in the language, and thereby gain insights that can be used for more detailed morphological analysis of Semitic languages.

1 Introduction

1.1 Linguistic background

The standard account of word-formation processes in Semitic languages describes words as combinations of two morphemes: a *root* and a *pattern*.[1] The root consists of consonants only, by default three (although longer roots are known), called *radicals*. The pattern is a combination of vowels and, possibly, consonants too, with 'slots' into which the root consonants can be 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. See Shimron (2003) for a survey.

As an example of root-and-pattern morphology, consider the Hebrew[2] roots *g.d.l*, *k.t.b* and *r.$.m* and the patterns *haCCaCa*, *hitCaCCut* and *miCCaC*, where the 'C's indicate the slots. When the roots combine with these patterns the resulting lexemes are *hagdala*, *hitgadlut*, *migdal*, *haktaba*, *hitkatbut*, *miktab*, *har$ama*, *hitra$mut*, *mir$am*, respectively. After the root combines with the pattern, some morpho-phonological alternations take place, which may be non-trivial: for example, the *hitCaCCut* pattern triggers assimilation when the first consonant of the root is *t* or *d*: thus, *d.r.$+hitCaCCut* yields *hiddar$ut*. The same pattern triggers metathesis when the first radical is *s* or *$*: *s.d.r+hitCaCCut* yields *histadrut* rather than the expected **hitsadrut*. Semi-vowels such as *w* or *y* in the root are frequently combined with the vowels of the pattern, so that *q.w.m+haCCaCa* yields *haqama*, etc. Frequently, root consonants such as *w* or *y* are altogether missing from the resulting form.

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 *w* is used both for the vowels *o* and *u* and for the consonant *v*, whereas *i* is similarly used both for the vowel *i* and for the consonant

[1] An additional morpheme, *vocalization*, is used to abstract the pattern further; for the present purposes, this distinction is irrelevant.

[2] To facilitate readability we use a transliteration of Hebrew using ASCII characters in this document:

'	b	g	d	h	w	z	x	v	i	k	l	m	n	s	y	p	c	q	r	$	t
א	ב	ג	ד	ה	ו	ז	ח	ט	י	כ	ל	מ	נ	ס	ע	פ	צ	ק	ר	ש	ת

y. On top of that, the script dictates that many particles, including four of the most frequent prepositions, the definite article, the coordinating conjunction and some subordinating conjunctions all attach to the words which immediately follow them. Thus, a form such as *mhgr* can be read as a lexeme ("immigrant"), as *m-hgr* "from Hagar"or even as *m-h-gr* "from the foreigner". Note that there is no deterministic way to tell whether the first *m* of the form is part of the pattern, the root or a prefixing particle (the preposition *m* "from").

The Hebrew script has 22 letters, all of which can be considered consonants. The number of tri-consonantal roots is thus theoretically bounded by 22^3, although several phonological constraints limit this number to a much smaller value. For example, while roots whose second and third radicals are identical abound in Semitic languages, roots whose first and second radicals are identical are extremely rare (see McCarthy (1981) for a theoretical explanation). To estimate the number of roots in Hebrew we compiled a list of roots from two sources: a dictionary (Even-Shoshan, 1993) and the verb paradigm tables of Zdaqa (1974). The union of these yields a list of 2152 roots.[3]

While most Hebrew roots are regular, many belong to *weak paradigms*, which means that root consonants undergo changes in some patterns. Examples include *i* or *n* as the first root consonant, *w* or *i* as the second, *i* as the third and roots whose second and third consonants are identical. For example, consider the pattern *hCCCh*. Regular roots such as *p.s.q* yield forms such as *hpsqh*. However, the irregular roots *n.p.l*, *i.c.g*, *q.w.m* and *g.n.n* in this pattern yield the seemingly similar forms *hplh*, *hcgh*, *hqmh* and *hgnh*, respectively. Note that in the first and second examples, the first radical (*n* or *i*) is missing, in the third the second radical (*w*) is omitted and in the last example one of the two identical radicals is omitted. Consequently, a form such as hC_1C_2h can have any of the roots $n.C_1.C_2$, $C_1.w.C_2$, $C_1.i.C_2$, $C_1.C_2.C_2$ and even, in some cases, $i.C_1.C_2$.

While the Hebrew script is highly ambiguous, ambiguity is somewhat reduced for the task we consider here, as many of the possible lexemes of a given form share the same root. Still, in order to correctly identify the root of a given word, context must be taken into consideration. For example, the form *$mnh* has more than a dozen readings, including the adjective "fat" (feminine singular), which has the root *$.m.n*, and the

[3]Only tri-consonantal roots are counted. Ornan (2003) mentions 3407 roots, whereas the number of roots in Arabic is estimated to be 10,000 (Darwish, 2002).

verb “count”, whose root is *m.n.i*, preceded by a subordinating conjunction. In the experiments we describe below we ignore context completely, so our results are handicapped by design.

1.2 Machine learning techniques in natural language processing

There is a dramatic increase in research on machine learning methods in natural language processing (NLP) in the last years, using different learning methods to automatically extract linguistic knowledge from natural language corpora which replace or augment those constructed by the linguist. Traditional machine learning uses corpora as data for training and testing, applying mainly statistical and rule-based methods and sometimes a hybrid between the two. Many problems in natural language processing require learning of a mapping from one discrete structure to another. Examples are parsing (learning a function from strings to trees); named entity extraction (learning a function from strings to an underlying segmentation identifying people, places, locations and so on); and machine translation (learning a function from sentences in a source language to sentences in a target language). A very common approach to these tasks is to use some kind of generative model, for example a hidden Markov model, or history-based models. These approaches are probabilistic, in that they attempt to model a joint or conditional probability distribution over possible structures. There are several challenges in applying these methods to natural language, since NLP problems involve rich structures, and feature spaces are often very high dimensional and very sparse (Collins, 2003).

In many machine learning applications it is necessary to make decisions that depend on the outcomes of several different classifiers in a way that provides a coherent inference that satisfies some constraints: the sequential nature of the data or other domain specific constraints. Consider, for example, the problem of chucking natural language sentences where the goal is to identify several kinds of phrases (e.g., noun phrases and verb phrases) in sentences. A task of this sort involves multiple predictions that interact in some way. One way to address the problem is to use two classifiers for each type of phrases, one of which recognizes the beginning of the phrase, and the other its end. Clearly, there are constraints over the predictions; for instance, phrases cannot overlap and there may also be probabilistic constraints over the order of phrases and over their lengths. The goal is to maximize some global measure of accuracy, not necessarily to maximize the performance of each individual classifier involved in the decision.

Moreover, this method is also used in complex morphological problems involving a large number of targets, such as morphological disambiguation of Hebrew. Morphological analysis is a crucial stage in a variety of natural language processing applications. Due to the complex structure of the Hebrew morphology, several optional analyses can be provided for each word, and a disambiguation process is needed. Given the large number of potential targets, the problem can be addressed by combining several classifiers, each predicting the value of one component of the analysis. The results of the naïve classifiers can be combined in a sophisticated manner, taking into account the constraints that hold among the various components.

In this work we apply several machine learning techniques to the problem of root identification. In all the experiments described in this paper we use SNoW (Roth, 1998) as the learning environment, with *winnow* as the update rule (using *perceptron* yielded comparable results). SNoW is a multi-class classifier that is specifically tailored for learning in domains in which the potential number of information sources (features) taking part in decisions is very large, of which NLP is a principal example. It works by learning a sparse network of linear functions over a pre-defined or incrementally learned feature space. SNoW has already been used successfully as the learning vehicle in a large collection of natural language related tasks, including POS tagging, shallow parsing, information extraction tasks, etc., and compared favorably with other classifiers (Roth, 1998; Punyakanok and Roth, 2001; Florian, 2002). Typically, SNoW is used as a classifier, and predicts using a winner-take-all mechanism over the activation values of the target classes. However, in addition to the prediction, it provides a reliable confidence level in the prediction, which enables its use in an inference algorithm that combines predictors to produce a coherent inference.

1.3 Research objectives

Identifying the root of a given word is an important task. Although existing morphological analyzers for Hebrew only provide a lexeme (which is a combination of a root and a pattern), for other Semitic languages, notably Arabic, the root is an essential part of any morphological analysis simply because traditional dictionaries are organized by root, rather than by lexeme. Furthermore, roots are known to carry some meaning, albeit vague. We believe that this information can be useful for computational applications.

We present a machine learning approach, augmented by limited linguistic knowledge, to the problem

of identifying the roots of Semitic words. To the best of our knowledge, this is the first application of machine learning to this problem. While there exist programs which can extract the root of words in Arabic (Beesley, 1998a; Beesley, 1998b) and Hebrew (Choueka, 1990), they are all dependent on labor intensive construction of large-scale lexicons which are components of full-scale morphological analyzers. Note that Buckwalter (2002)'s Arabic morphological analyzer only uses "word stems – rather than root and pattern morphemes – to identify lexical items. (The information on root and pattern morphemes could be added to each stem entry if this were desired.)" The challenge of our work is to automate this process, avoiding the bottleneck of having to laboriously list the root and pattern of each lexeme in the language, and thereby gain insights that can be used for more detailed morphological analysis of Semitic languages. As we show presently, identifying roots is a non-trivial problem even for humans, due to the complex nature of Semitic derivational and inflectional morphology and the peculiarities of the orthography.

We focus on Hebrew in the first part of this paper. We discuss in section 2 a simple, baseline, learning approach and then propose several methods for combining the results of interdependent classifiers (section 3). Then, the same technique is applied to Arabic in section 4 and we demonstrate comparable improvements. In section 5 we discuss the influence of global constraints on local classifiers. We conclude with suggestions for future research.

Previous versions of this work were published as Daya, Roth, and Wintner (2004) and Daya, Roth, and Wintner (Forthcoming).

2 Naïve classification methods

2.1 Data and methodology

We take a machine learning approach to the problem of determining the root of a given word. For training and testing, a Hebrew linguist manually tagged a corpus of 15,000 words (a set of newspaper articles). Of these, only 9752 were annotated; the reason for the gap is that some Hebrew words, mainly borrowed but also some frequent words such as prepositions, do not have roots; we further eliminated 168 roots with more than three consonants and were left with 5242 annotated word types, exhibiting 1043 different roots. Table 1 shows the distribution of word types according to root ambiguity.

Number of roots	1	2	3	4
Number of words	4886	335	18	3

Table 1: Root ambiguity in the corpus

Table 2 provides the distribution of the roots of the 5242 word types in our corpus according to root type, where R_i is the i-th radical (note that some roots may belong to more than one group).

Paradigm	Number	Percentage
$R_1 = i$	414	7.90%
$R_1 = w$	28	0.53%
$R_1 = n$	419	7.99%
$R_2 = i$	297	5.66%
$R_2 = w$	517	9.86%
$R_3 = h$	18	0.19%
$R_3 = i$	677	12.92%
$R_2 = R_3$	445	8.49%
Regular	3061	58.41%

Table 2: Distribution of root paradigms

As assurance for statistical reliability, in all the experiments discussed in the sequel (unless otherwise mentioned) we performed 10-fold cross validation runs for every classification task during evaluation. We also divided the test corpus into two sets: a *development set* of 4800 words and a *held-out set* of 442 words. Only the development set was used for parameter tuning. A given *example* is a word type with all its (manually tagged) possible roots. In the experiments we describe below, our system produces one or more root *candidates* for each example. For each example, we define *tp* as the number of candidates correctly produced by the system; *fp* as the number of candidates which are not correct roots; and *fn* as the number of correct roots the system did not produce. As usual, we define *recall* as $\frac{tp}{tp+fp}$ and *precision* as $\frac{tp}{tp+fn}$; we then compute f-measure for each example (with $\alpha = 0.5$) and (macro-) average to obtain the system's overall f-measure.

To estimate the difficulty of this task, we asked six human subjects to perform it. Subjects were asked to identify all the possible roots of all the words in a list of 200 words (without context), randomly chosen from the test corpus. All subjects were computer science graduates, native Hebrew speakers with no linguistic background. The average precision of humans on this task is 83.52%, and with recall at 80.27%, f-measure is 81.86%. Two main reasons for the low performance of humans are the lack of context and the ambiguity of some of the weak paradigms.

2.2 Direct prediction

To establish a baseline, we first performed two experiments with simple, baseline classifiers. All the experiments we describe in this work share the same features and differ only in the target classifiers. The features that are used to characterize a word are both grammatical and statistical:

- Location of letters (e.g., the third letter of the word is *b*). We limit word length to 20, thus obtaining 440 features of this type (recall that the size of the alphabet is 22).

- Bigrams of letters, independently of their location (e.g., the substring *gd* occurs in the word). This yields 484 features.

- Prefixes (e.g., the word is prefixed by *k$h* "when the"). We have 292 features of this type, corre-

sponding to 17 prefixes and sequences thereof.

- Suffixes (e.g., the word ends with *im*, a plural suffix). There are 26 such features.

In the first of the two experiments, referred to as Experiment A, we trained a classifier to learn roots as a single unit. The two obvious drawbacks of this approach are the large set of targets and the sparseness of the training data. Of course, defining a multi-class classification task with 2152 targets, when only half of them are manifested in the training corpus, does not leave much hope for ever learning to identify the missing targets.

In Experiment A, the macro-average precision of ten-fold cross validation runs of this classification problem is 45.72%; recall is 44.37%, yielding an f-score of 45.03%. In order to demonstrate the inadequacy of this method, we repeated the same experiment with a different organization of the training data. We chose 30 roots and collected all their occurrences in the corpus into a test file. We then trained the classifier on the remainder of the corpus and tested on the test file. As expected, the accuracy was close to 0%.

2.3 Decoupling the problem

In the second experiment, referred to as Experiment B, we separated the problem into three different tasks. We trained three classifiers to learn each of the root consonants in isolation and then combined the results in the straight-forward way (a conjunction of the decisions of the three classifiers). This is still a multi-class classification but the number of targets in every classification task is only 22 (the number of letters in the Hebrew alphabet) and data sparseness is no longer a problem. As we show below, each classifier achieves much better generalization, but the clear limitation of this method is that it completely ignores interdependencies between different targets: the decision on the first radical is completely independent of the decision on the second and the third.

We observed a difference between recognizing the first and third radicals and recognizing the second one, as can be seen in table 3. These results correspond well to our linguistic intuitions: the most difficult cases for humans are those in which the second radical is *w* or *i*, and those where the second and the third consonants are identical. Combining the three classifiers using logical conjunction yields an f-measure of

52.84%. Here, repeating the same experiment with the organization of the corpus such that testing is done on unseen roots yielded 18.1% accuracy.

	R_1	R_2	R_3	root
Precision:	82.25	72.29	81.85	53.60
Recall:	80.13	70.00	80.51	52.09
f-measure:	81.17	71.13	81.18	52.84

Table 3: Accuracy of SNoW's identifying the correct radical

To demonstrate the difficulty of the problem, we conducted yet another experiment. Here, we trained the system as above but we tested it on different words whose roots were known to be in the training set. The results of experiment A here were 46.35%, whereas experiment B was accurate in 57.66% of the cases. Evidently, even when testing only on previously seen roots, both naïve methods are unsuccessful.

3 Combining interdependent classifiers

Evidently, simple combination of the results of the three classifiers leaves much room for improvement. Therefore we explore other ways for combining these results. We can rely on the fact that SNoW provides insight into the decisions of the classifiers – it lists not only the selected target, but rather all candidates, with an associated confidence measure. Apparently, the correct radicals are chosen among SNoW's top-n candidates with high accuracy. In table 4, we present the recall of the correct root radicals among top-n candidates.

	R_1	R_2	R_3
top-1	80.13	70.00	80.51
top-2:	91.83	85.94	92.08
top-5:	97.78	96.52	98.09
top-10:	99.46	99.30	99.60

Table 4: Recall of SNoWs identifying the correct radical among top-n candidates

This observation calls for a different way of combining the results of the classifiers which takes into account not only the first candidate but also others, along with their confidence scores.

3.1 HMM combination

We considered several ways, e.g., via HMMs, of appealing to the sequential nature of the task (R_1 followed by R_2, followed by R_3). Not surprisingly, direct applications of HMMs are too weak to provide satisfactory results, as suggested by the following discussion. The approach we eventually opted for combines the predictive power of a classifier to estimate more accurate state probabilities.

Given the sequential nature of the data and the fact that our classifier returns a distribution over the possible outcomes for each radical, a natural approach is to combine SNoW's outcomes via a Markovian approach. Variations of this approach are used in the context of several NLP problems, including POS tagging (Schütze and Singer, 1994), shallow parsing (Punyakanok and Roth, 2001) and named entity recog-

nition (Tjong Kim Sang and De Meulder, 2003).

Formally, we assume that the confidence supplied by the classifier is the probability of a state (radical, r) given the observation o (the word), $P(r|o)$. This information can be used in the HMM framework by applying Bayes rule to compute

$$P(o|r) = \frac{P(r|o)P(o)}{P(r)},$$

where $P(o)$ and $P(r)$ are the probabilities of observing o and being at r, respectively. That is, instead of estimating the observation probability $P(o|r)$ directly from training data, we compute it from the classifiers' output. Omitting details (see Punyakanok and Roth (2001)), we can now combine the predictions of the classifiers by finding the most likely root for a given observation, as

$$r = argmax P(r_1 r_2 r_3 | o, \theta)$$

where θ is a Markov model that, in this case, can be easily learned from the supervised data. Clearly, given the short root and the relatively small number of values of r_i that are supported by the outcomes of SNoW, there is no need to use dynamic programming here and a direct computation is possible.

However, perhaps not surprisingly given the difficulty of the problem, this model turns out to be too simplistic. In fact, performance deteriorated to an f-score of 37.79%. We conjecture that the static probabilities (the model) are too biased and cause the system to abandon good choices obtained from SNoW in favor of worse candidates whose global behavior is better.

For example, the root *&.b.d* was correctly generated by SNoW as the best candidate for the word *&wbdim*, but since $P(R_3 = b|R_2 = b)$, which is 0.1, is higher than $P(R_3 = d|R_2 = b)$, which is 0.04, the root *&.b.b* was produced instead. Note that in the above example the root *&.b.b* cannot possibly be the correct root of *&wbdim* since no pattern in Hebrew contains the letter *d*, which must therefore be part of the root. It is this kind of observations that motivate the addition of linguistic knowledge as a vehicle for combining the results of the classifiers. An alternative approach, which we investigate in section 3.4 , is the introduction of higher-level classifiers which take into account interactions between the radicals (Punyakanok and Roth, 2001).

3.2 Adding linguistic constraints

The experiments discussed previously are completely devoid of linguistic knowledge. In particular, experiment B inherently assumes that any sequence of three consonants can be the root of a given word. This is obviously not the case: with very few exceptions, all radicals must be present in any inflected form (in fact, only *w, i, n* and in an exceptional case *l* can be deleted when roots combine with patterns). We therefore trained the classifiers to consider as targets only letters that occurred in the observed word, plus *w, i, n* and *l*, rather than any of the alphabet letters. The average number of targets is now 7.2 for the first radical, 5.7 for the second and 5.2 for the third (compared to 22 each in the previous setup).

In this model, known as the *sequential model* (Even-Zohar and Roth, 2001), SNoW's performance improved slightly, as can be seen in table 5 (compare to table 4). Combining the results in the straightforward way yields an f-measure of 58.89%, a small improvement over the 52.84% performance of the basic method. This new result should be considered baseline. In what follows we always employ the sequential model for training and testing the classifiers, using the same constraints. However, we employ more linguistic knowledge for a more sophisticated combination of the classifiers.

	R_1	R_2	R_3	root
Precision:	83.06	72.52	83.88	59.83
Recall:	80.88	70.20	82.50	57.98
f-measure:	81.96	71.34	83.18	58.89

Table 5: Accuracy of SNoW's identifying the correct radical, sequential model

3.3 Combining classifiers using linguistic knowledge

SNoW provides a ranking on all possible roots. We now describe the use of linguistic constraints to re-rank this list. We implemented a function, dubbed *the scoring function* below, which uses knowledge pertaining to word-formation processes in Hebrew in order to estimate the likelihood of a given candidate being the root of a given word. The function practically classifies the candidate roots into one of three classes: good candidates, which are likely to be the root of the word; bad candidates, which are highly unlikely; and

average cases.

Definition 1 *A root $r = r_1r_2r_3$ is*

- *in **Paradigm P1** if $r_1 \in \{w, i, n\}$;*
- *in **Paradigm P2** if $r_2 \in \{w, i\}$;*
- *in **paradigm P3** if $r_3 \in \{h, i\}$;*
- *in **paradigm P4** if $r_2 = r_3$;*
- ***regular** if none of the above holds.*

Given a word w and a candidate root $r = r_1r_2r_3$, the scoring function approximates the likelihood of r being the root of w by implementing the following constraints:

Constraint 1 *If r is **regular** then r_1, r_2, r_3 must occur in w in this order. Furthermore, either $r_1r_2r_3$ are consecutive in w, or a single letter intervenes between r_1 and r_2 or between r_2 and r_3 (or both). The intervening letter between r_1 and r_2 can only be w, i, t (if r_1 is $ or s), d (if r_1 is z) or v (if r_1 is c). The intervening letter between r_2 and r_3 can only be w or i.*

Constraint 2 *If r is in **Paradigm P1** and not in **P2**, **P3** or **P4** paradigms, then r_2, r_3 must occur in w in this order. Furthermore, either r_2r_3 are consecutive in w, or a single letter intervenes between r_2 and r_3 . The intervening letter can only be w or i.*

Constraint 3 *If r is in **Paradigm P2** and not in **P1** or **P3** paradigms, then r_1, r_3 must occur in w in this order. Furthermore, either r_1r_3 are consecutive in w, or a single letter intervenes between r_1 and r_3 . The intervening letter can only be w, i, t (if r_1 is $ or s), d (if r_1 is z) or v (if r_1 is c).*

Constraint 4 *If r is in **Paradigm P3** and not in **P1** or **P2** paradigms, then r_1, r_2 must occur in w in this order. Furthermore, either r_1r_2 are consecutive in w, or a single letter intervenes between r_1 and r_2 . The intervening letter can only be w, i, t (if r_1 is $ or s), d (if r_1 is z) or v (if r_1 is c).*

Constraint 5 *If r is in **Paradigm P4** and not in **P1** or **P2** paradigms, then r_1, r_2 must occur in w in this order. Furthermore, either $r_1 r_2$ are consecutive in w, or a single letter intervenes between r_1 and r_2 . The intervening letter can only be w, i, t (if r_1 is $ or s), d (if r_1 is z) or v (if r_1 is c).*

Constraint 6 *r must occur in the pre-compiled list of roots.*

The decision of the function is based on the observation that when a root is regular it either occurs in a word consecutively or with a certain single letter between any two of its radicals (constraint 1). The scoring function checks, given a root and a word, whether this is the case. If the condition holds, the scoring function returns a high value. The weak paradigms constrains (2 - 5) are assigned a middle score, since in such paradigms we are limited to a partial check on the root as we only check for the occurrence of two root radicals instead of three in the word. The scoring function does not treat roots that are in more than one paradigm, since these roots are very rare in Hebrew and it is very difficult to predict their behaviour in the word-formation process. In this case, the scoring function returns an average score as a default value. We also make use in this function of our pre-compiled list of roots. A root candidate which does not occur in the list (constraint 6) is assigned the low score.

The actual values that the function returns were chosen empirically by counting the number of occurrences of each class in the training data. Thus, "good" candidates make up 74.26% of the data, hence the value the function returns for "good" roots is set to 0.7426. Similarly, the middle value is set to 0.2416 and the low – to 0.0155.

As an example, consider *ki$lwn*, whose only possible root is *k.$.l.* Here, the correct candidate will be assigned the high score, since *k.$.l* is a regular root and its radicals occur consecutively in the word with a single intervening letter *i* between *k* and *$* (constraint 1). The candidate root *$.l.i* will be assigned a middle score, since this root is in paradigm P3 and constraint 4 holds. The candidate root *$.l.n* will score low, as it does not occur in the list of roots (constraint 6).

In addition to the scoring function we implemented a simple edit distance function which returns, for a given root and a given word, the inverse of the edit distance between the two. For example, for *hipltm*, the (correct) root *n.p.l* scores $1/4$ whereas *p.l.t* scores $1/3$.

We then run SNoW on the test data and rank the results of the three classifiers *globally*, where the order is determined by the product of the three different classifiers. This induces an order on *roots*, which are combinations of the decisions of three independent classifiers. Each candidate root is assigned three scores: the product of the confidence measures of the three classifiers; the result of the scoring function; and the inverse edit distance between the candidate and the observed word. We rank the candidates according to the product of the three scores (i.e., we give each score an equal weight in the final ranking).

In order to determine how many of the candidates to produce for each example, we experimented with two methods. First, the system produced the top-i candidates for a fixed value of i. The results on the development set are given in table 6.

$i =$	1	2	3	4
Precision	**82.02**	46.17	32.81	25.19
Recall	**79.10**	87.83	92.93	94.91
f-measure	**80.53**	60.52	48.50	39.81

Table 6: Performance of the system when producing top-i candidates.

Obviously, since most words have only one root, precision drops dramatically when the system produces more than one candidate. This calls for a better threshold, facilitating a non-fixed number of outputs for each example. We observed that in the "difficult" examples, the top ranking candidates are assigned close scores, whereas in the easier cases, the top candidate is usually scored much higher than the next one. We therefore decided to produce all those candidates whose scores are not much lower than the score of the top ranking candidate. The drop in the score, δ, was determined empirically on the development set. The results are listed in table 7, where δ varies from 0.1 to 1 (δ is actually computed on the log of the actual score, to avoid underflow).

These results show that choosing $\delta = 0.4$ produces the highest f-measure. With this value for δ, results for the held-out data are presented in table 8. The results clearly demonstrate the added benefit of the linguistic knowledge. In fact, our results are slightly better than average human performance, which we recall as well. Interestingly, even when testing the system on a set of roots which do *not* occur in the training

$\delta =$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Precision	81.81	80.97	79.93	**78.86**	77.31	75.48	73.71	71.80	69.98	67.90
Recall	81.06	82.74	84.03	**85.52**	86.49	87.61	88.72	89.70	90.59	91.45
f-measure	81.43	81.85	81.93	**82.06**	81.64	81.10	80.52	79.76	78.96	77.93

Table 7: Performance of the system, producing candidates scoring no more than δ below the top score.

corpus, we obtain an f-score of 65.60%. This result demonstrates the robustness of our method.

	Held-out data	Humans
Precision:	**80.90**	83.52
Recall:	**88.16**	80.27
f-measure:	**84.38**	81.86

Table 8: Results: performance of the system on held-out data.

It must be noted that the scoring function alone is *not* a function for extracting roots from Hebrew words. First, it only scores a given root candidate against a given word, rather than yield a root given a word. While we could have used it exhaustively on all possible roots in this case, in a general setting of a number of classifiers the number of classes might be too high for this solution to be practical. Second, the function only produces three different values; when given a number of candidate roots it may return more than one root with the highest score. In the extreme case, when called with all 22^3 potential roots, it returns on the average more than 11 candidates which score highest (and hence are ranked equally).

Similarly, the additional linguistic knowledge is not merely *eliminating* illegitimate roots from the ranking produced by SNoW. Using the linguistic constraints encoded in the scoring function only to eliminate roots, while maintaining the ranking proposed by SNoW, yields much lower accuracy. Specifically, when we use only the list of roots as the single constraint when combining the three classifiers, thereby implementing only a filter of infeasible results, we obtain a precision of 65.24%, recall of 73.87% and an f-measure of 69.29%. Clearly, our linguistically motivated scoring does more than elimination, and actually *re-ranks* the roots. We conclude that it is only the *combination* of the classifiers with the linguistically motivated

scoring function which boosts the performance on this task.

3.4 Sequential combination

In the previous methods, three classifiers for each radical were trained independently, disregarding the dependency between the root radicals. In Hebrew and Arabic, some letters cannot appear in a sequence, mostly due to phonetic restrictions. For example, if the first radical is *s*, the second radical cannot be *z*, *c* or *e*. Taking into account the dependency between the root radicals is an interesting learning experiment which may provide a better results. Of course, during training we know what the correct root is, and all we need is to consider the previous root radical. In other words, the classifier of the first radical does not require any change, and we only need to add one more feature to the feature set of each of the two other classifiers. While the training process is relatively simple, the testing process requires a more sophisticated routine.

Consider a specific example of some word *w*. We already trained a classifier for R_1 (the first root radical), so we can look at the predictions of the R_1 classifier for *w*. Assume that this classifier predicts $a_1, a_2, a_3, ..., a_k$ with confidence scores $c_1, c_2, ..., c_k$ respectively (the maximum value of k is 22). For each value a_i $(1 \leq i \leq k)$ predicted by the R_1 classifier, we run the R_2 classifier where the value of the feature for R_1 is a_i. That is, we run the R_2 classifier k times for each word *w* (k depends on *w*). Then, we check which value of i (where i runs from 1 to k) gives the best sum of the two classifiers, both the confidence measure of the R_1 classifier on a_i and the confidence measure of the R_2 classifier. This gives a confidence ranking for R_2. We perform the same evaluation for R_3, using the results of the R_2 classifier as the value of the R_3 added feature. We also checked the results of taking the best value of i disregarding the confidence score of the previous classifier, as shown in table 9 on the development set and in table 10 on the held-out data.

The results of this experiment are not better than the ones reported in section 3.3. We conjecture that the reason, similarly to many machine learning tasks in NLP, is data sparseness. While there are very frequent letter sequences, most root sequences occur rarely. Extending the annotated corpus would probably provide better results.

Development set	Regarding previous score	Disregarding previous score
Precision:	76.99%	76.87%
Recall:	84.78%	84.78%
f-measure:	80.7%	80.63%

Table 9: Results: Combining dependent classifiers - development set

Held-out data	Regarding previous score	Disregarding previous score
Precision:	77.11%	77.07%
Recall:	89.44%	89.66%
f-measure:	82.82%	82.89%

Table 10: Results: Combining dependent classifiers - held-out data

3.5 Learning bigrams

In the first experiments of this research, we trained a classifier to learn roots as a single unit. As already mentioned, the drawbacks of this approach are the large set of targets and the sparseness of the training data. Then, we decoupled the problem into three different classifiers to learn each of the root consonants in isolation and then combined the results in various ways. Training the classifiers in the *sequential model*, considering as targets only letters that occurred in the observed word, plus *w*, *i*, *n* and *l*, reduced the number of targets from 22 to approximately 7. This facilitates a different experiment whereby bigrams of the root radicals, rather than each radical in isolation, are learned, taking advantage of the reduced number of targets for each radical. On one hand, the average number of targets is still relatively large (about 50), but on the other, we only have to deal with a combination of two classifiers. In this method, each of the classifiers should predict two letters at once. For example, we define one classifier to learn the first and second radicals (R_1R_2), and a second classifier to learn the second and third radicals (R_2R_3). Alternatively, the first and third radicals can be learned as a single unit by a different classifier. In this case, we only need to combine this classifier with one of the above mentioned classifiers to obtain the complete root.

It should be noted that the number of potential roots for a given word example in combining three

different classifiers (for each of the root radicals) is determined by multiplying the number of targets of each of the classifiers. In this classification problem, each classifier predicts two root radicals; meaning that the classifiers overlap in one radical. This common radical should be identical in the combination (e.g., R_1R_2 and R_2R_3 overlap in R_2), and thus the number of potential roots is significantly reduced. The results of these experiments on the development set are reported in table 11 and on the held-out data in table 12.

Development set	$R_1R_2\&R_2R_3$	$R_1R_2\&R_1R_3$	$R_2R_3\&R_1R_3$
Precision:	82.11	79.71	79.11
Recall:	85.28	86.4	86.64
f-measure:	83.67	82.92	82.71

Table 11: Accuracy of combining classifiers of root radicals bigrams - development set.

Held-out data	$R_1R_2\&R_2R_3$	$R_1R_2\&R_1R_3$	$R_2R_3\&R_1R_3$
Precision:	84.61	80.95	80.82
Recall:	82.88	83.78	82.76
f-measure:	83.73	82.34	81.78

Table 12: Accuracy of combining classifiers of root radicals bigrams - held-out data.

As can be seen in table 11, these experiments yield results which are comparable to those of section 3.3. Reducing the number of classifiers seems to be crucial for the task of combining several interdependent classifiers. In this specific learning problem of roots, we can point out two main reasons for receiving these results: First, the sequential model significantly reduced the number of potential targets in each classifier (by almost 70%); second, overlapping classifiers reduced the potential number of root candidates. These factors may not be valid in similar learning problems.

3.6 Error analysis

Looking at the questionnaires filled in by our subjects (section 2.1), it is obvious that humans have problems identifying the correct roots in two general cases: when the root paradigm is weak (i.e., when the

root is irregular) and when the word can be read in more than one way and the subject chooses only one (presumably, the most prominent one). Our system suffers from similar problems: first, its performance on the regular paradigms is far superior to its overall performance; second, it sometimes cannot distinguish between several roots which are in principle possible, but only one of which happens to be the correct one.

To demonstrate the first point, we evaluated the performance of the system on a different organization of the data. We tested separately words whose roots are all regular, vs. words all of whose roots are irregular. We also tested words which have at least one regular root (mixed). The results are presented in table 13, and clearly demonstrate the difficulty of the system on the weak paradigms, compared to almost 95% on the easier, regular roots.

	Regular	Irregular	Mixed
Number of words	2598	2019	2781
Precision:	92.79	60.02	92.54
Recall:	96.92	73.45	94.28
f-measure:	94.81	66.06	93.40

Table 13: Error analysis: performance of the system on different cases.

A more refined analysis reveals differences between the various weak paradigms. Table 14 lists f-measure for words whose roots are irregular, classified by paradigm. As can be seen, the system has great difficulty in the cases of $R_2 = R_3$ and $R_3 = i$.

Paradigm	f-measure
$R_1 = i$	70.57
$R_1 = n$	71.97
$R_2 = i/w$	76.33
$R_3 = i$	58.00
$R_2 = R_3$	47.42

Table 14: Error analysis: the weak paradigms

Finally, we took a closer look at some of the errors, and in particular at cases where the system produces several roots where fewer (usually only one) are correct. Such cases include, for example, the word *hkwtrt* (“the title”), whose root is the regular *k.t.r*; but the system produces, in addition, also *w.t.r*, mistaking the *k* to be a prefix. This is the kind of errors which are most difficult to fix.

However, in many cases the system’s errors are relatively easy to overcome. Consider, for example, the word *hmtndbim* (“the volunteers”) whose root is the irregular *n.d.b*. Our system produces as many as five possible roots for this word: *n.d.b*, *i.t.d*, *d.w.b*, *i.h.d*, *i.d.d*. Clearly some of these could be eliminated. For example, *i.t.d* should not be produced, because if this were the root, nothing could explain the presence of the *b* in the word; *i.h.d* should be excluded because of the location of the *h*. Similar phenomena abound in the errors the system makes; they indicate that a more careful design of the scoring function can yield still better results, and this is a direction we intend to pursue in the future.

4 Extension to Arabic

Although Arabic and Hebrew have a very similar morphological system, being both semitic languages, the task of learning roots in Arabic is more difficult than in Hebrew, for the following reasons:

- There are 28 letters in Arabic which are represented using approximately 40 characters in Buckwalter (2002)'s transliteration of Modern Standard Arabic orthography. Thus, the learning problem is more complicated due the increased number of targets (potential root radicals) as well as the number of characters available in a word.

- The number of roots in Arabic is significantly higher. We pre-compiled a list of 3822 trilateral roots from Buckwalter's list of roots, 2517 of which occur in our corpus. According to our lists, Arabic has almost twice as many roots as Hebrew.

- Not only is the number of roots high, the number of patterns in Arabic is also much higher than in Hebrew.

- While in Hebrew the only possible letters which can intervene between root radicals in a word are *i* and *w*, in Arabic there are more possibilities. The possible intervening letter sequences between r_1 and r_2 are *y, w, A, t* and *wA*, and between r_2 and r_3 *y, w, A* and *A*}.[4]

We applied the same methods discussed above to the problem of learning (Modern Standard) Arabic roots. For training and testing, we produced a corpus of 31,991 word types (we used Buckwalter (2002)'s morphological analyzer to analyze a corpus of 152,666 word tokens from which our annotated corpus was produced). Table 15 shows the distribution of word types according to root ambiguity.

We then trained naïve classifiers to identify each radical of the root in isolation, using features of the same categories as for Hebrew. Despite the rather pessimistic starting point, each classifier provides satisfying results, as shown in table 16, probably owing to the significantly larger training corpus. The first three columns present the results of each of the three classifiers, and the fourth column is a straight-forward combination of the three classifiers.

[4] '}' is a character in Buckwalter's transliteration.

Roots	Words
1	28741
2	2258
3	664
4	277
5	48
6	3

Table 15: Arabic root ambiguity in the corpus

	R_1	R_2	R_3	root
Precision:	86.02	70.71	82.95	54.08
Recall:	89.84	80.29	88.99	68.10
f-measure:	87.89	75.20	85.86	60.29

Table 16: Accuracy of SNoW's identifying the correct radical in Arabic

We combined the classifiers using linguistic knowledge pertaining to word-formation processes in Arabic, by implementing a function that estimates the likelihood of a given candidate to be the root of a given word. The function actually checks the following cases:

- If a root candidate is indeed the root of a given word, then we expect it to occur in the word consecutively or with one of {*y*, *w*, *A*, *t*, *wA*} intervening between R_1 and R_2, or with one of { *y*, *w*, *A*, *A*} } between R_2 and R_3 (or both).

- If a root candidate does not occur in our pre-compiled list of roots, it cannot be a root of any word in the corpus.

We suppressed the constraints of weak paradigms in the Arabic experiments, since in such paradigms we are limited to a partial check on the root as we only check for the occurrence of two root radicals instead of three in the word. This limitation seems to be crucial in Arabic, considering the fact that the number

of roots is much higher and in addition, there are more possible intervening letter sequences between the root radicals. Consequently, more incorrect roots are wrongly extracted as correct ones. Of course, this is an over-simplistic account of the linguistic facts, but it serves our purpose of using very limited and very shallow linguistic constraints on the combination of specialized "expert" classifiers. Table 17 shows the final results.

Precision:	78.21%
Recall:	82.80%
f-measure:	80.44%

Table 17: Results: Arabic root identification

The Arabic results are slightly worse than the Hebrew ones. One reason is that in Hebrew the number of roots is smaller than in Arabic (2152 vs. 3822), which leaves much room for wrong root selection. Another reason can be the fact that in Arabic word formation is a more complicated process, for example by allowing more characters to occur in the word between the root letters as previously mentioned. This may have caused the scoring function to wrongly tag some root candidates as possible roots.

5 Improving local classifiers by applying global constraints

In section 3 we presented several methods addressing the problem of learning roots. In general, we trained stand-alone classifiers, each predicting a different root component, in which the decision for the complete root depends on the outcomes of these different but mutually dependent classifiers. The classifiers outcomes need to respect some constraints that arise from the dependency between the root radicals, requiring a level of inference on top the predictions, which is implemented by the scoring function (section 3.3).

In this section we show that applying global constraints, in the form of the scoring function, not only improves global decisions but also significantly improves the local classification task. Specifically, we show that the performance of identifying each radical in isolation improves after the scoring function is applied. In this experiment we trained each of the three radical classifiers as above, and then applied inference to re-rank the results. The combined classifier now predicts the complete root, and in particular, induces a confidence measure on each of the radicals which, due to re-ranking, may differ from the original prediction of the local classifiers.

Table 18 shows the results of each of the radical classifiers after inference with the scoring function. There is a significant improvement in each of the three classifiers (10% in R_1, 16% in R_2 and 8% in R_3) after applying the global constraints (table 18 vs. table 5). The most remarkable improvement is of the R_2 classifier. The gap between R_2 and other classifiers, as stand-alone classifiers with no external knowledge is 10-12%, due to linguistic reasons. Now, after employing the global constraints, the gap is reduced to only 4%. In such scenarios, global constraints can significantly aid local classifiers.

	R_1	R_2	R_3
Precision:	89.67	84.7	89.27
Recall:	93.08	90.17	93.16
f-measure:	91.34	87.35	91.17

Table 18: Accuracy of each classifier after applying global constraints.

Since the most dominant constraint is the occurrence of the candidate root in the pre-comipled list of roots, we examined the results of applying only this constraint on each of the three classifiers, as a single

global constraint. Although there is an improvement of: 6%, 10% and 3% for R_1, R_2 and R_3 respectively, as shown in table 19, applying this single constraint still performs worse than applying all the constraints mentioned in 3.3. Again, we conclude that re-ranking the candidtaes produced by the local classifiers is essential for improving the accuracy, and filtering out infeasible results is not sufficient.

	R_1	R_2	R_3
Precision:	86.33	78.58	83.63
Recall:	88.59	83.75	88.82
f-measure:	87.45	81.08	86.15

Table 19: Accuracy of each classifier applying the list of roots as a single constraint.

Finally, to further emphasize the contribution of global inference to local classification, we repeated the same experiment, measuring accuracy of each of the radical classifiers induced by the root identification system, for *Arabic*. The results are listed in table 20, and show a significant improvment over the basic classifiers (compare to table 16).

	R_1	R_2	R_3
Precision:	90.41	84.40	87.92
Recall:	92.90	89.59	92.19
f-measure:	91.64	86.92	90.01

Table 20: Accuracy of each classifier after applying global constraints (Arabic).

6 Conclusions

We have shown that combining machine learning with limited linguistic knowledge can produce state-of-the-art results on a difficult morphological task, the identification of roots of Semitic words. Our best result, over 80% precision, was obtained using simple classifiers for each of the root's consonants, and then combining the outputs of the classifiers using a linguistically motivated, yet extremely coarse and simplistic, scoring function. This result is comparable to average human performance on this task.

This work can be improved in a variety of ways. As is well-known from other learning tasks, fine-tuning of the feature set can produce additional accuracy; we expect this to be the case in this task, too. In particular, introducing features that capture contextual information is likely to improve the results. Similarly, our scoring function is simplistic and we believe that it can be improved. The edit-distance function can be improved such that the cost of replacing characters reflect phonological and orthographic constraints (Kruskal, 1999).

In another track, there are various other ways in which different inter-related classifiers can be combined. Here we only used a simple multiplication of the three classifiers' confidence measures, which is then combined with the linguistically motivated functions. We intend to investigate more sophisticated methods for this combination.

Finally, we plan to extend these results to more complex cases of learning tasks with a large number of targets, in particular such tasks in which the targets are structured. We are currently working on morphological disambiguation in languages with non-trivial morphology, which can be viewed as a POS tagging problem with a large number of tags on which structure can be imposed using the various morphological and morpho-syntactic features that morphological analyzers produce.

References

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.

Buckwalter, Tim. 2002. Buckwalter Arabic morphological analyzer. Linguistic Data Consortium (LDC) catalog number LDC2002L49 and ISBN 1-58563-257-0.

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.

Collins, Michael. 2003. Machine learning methods in natural language processing. Talk on COLT 2003: tutorial slides.

Darwish, Kareem. 2002. Building a shallow Arabic morphological analyzer in one day. In Mike Rosner and Shuly Wintner, editors, *Computational Approaches to Semitic Languages, an ACL'02 Workshop*, pages 47–54, Philadelphia, PA, July.

Daya, Ezra, Dan Roth, and Shuly Wintner. 2004. Learning Hebrew roots: Machine learning with linguistic constraints. In *Proceedings of EMNLP'04*, pages 357–364, Barcelona, Spain, July.

Daya, Ezra, Dan Roth, and Shuly Wintner. Forthcoming. Learning to identify Semitic roots. In Antal van den Bosch, Guenter Neumann, and Soudi Abdelhadi, editors, *Arabic Computational Morphology: Knowledge-based and Empirical Methods*, Text, Speech, and Language Technology. Springer.

Even-Shoshan, Abraham. 1993. *HaMillon HaXadash (The New Dictionary)*. Kiryat Sefer, Jerusalem. In Hebrew.

Even-Zohar, Y. and Dan Roth. 2001. A sequential model for multi class classification. In *EMNLP-2001, the SIGDAT Conference on Empirical Methods in Natural Language Processing*, pages 10–19.

Florian, Radu. 2002. Named entity recognition as a house of cards: Classifier stacking. In *Proceedings of CoNLL-2002*, pages 175–178. Taiwan.

Kruskal, Joseph. 1999. An overview of sequence comparison. In David Sankoff and Joseph Kruskal, editors, *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison*. CSLI Publications, Stanford, CA, pages 1–44. Reprint, with a foreword by John Nerbonne.

McCarthy, John J. 1981. A prosodic theory of nonconcatenative morphology. *Linguistic Inquiry*, 12(3):373–418.

Ornan, Uzzi. 2003. *The Final Word*. University of Haifa Press, Haifa, Israel. In Hebrew.

Punyakanok, Vasin and Dan Roth. 2001. The use of classifiers in sequential inference. In *NIPS-13; The 2000 Conference on Advances in Neural Information Processing Systems 13*, pages 995–1001. MIT Press.

Roth, Dan. 1998. Learning to resolve natural language ambiguities: A unified approach. In *Proceedings of AAAI-98 and IAAI-98*, pages 806–813, Madison, Wisconsin.

Schütze, H. and Y. Singer. 1994. Part-of-speech tagging using a variable memory markov model. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*.

Shimron, Joseph, editor. 2003. *Language Processing and Acquisition in Languages of Semitic, Root-Based, Morphology*. Number 28 in Language Acquisition and Language Disorders. John Benjamins.

Tjong Kim Sang, Erik F. and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In Walter Daelemans and Miles Osborne, editors, *Proceedings of CoNLL-2003*, pages 142–147. Edmonton, Canada.

Zdaqa, Yizxaq. 1974. *Luxot HaPoal (The Verb Tables)*. Kiryath Sepher, Jerusalem. In Hebrew.