Identifying Semitic Roots: Machine Learning with Linguistic Constraints

Ezra Daya*
University of Haifa

Dan Roth**
University of Illinois

Shuly Wintner†
University of Haifa

Words in Semitic languages are formed by combining two morphemes: a root and a pattern. The root consists of consonants only, by default three, and the pattern is a combination of vowels and consonants, with non-consecutive 'slots' into which the root consonants are inserted. Identifying the root of a given word is an important task, considered to be an essential part of morphological analysis of Semitic languages, and information on roots is important for linguistics research as well as for practical applications. We present a machine learning approach, augmented by limited linguistic knowledge, to the problem of identifying the roots of Semitic words. While programs exist 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 advantage of our method is an automation of this process, avoiding the bottleneck of having to laboriously list the root and pattern of each lexeme in the language. To the best of our knowledge, this is the first application of machine learning to this problem, and one of the few attempts to directly address non-concatenative morphology using machine learning. More generally, our results shed light on the problem of combining classifiers under (linguistically motivated) constraints.

1. Introduction

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 fills the second slot and the third fills the last slot. See Shimron (2003) for a survey.

---

* Department of Computer Science, University of Haifa, 31905 Haifa, Israel. E-mail: edaya@cs.haifa.ac.il
** Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801. E-mail: danr@cs.uiuc.edu
† Department of Computer Science, University of Haifa, 31905 Haifa, Israel. E-mail: shuly@cs.haifa.ac.il

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

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, and one of the few attempts to directly address the non-concatenative morphology of Semitic languages using machine learning. While there exist programs which can extract the root of words in Arabic (Beesley 1998a, 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 the Arabic morphological analyzer of Buckwalter (2002) only uses "word stems – rather than root and pattern morphemes – to identify lexical items." Buckwalter (2002) further notes that "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.

Identifying the root of a given word is a non-trivial problem, due to the complex nature of Semitic derivational and inflectional morphology and the peculiarities of the orthography. It is also 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 (Owens 1997). Information on roots is important for linguistic research, since roots can shed light on etymological processes, both within a single language and across languages. Furthermore, roots are known to carry some meaning, albeit vague. This information can be useful for computational applications: for example, several studies show that indexing Arabic documents by root improves the performance of information retrieval systems (Al-Kharashi and Evens 1994; Abu-Salem, Al-Omari, and Evens 1999; Larkey, Ballesteros, and Connell 2002).[2]

The contributions of this paper are manifold. First and foremost, we report on a practical system which can be used to extract roots in Hebrew and Arabic (the system is freely available; an on-line demo is provided at `http://cl.haifa.ac.il/projects/roots/index.shtml`). The system can be used for practical applications or for scientific (linguistic) research, and constitutes an important addition to the growing set of resources dedicated to Semitic languages. It is one of the few attempts to directly address the non-concatenative morphology of Semitic languages and extract non-contiguous morphemes from surface forms. As a machine learning application, this work describes a set of experiments in combination of classifiers under constraints. The resulting insights can be used for other applications of the same techniques for similar problems (see, e.g., Habash and Rambow (2005)). Furthermore, this work demonstrates that providing a data-driven classifier with limited linguistic knowledge significantly improves the classification results.

We focus on Hebrew in the first part of this paper. After sketching the linguistic data in section 2 and our methodology in section 3, we discuss in section 4 a simple, baseline, learning approach. We then propose several methods for combining the results of interdependent classifiers in section 5 and demonstrate the benefits of using limited linguistic knowledge in the inference procedure. Then, the same technique is applied to Arabic in section 6 and we demonstrate comparable improvements. In section 7

---

2 These results are challenged by Darwish and Oard (2002), who conclude that roots are inferior to character n-grams for this task.

we discuss the influence of global constraints on local classifiers. We conclude with suggestions for future research.

2. Linguistic background

Root and pattern morphology is the major word formation device of Semitic languages. As an example, consider the Hebrew roots *g.d.l*, *k.t.b* and *r.$.m* and the patterns *haCCaCa*, *hitCaCCut* and *miCCaC*, where the 'C's indicate the slots.[3] When the roots combine with these patterns the resulting lexemes are *hagdala, hitgadlut, migdal, haktaba, hitkatbut, miktab, har$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.[4] 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 *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.[5]

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

---

3 To facilitate readability we use a straight-forward transliteration of Hebrew using ASCII characters, where the characters (in Hebrew alphabetic order) are: 'bgdhwzxviklmnsypcqr$t.

4 In this work we consider only texts in *undotted*, or *unvocalized* script. This is the standard script of both Hebrew and Arabic.

5 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). We do not know why Arabic should have so many more roots than Hebrew.

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, $i.C_1.C_2$.

Although root and pattern morphology is the major word formation device of Semitic languages, both Hebrew and Arabic have words which are not generated through this mechanism, and therefore have no root. These are either loan words (which are oftentimes longer than originally Semitic words, in particular in the case of proper names) or short functional or frequent words whose origin is more ancient. For example, the most frequent token in Hebrew texts is the accusative preposition *'t*, which is not formed through root and pattern processes. Of course, there may be surface forms which are ambiguous: one reading based on root and pattern morphology, the other a loan word, e.g., *npl* (either "Nepal" or "fall (past, 3rd person masculine singular)").

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 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. Adding contextual information renders the problem very similar to that of word sense disambiguation (as different roots denote distinct senses), and we opted to focus only on morphology here.

3. Data and methodology

3.1 Machine learning framework

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; Carlson et al. 1999) 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. It is an on-line linear classifier, as are most of the classifiers used these days in NLP, over a variable number of expressive features. In addition to the "standard" perceptron-like algorithms, SNoW has a number of extensions such as regularization and good treatment of multi-class classification. SNoW provides, in addition to classification, a reliable confidence in the instance prediction which facilitates its use in an inference algorithm that combines predictors to produce a coherent inference.

SNoW has already been used successfully as the learning vehicle in a large collection of natural language related tasks, including part-of-speech tagging, shallow parsing, information extraction tasks, etc., and compared favorably with other classifiers (Roth 1998; Golding and Roth 1999; Punyakanok and Roth 2001; Banko and Brill 2001; Florian 2002; Punyakanok, Roth, and Yih 2005). The choice of SNoW as the learning algorithm in this work is motivated by its good performance on other, similar tasks and on the availability in SNoW of easy to tune state-of-the-art versions of three linear algorithms (Perceptron, Winnow and naïve Bayes) in a single package. As was shown by

Roth (1998), Golding and Roth (1999) and in countless experimental papers thereafter, most algorithms used today, from on-line variations of Winnow and Perceptron to maximum entropy algorithms to SVMs, perform comparably if tuned properly, and the eventual performance depends mostly on the selection of features.

3.2 Data and evaluation

For training and testing, a Hebrew linguist manually tagged a corpus of 15,000 words (from 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, are not formed by the root and pattern paradigm. Such words are excluded from our experiments in this work; in an application, such words have to be identified and handled separately. This can be rather easily done using simple heuristics and a small list of frequent closed-class words, because words which do not conform to the root and pattern paradigm are either (short, functional) closed-class words, or loan words which tend to be longer and, in many cases, involve 'foreign' characters (typically proper names). This problem is orthogonal to the problem of identifying the root, and hence a pipeline approach is reasonable.

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 word types	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 annotated corpus into two

sets: a *training set* of 4800 words and a *test set* of 442 words. Only the training set was used for most experiments, and the results reported here refer to these data unless stated otherwise. We used the training set to tune the parameter δ (see section 5.4), and once δ was set we report on results obtained by training on the training set and testing on test data (table 8).

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 correct candidates produced by the system; *fp* as the number of candidates which are not correct roots; and *fn* as the number of roots the system did not produce. As usual, we define *recall* as $\frac{tp}{tp+fp}$, *precision* as $\frac{tp}{tp+fn}$ and f-measure as $\frac{2 \times recall \times precision}{recall+precision}$; we then (macro-) average over all words 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 training 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%. We conjecture that the main reasons for the low performance of our subjects are the lack of context (people tend to pick the most prominent root and ignore the less salient ones) and the ambiguity of some of the weak paradigms (Hebrew speakers are unaware of the correct root in many of the weak paradigms, even when only one exists).

3.3 Feature design

All the experiments we describe in this work share the same features and differ only in the target classifiers. One of the advantages of SNoW is that it makes use of variable-sized feature vectors, represented as the list of the active (present) features in a given instance, rather than the fixed-sized Boolean vectors. This facilitates the use of very long (theoretically, unbounded) feature lists, which are typically very sparse. The features that are used to characterize a word are both grammatical and statistical:

- Position of letters (e.g., the third letter of the word is *b*). We limit word length to 20, as the longest string generated by a Hebrew morphological generator (Yona and Wintner Forthcoming) is 18. We thus obtain up to 440 features[6] 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 up to 484 features.

- Prefixes (e.g., the word is prefixed by *k\$h* "when the"). We have 292 features of this type, corresponding to 17 prefixes and sequences thereof.

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

---

6 Some of these features are never active and are thus never represented.

The lists of suffixes and of prefix sequences were compiled from a morphological grammar of Hebrew (Yona and Wintner Forthcoming). In the general case, such features can be elicited from (non-expert) native speakers or extracted from a morphologically analyzed corpus, if one exists.

3.4 Linguistic resources

One of our goals in this work is to demonstrate the contribution of limited linguistic knowledge to a machine-learning approach to an NLP task. Specifically, we used the following resources for Hebrew and Arabic:

- a list of roots (section 2)

- lists of common prefixes and suffixes (section 3.3)

- corpora annotated with roots (section 3.2)

- knowledge of word-formation processes, and in particular the behavior of the weak roots in certain paradigms (see section 5.4).

It is important to note that these resources do not constitute a method for identifying, even approximately, the root of a given word. We are unaware of any set of rules which attempts to address this task, or of the chances to solve this problem deterministically.

4. Naïve classification methods

4.1 Direct prediction

To establish a baseline, we first performed two experiments with simple, baseline classifiers. 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. There is no generalization when the whole root is predicted as a single unit with a simple classifier.

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%.

4.2 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. While each classifier performs well in isolation, 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%. Repeating the same experiment, but testing only 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 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.

5. Combining interdependent classifiers

5.1 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 few exceptions, all radicals must be present in any inflected form. In fact, when roots and patterns combine, the first radical can be deleted only when it is *w, i, n* and in an exceptional case *l*; the second radical can only be deleted if it is *w* or *i*; and the third, only if it is *i*. We therefore trained the classifiers to consider as targets, during training and testing, only letters that occurred in the observed word, plus *w, i, n* and *l* (depending on the radical), 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 4 (compare to table 3). Combining the results in the straight-forward 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 4
Accuracy of identifying the correct radical, sequential model

5.2 Sequential combination

Evidently, simple combination of the results of the three classifiers leaves much room for improvement. We therefore 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, as shown in table 5. 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.

	R_1	R_2	R_3
top-1:	80.88	70.20	82.50
top-2:	92.98	86.99	93.85
top-5:	99.14	99.38	99.68
top-10:	99.69	99.90	99.70

Table 5
Recall of identifying the correct radical among top-n candidates

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 natural language problems, including part-of-speech tagging (Schütze and Singer 1994), shallow parsing (Punyakanok and Roth 2001) and named entity recognition (Tjong Kim Sang and De Meulder 2003).

However, perhaps not surprisingly given the difficulty of the problem, this model is 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 *q.r.n* was correctly generated by SNoW as the best candidate for the word *mqrn*, but since $P(R_3 = r \mid R_2 = r)$, which is 0.066, is higher than $P(R_3 = n \mid R_2 = r)$, which is 0.025, the root *q.r.r* was produced instead.

Similar examples of interdependencies among radicals abound. 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. We therefore extend the naïve HMM approach to

account for such dependencies, following the PMM model of Punyakanok and Roth (2001).

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 then select the root which maximizes the confidence of R_3; selecting the root which maximizes all three classifiers yields similar results, as shown in table 6.

	Maximizing all radicals	Maximizing R_3	Baseline (Table 4)
Precision:	76.99	76.87	59.83
Recall:	84.78	84.78	57.98
f-measure:	80.70	80.63	58.89

Table 6
Results: Combining dependent classifiers

As the results demonstrate, this is a promising approach which we believe can yield even better results with more training data. However, as we show below, adding more linguistic knowledge can improve performance even further.

5.3 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 are depicted in table 7.

	$R_1R_2 \& R_2R_3$	$R_1R_2 \& R_1R_3$	$R_2R_3 \& R_1R_3$
Precision:	82.11	79.71	79.11
Recall:	85.28	86.40	86.64
f-measure:	83.67	82.92	82.71

Table 7
Results: combining classifiers of root radicals bigram

5.4 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.

It is important to note 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 average more than 11 candidates which score highest (and hence are ranked equally). The linguistic knowledge employed by the system, while significant to the improved performance, is far from being sufficient for devising a deterministic root extraction algorithm.

We now discuss the constraints employed by the scoring function in detail. In what follows, a root $r = r_1r_2r_3$ is said to be 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 $r_2 = r_3$; and *regular* if none of the above holds. The constraints are deterministic, in the sense they they always hold in the training data. They can be easily evaluated since determining the paradigm(s) a given root belongs to is deterministic and efficient. In the examples, root consonants are typeset in boldface.

1. If r is regular then r_1, r_2, r_3 must occur in the word in this order. Furthermore, either $r_1r_2r_3$ are consecutive in the target word, 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. For example, *h**gr**ywm hm**sxr**i **gd**l bkt$**y**h '**xwz**im; hh**std**rwt h**zd**rzh lh**ctl**m wlh**cvd**q*.

2. If r is in Paradigm P1 and not in Paradigm P2, P3 or P4, then r_2, r_3 must occur in the word in this order. Furthermore, either $r_2 r_3$ are consecutive in the target word, or a single letter intervenes between r_2 and r_3 . The intervening letter can only be w or i. Examples: *l$bt (**i.$.b**); hwdiyh (**i.d.y**); mpilim (**n.p.l**)*.

3. If r is in Paradigm P2 and not in Paradigm P1 or P3, then r_1, r_3 must occur in the word in this order. Furthermore, either $r_1 r_3$ are consecutive in the target word, 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). Examples: *hqmt (**q.w.m**) hm**lwn** hcviirh (**c.i.r**) kmbi'h (**b.w.'**) r**ww**xim*.

4. If r is in Paradigm P3 and not in Paradigm P1 or P2, then r_1, r_2 must occur in the word in this order. Furthermore, either $r_1 r_2$ are consecutive in the target word, 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). Examples: *tgliwt (**g.l.i**); hzdkw (**z.k.i**)*.

5. If r is in Paradigm P4 and not in Paradigm P1 or P2, then r_1, r_2 must occur in the word in this order. Furthermore, either $r_1 r_2$ are consecutive in the target word, 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). Examples: *mgilh (**g.l.l**); hmginim (**g.n.n**)*.

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 paradigm constraints (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 in the word. For roots that are in more than one paradigm, 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. Edit distance (Levenshtein 1965) is the minimum number of character insertions and deletions required to transform one string to another. For example, for *hipilw*, the (correct) root *n.p.l* scores $1/5$ whereas *h.p.l* scores $1/3$. The inverse edit

distance increases with the similarity between two strings; finer tuning of this similarity measure is of course possible, but was not the focus of this work.

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).

Recall that a given word form may have several possible roots; our system therefore has to determine how many roots to produce 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 training set and was set to $\delta = 0.4$. With this value for δ, results for the test data are presented in table 8.

	System	Baseline
Precision:	**80.90**	59.83
Recall:	**88.16**	57.98
f-measure:	**84.38**	58.89

Table 8
Results: using linguistic constraints for inference

The results clearly demonstrate the added benefit of the linguistic knowledge. Interestingly, even when testing the system on a set of roots which do *not* occur in the training corpus, we obtain an f-score of 65.60%. This result demonstrates the robustness of our method.

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.

5.5 Error analysis

Looking at the questionnaires filled in by our subjects (section 3.2), 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 (this group is titled 'mixed' below). As an additional experiment, we extracted from the corpus a sample of 200 'hard' words: these are surface forms in which either one of the root characters is missing, or two root characters are transposed due to metathesis. The results are presented in table 9, and clearly demonstrate the difficulty of the system on the weak paradigms, compared to almost 95% on the easier, regular roots.

	Regular	Irregular	Mixed	Hard
Number of words	2598	2019	2781	200
Precision:	92.79	60.02	92.54	47.87
Recall:	96.92	73.45	94.28	55.11
f-measure:	94.81	66.06	93.40	51.23

Table 9
Error analysis: performance of the system on different cases

A more refined analysis reveals differences between the various weak paradigms. Table 10 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 $R_3 = i$. Refer back to table 2 for the sizes of the different root classes.

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 10
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.

6. 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 the transliteration of Modern Standard Arabic orthography of Buckwalter (2002). Thus, the learning problem is more complicated due to 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}*.[7]

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 the morphological analyzer of Buckwalter (2002) to analyze a corpus of 152,666 word tokens from which our annotated corpus was produced). Table 11 shows the distribution of word types according to root ambiguity.

Number of roots	1	2	3	4	5	6
Number of words	28741	2258	664	277	48	3

Table 11
Arabic root ambiguity in the corpus

We then trained standalone classifiers to identify each radical of the root in isolation, using features of the same categories as for Hebrew: location of letters, letter bigrams (independently of their location), and prefixes and suffixes compiled manually from a morphological grammar (Buckwalter 2002). Despite the rather pessimistic starting point, each classifier provides satisfying results, as shown in table 12, 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.

---

7 '}' is a character in Buckwalter's transliteration.

	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 12
Accuracy of 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 approximates 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 13 shows the final results.

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

Table 13
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.

7. Improving local classifiers by applying global constraints

In section 5 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 classifiersŠ 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 5.4).

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 new local classifier decision on each of the radicals which, due to re-ranking, may differ from the original prediction of the local classifiers.

Table 14 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 after applying the global constraints (table 14; cf. table 4). 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 classification.

	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 14
Accuracy of each classifier after applying global constraints

Since the most dominant constraint is the occurrence of the candidate root in the pre-compiled 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 in all classifiers, as shown in table 15, applying this single constraint still performs worse than applying all the constraints mentioned in 5.4. Again, we conclude that re-ranking the candidates 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 15
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 16, and show a significant improvement over the basic classifiers (compare to table 12).

	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 16
Accuracy of each classifier after applying global constraints (Arabic)

8. 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% accuracy, 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 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). Other, learning-based, re-ranking methods can also be used to improve the results.

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 part-of-speech 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.

Idan Szpektor for useful comments. We benefitted greatly from useful and instructive comments by three reviewers.

References

Abu-Salem, Hani, Mahmoud Al-Omari, and Martha W. Evens. 1999. Stemming methodologies over individual query words for an Arabic information retrieval system. *Journal of the American Society for Information Science*, 50(6):524–529.

Al-Kharashi, Ibrahim A. and Martha W. Evens. 1994. Comparing words, stems, and roots as index terms in an Arabic information retrieval system. *Journal of the American Society for Information Science*, 45(8):548–560.

Banko, Michele and Eric Brill. 2001. Scaling to very very large corpora for natural language disambiguation. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 26–33, Morristown, NJ, USA. Association for Computational Linguistics.

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.

Carlson, Andrew J., Chad M. Cumby, Jeff L. Rosen, and Dan Roth. 1999. The SNoW learning architecture. Technical Report UIUCDCS-R-99-2101, UIUC Computer Science Department, May.

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.

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.

Darwish, Kareem and Douglas W. Oard. 2002. Term selection for searching printed Arabic. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on research and development in information retrieval*, pages 261–268, New York, NY, USA. ACM Press.

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. 2007. Learning to identify Semitic roots. In Abdelhadi Soudi, Guenter Neumann, and Antal van den Bosch, editors, *Arabic Computational Morphology: Knowledge-based and Empirical Methods*, volume 38 of *Text, Speech and Language Technology*. Springer, pages 143–158.

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

Even-Zohar, Yair 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.

Golding, Andrew R. and Dan Roth. 1999. A Winnow based approach to context-sensitive spelling correction. *Machine Learning*, 34(1-3):107–130.

Habash, Nizar and Owen Rambow. 2005. Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 573–580, Ann Arbor, Michigan, June. Association for Computational Linguistics.

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.

Larkey, Leah S., Lisa Ballesteros, and Margaret E. Connell. 2002. Improving stemming for Arabic information retrieval: light stemming and co-occurrence analysis. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–282, New York, NY, USA. ACM Press.

Levenshtein, Vladimir I. 1965. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848.

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.

Owens, Jonathan. 1997. The Arabic grammatical tradition. In Robert Hetzron, editor, *The Semitic Languages*. Routledge, London and New York, chapter 3, pages 46–58.

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.

Punyakanok, Vasin, Dan Roth, and Wen-Tau Yih. 2005. The necessity of syntactic parsing for semantic role labeling. In *Proceedings of IJCAI 2005*, pages 1117–1123.

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, Hinrich and Yoram 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*, pages 181–187.

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.

Yona, Shlomo and Shuly Wintner. Forthcoming. A finite-state morphological grammar of Hebrew. *Natural Language Engineering*.

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