Syntactic Analysis of Hebrew Sentences*

Shuly Wintner and Uzzi Ornan
Computer Science Department, Technion
Haifa, Israel 32000
e-mail: {shuly,ornan}@cs.technion.ac.il

Abstract

Due to recent development in the area of computational formalisms for linguistic representation, the task of designing a parser for a specified natural language is now shifted to the problem of designing its grammar in certain formal ways. This paper describes the results of a project whose aim was to design a formal grammar for modern Hebrew. Such a formal grammar has never been developed before. Since most of the work on grammatical formalisms was done without regarding Hebrew (and other Semitic languages as well), we had to choose a formalism that would best fit the specific needs of the language. This part of the project has been described elsewhere. In this paper we describe the details of the grammar we developed. The grammar deals with simple, subordinate and coordinate sentences as well as interrogative sentences. Some structures were thoroughly dealt with, among which are noun phrases, verb phrases, adjectival phrases, relative clauses, object and adjunct clauses; many types of adjuncts; subcategorization of verbs; coordination; numerals, etc. For each phrase the parser produces a description of the structure tree of the phrase as well as a representation of the syntactic relations in it. Many examples of Hebrew phrases are demonstrated, together with the structure the parser assigns them. In cases where more than one parse is produced, the reasons of the ambiguity are discussed.

1 Introduction

Computer "understanding" of a natural language text, which means the ability of a computer to process natural language input and to produce output that will resemble human reaction, is composed of four phases:

Morphologic Analysis – whose goal is to identify the lexical items in the text and to collect the grammatical information they contain;

Syntactic Analysis – whose goal is to grasp the structure by which phrases, sentences, paragraphs and even larger bulks of text are composed and to produce a representation of these structures;

Semantic Analysis – whose goal is to associate each structure in the text with some meaning, drawn from some semantic domain;

Pragmatic Processing – which improves the output of the previous phase by using information on the speaker, the context, the environment and other non-linguistic resources.

These stages of processing do not have to be distinct, and some of them can be merged; in certain conditions some stages may be skipped; however, this is the general scheme for computerized processing of natural language.

There are many applications for natural language understanding: automatic translation of texts from one language to another; automatic condensation of texts; natural language interfaces for data systems

* *Natural Language Engineering* 1(3):261-288, September 1995

and robots etc. For different applications different levels of understanding suffice, but since the syntactic analysis is such a basic phase in natural language processing, it is likely to be a part of every system.

In recent years many linguistic theories have been developed which describe the ways by which a grammar for natural languages should be defined. As an outcome of these theories **grammatical formalisms** have been developed. Many of these formalisms were combined with parsing algorithms and were implemented as **parser compilers** – systems in which the grammar writer defines the rules of the language according to the formalism, and a parser for that language is automatically generated.

In a previous work ([21]), we compared and evaluated some environments for developing grammars, our criteria being the fitness of the models for defining a computational grammar for Hebrew. Among these models were PATR ([15, 17]), Generalized LR Parser/Compiler ([18, 19]) and Slot Grammar ([10]). For various reasons we choose Generalized LR Parser/Compiler (LRP/C) as the framework for developing our grammar. Once the environment has been set, our main concern was to define a fairly broad-coverage grammar of modern Hebrew.

In the next section we list some of the problems that the Hebrew language poses for computational processing and survey some of the works that were done for Hebrew. In section 3 we describe the input and output of the parser, as well as the structure of the rules and of the lexicon. Section 4 lists some of the rules we defined. Among the phenomena we describe are noun phrases, verb phrases and subcategorization, prepositional, adjectival and adverbial phrases, sentences, including subordinate and relative clauses and interrogative sentences, coordination and numbers. We cite some of the grammar rules and accompany them with examples to illustrate how they are applied. In addition we present some ambiguous phrases and specify some of the reasons of syntactic ambiguity in Hebrew.

2 Hebrew Processing

The Hebrew script has two features that make it hard to process: first, it uses a character set that differs from the one that appears on an ordinary computer keyboard; second, it is highly ambiguous as most of the vowels are not written (see [13]). In addition, Hebrew is an agglutinative language in which words are formed by combining a number of morphemes and applying morphophonemic changes to them. Consequently, the task of morphological processing of Hebrew is far from trivial.

To overcome the problems, two different lines of work were pursued. The first, exemplified by works such as [1, 2] or [5], analyzes Hebrew words written in the Hebrew script, thus obtaining a high degree of ambiguity, that is later reduced by contextual and heuristic information. The full morphological analyzers are described in [1, 4].

The second approach transliterates Hebrew using a **phonemic script** which is highly unambiguous. The process of transliteration can be done automatically (see [13]), and results in many images for one word; here, too, the ambiguity can be reduced by contextual and heuristic knowledge as is done in [14]. Various projects were carried out in this framework, among which are [12, 9, 7, 16, 14].

Thus, several works tackle the problems of Hebrew morphology with high degree of success. Therefore, our work is concentrated with the syntax only. We decided to use the phonemic script as our input language script, and assume that the input has already been analyzed morphologically.

The task of describing a reasonable set of grammar rules in a formal manner has scarcely been done for Hebrew. Informal descriptions of Hebrew grammar are given in, e.g., [6, 11]. A more formal, transformation-based grammar is suggested in [3]. While it is more suitable for computational processing than the previously mentioned descriptions, it relies heavily on (non context free) transformations and reflects more the concepts of the grammar of English than that of the Hebrew language.

3 Structure of the Grammar

3.1 Input

As mentioned above, we use as input a transcription of Hebrew known as Phonemic Script ([12, 13]).[1] Table 1 shows the phonemic transcription of the consonants. For more details regarding this script refer to [14].

' - aleph	b - beth	g - gimmel	d - daleth	h - he	w - waw
z - zayn	x - xeth	@ - teth	y - yod	k - kaph	l - lammed
n - nun	s - samekh	‘ - ‘ain	p - pe	c - tsadi	q - qoph
r - reish	$ - shin	j - sin	t - tav		

Table 1: Phonemic Representation of Consonants

Unlike Hebrew script, Phonemic Script represents vowels explicitly. The representation uses the letters *a,e,i,o,u* for the five vowels of Hebrew. In Segolate nouns, where the last vowel is not stressed, it is not written. Double consonants are written explicitly. Also, unlike Hebrew script, every word in Phonemic Script has a stand-alone position. Prepositions, definite article, conjunctions, etc., which are attached to their successive word in Hebrew script, will here appear separately. Efficient algorithms exist for transforming Phonemic script to Hebrew Script; the other direction can be done to some extent, but more information is needed to reduce ambiguity (see [14]).

3.2 Output

The output of the parser, when applied to an input sentence, is a representation of the sentence's structure and the grammatical functions that are filled by each of its constituents. The categories and functions are printed in capitals while the words (the terminals) are printed in lower-case. For example, consider the output for the sentence[2]

ha-	yalda	racta	'et	ha-	tappuz
the	girl	wanted	ACC	the	orange

the girl wanted the orange

```
PRED     SUBJ     NP       ART      ha-
                           HEAD     yalda
         HEAD     racta (raca)
         OBJ      PP       PREP     'et
                           POBJ     NP       ART      ha-
                                             HEAD     tappuz
```

The output represents the fact that the input string is indeed a sentence, described as a predication (PRED) whose head is the main verb “racta”. Three functions are realized in the sentence: SUBJect, HEAD and OBJect. The subject is a Noun Phrase that contains two functions: an ARTicle, represented by the word “ha-”, and a HEAD: the noun “yalda”. The head of the sentence is the verb “racta”, and its base (“raca”) is printed next to it. The object is realized by a Prepositional Phrase that is composed of a PREPosition “'et” and a Preposition OBJect which is a Noun Phrase headed by the noun “tappuz”.

This representation of the output is not produced by LRP/C; to produce it we wrote special functions that carry the structure of constituents during the parsing. The output can thus include information of constituents that don't appear explicitly in the sentence. One such case is pronouns that are attached

[1]This script is under final consideration as a standard number ISO-259-3.

[2]Hebrew uses a preposition, “'et”, to introduce definite direct objects.

to words (both to prepositions: "lo" (to him), "bahem" (in them), "lpaneiha" (before her); and to possessives: "yaldi" (my child), "rconenu" (our wish)). When a pronoun is attached to a word, the parser separates it from the word and associates with it a unique symbol. For example, that is the output of the sentence

ha-	yeld	natan	lah	'et	bubbatah
the	boy	gave	to-her	ACC	her-doll

the boy gave her her doll

```
PRED    SUBJ       NP         ART        ha-
                   HEAD       yeld
        HEAD       natan (natan)
        OBJ        PP         PREP       l-
                              POBJ       hi'  var4851
        OBJ        PP         PREP       et
                              POBJ       NP         HEAD         bubba
                                                    POSSESSIVE   hi'  var5336
```

The symbol var4851 for the object pronoun of the preposition "l- (to)" and the symbol var5336 for the possessive pronoun of the noun "bubbatah (her doll)" indicate that the constituents that fill these roles are not known yet. A system for anaphora resolution can make use of these symbols.

Another case in which the parser adds information on non-explicit relations is the case of relative clauses. In these clauses the parser binds the head noun to the function it fills in the relative clause. Consider the output for the noun phrase

ha-	wa'da	ha-	bodeqt	'et	ha-	'irru'im
the	committee	the	is-checking	ACC	the	events

the committee that checks the events

```
NP      ART       ha-
        HEAD      wa'da (var1311)
        NREL      SUBORD    ha-
                  PRED      HEAD      bodeqt (badaq)
                            OBJ       PP         PREP     et
                                                 POBJ     NP      ART     ha-
                                                                  HEAD    'irru'im
                            SUBJ      var1311
```

The parser recognizes the fact that the noun "wa'da" (committee) is the subject of the relative clause "bodeqt 'et ha-'irru'im (checks the events)". The relation is represented by assigning the same symbol to both constituents. Relative clauses are discussed in more detail in section 4.3.2.

3.3 Structure of the Rules

In what follows, we give a brief description of LRP/C rules structure. The interested reader is referred to [17, 18, 20] for more details.

LRP/C is an Augmented Context Free Grammar (CFG) based formalism. Each of the rules in an LRP/C grammar is an augmented Context Free (CF) rule composed of two constituents: a CF rule and a set of equations. The equations describe several operations among the feature structures that are associated with each of the symbols in the CF rules. Within the equations one can refer to these symbols: the left hand side symbol is referred to as X0; the leftmost symbol on the right hand side is referred to as X1; the next symbol in the right hand side is X2 etc.

The basic operation on feature structures is unification, denoted by the operator '='. LRP/C uses *pseudo unification* rather than bona-fide unification (see [18, 19]). Other operations are constraining equations (denoted by '`=c`'), assignment operation ('`<=`'), calls to LISP functions ('`*test*`'), disjunctive equations ('`*eor*`') and so on.

A **path** in a feature structure is denoted by a list of features in LRP/C. The path *(X0 agr number)*, for example, refers to the value of the *number* feature in the feature structure which is the value of the feature *agr* on the feature structure that is associated with the symbol in the left hand side of the rule.

3.4 Structure of the Lexicon

Since the scope of this work is syntactic analysis only, we assume that the input sentence's words have already been analyzed morphologically. This assumption is reasonable considering the fact that a number of works that deal with morphological analysis of Hebrew have been carried out (see section 2). The assumption is realized by managing a lexicon of inflections rather then basic forms. Of course, for a practical system, the number of entries in such a lexicon will be inconceivable; a full morphological analyzer has been constructed in the Technion recently and we plan to interface our parser with it in the future.

A lexicon in LRP/C is simply a set of rules, the structure of which resembles very much that of the grammar rules. The right hand side of each rule consists of a terminal word. The equations that are associated with the rule serve as a means of initiating the feature structure of the symbol on the left hand side. The following is an example of the lexicon rule for the adjective "gdola ($\text{big}_{\text{sg,f}}$)":

```
(<adj> <--> (g d o l a)
       (
        ((x0 head agr number) = sg)
        ((x0 head agr gender) = f)
        ((x0 struct) = "gdola")
       ))
```

In order to ease the task of writing lexical entries, a set of macros was written. The macro for the ADJ category is listed below. It receives a word list `wordlist`, a `gender` and a `number` as parameters and builds the lexical rules automatically. The macro and the actual entry of the adjective "gdola" are listed below.

```
(defmacro %adj (wordlist &key (gender 'm) (number 'sg))
  (if (atom wordlist) (setq wordlist (list wordlist)))
  (append-dolist (word wordlist)
    `(
      (<adj> <--> ,(explode-string word)
             (
              ((x0 head agr number) = ,number)
              ((x0 head agr gender) = ,gender)
              ((x0 struct) = ,word)
             ))
      )))

(%adj ("gdola") :gender f)
```

4 The Grammar Rules

The grammar of Hebrew we defined[3] consists of about one hundred major rules and many more minor ones. Some of these rules are described in this section. The description is ordered by categories; in some

[3]While LRP/C is meant to be an implementation of LFG ([8]), we do not claim that our grammar is aligned with the principles underlying LFG theory.

of the categories we describe the feature structure that is associated, through lexical rules, with the head of the category. Then we list the possible modifiers of the category, as well as the rules for attaching these modifiers to the head and the way in which the feature structures and the parse tree are modified. Some of the grammar rules are cited and examples of parsing of some phrases is given.

The categories we defined do not necessarily stand for structural elements; some categories represent functional elements such as subject or object. By defining the function *subj* as a category one can create rules for building subjects out of noun phrases, verb phrases, etc. The rules that modify a verb phrase by a subject make use of this category. The features that were defined in the feature structures of different categories do not correspond to any particular grammatical theory; the definition and the use of features was guided by convenience and simplicity considerations. The grammar itself is not complete; some linguistic phenomena are not yet covered by it. In some cases, we note constructions that still need additional, more detailed consideration.

4.1 Noun Phrases

A noun phrase NP is a phrase that is headed by a noun. NPs can have possessive adjuncts but can also bear a possessive clitic, which marks the number, gender and person of the possessing noun. For example, the NP "sirrubah (her refusal)" is constructed out of "sirrub (refusal)" and "ah", the clitic denoting third person feminine singular form. Noun-noun compounds are common in Hebrew, and usually the first noun is the head. The first noun sometimes has a form ("nismak") that slightly differs from the regular ("niprad") form. The second noun (which heads an adjunct) is referred to as "somek".

The feature structure associated with each NP in the lexicon contains one feature, *head*, which consists of the following fields:

agr the noun's agreement features. This complex feature contains the following fields:

> **person** - possible values: `1`,`2`,`3` or `nil` (when not specified).
>
> **gender** - `m` for masculine, `f` for feminine or `nil`.
>
> **number** - `sg` for singular, `pl` for plural.

trans - The value of this feature is the lexical representation of the noun.

nismak - Is the noun in "nismak" form. Possible values: `yes`, `no` or `nil` (when the "nismak" form is identical to the regular form).

beinoni - Is the noun in "beinoni (present tense)" form. Such nouns can serve as verbs. Possible values: `yes`/`no`.

modifier - The value of this feature is `vpinf` in nouns that subcategorize for infinitival verb phrase modifiers, such as "wish" ("the wish to succeed") or "decision".

possessive - This complex feature appears in the feature structures of nouns that include a possessive suffix, such as "rcono (his will)". The value of this feature contains the fields *person*, *gender* and *number*. If this feature is defined, the feature *(head agr def)* is added to the noun's feature structure with the value `yes`, to indicate the fact that nouns with a possessive suffix are definite.

For example, that is the feature structure of the noun "sirrubah (her refusal)":

$$\left[\begin{array}{ll} head: & \left[\begin{array}{ll} agr: & \left[\begin{array}{ll} person: & 3 \\ number: & sg \\ gender: & m \end{array}\right] \\ trans: & \text{"}sirrubah\text{"} \\ modifier: & vpinf \\ nismak: & no \\ possessive: & \left[\begin{array}{ll} person: & 3 \\ gender: & f \\ number: & sg \end{array}\right] \end{array}\right] \end{array}\right]$$

The category of nouns in the lexicon is *n*. The category that immediately dominates *n*'s is *np* - a noun phrase whose definiteness is already known. Hence, an *np* can be derived from an *n* by means of two rules: one for definite and the other for indefinite nouns. In both cases we demand that the head noun is not in "nismak" form, since such nouns can become *np*'s only by a noun-noun construct.

```
(<np> <==> (<n>)
      (
       ((x1 head nismak) = no)
       (x0 = x1)
       ((x0 head agr def) = no)
       ((x0 struct) <= (build-struct 'build
                                     'np
                                     `(head ,(x1 head trans))))
       ))
```

```
(<np> <==> (<art> <n>)
      (
       ((x2 head nismak) = no)
       (x0 = x2)
       ((x0 head agr def) = yes)
       ((x0 struct) <= (build-struct 'build
                                     'np
                                     '(art "ha-")
                                     `(head ,(x2 head trans))))
       ))
```

A special rule was written for nouns with a possessive suffix. The reason is that the possessive should be assigned a special variable. The use of the *refer-pron* feature is explained in section 4.3.2.

```
(<np> <==> (<n>)
      (
       ((x1 head possessive) = *defined*)
       ((x1 head nismak) = no)
       (x0 = x1)
       ((x0 head agr def) = yes)
       ((x0 head refer-pron agr) = (x1 head possessive))
       ((x0 head refer-pron funct) = possessive)
       ((x0 head refer-pron var) <= (gensym "var")) ; a new, unique variable
       ((x0 struct) <= (build-struct 'build 'np
                                     `(head ,(x1 head trans))
                                     `(possessive ,(x0 head refer-pron var))))
       ))
```

Other kinds of *np*'s are pronouns (*pron*) and proper nouns (*pn*). They have separate entries in the lexicon with feature structures similar to those of common nouns and an additional feature called *pron* or *pn*, respectively, with the value `yes`. Both pronouns and proper nouns are definite:

```
(<np> <==> (<pron>)
        (
         (x0 = x1)
         ((x0 head agr def) = yes)
         ))

(<np> <==> (<pn>)
       (
        (x0 = x1)
        ((x0 head agr def) = yes)
        ((x0 struct) <= (build-struct 'build
                                      'propn
                                      `(head ,(x1 head trans))))
        ))
```

Many different adjuncts can modify noun phrases, usually in a fixed order. Preceding the head are focus adverbs (categorized under *fadv*) such as "lo (no)", "raq (only)", "dawqa' (in particular)" and "gamm (also)"; quantifiers such as "koll (all)", "robb (most of)" or "xelq mi- (part of)"; and cardinals. The modifiers that succeed the head are, in the following order: "somek" (noun-noun compound); adjectives or ordinals; demonstrative determiners ("ze", "ha-zo", etc.); possessives; prepositional phrases; infinitival verb phrases or apposed clauses; and relative clauses. Though diversions of this order are allowed, they are rare.

We list below some of the properties of adjuncts:

Quantifiers - some quantifiers in Hebrew are specified for number and will only modify nouns that agree with them on number. The same holds for definiteness: some quantifiers can only be added to definite nouns. For example, the word "koll" has two meanings: "all" and "every". In its first meaning, "koll" can modify only definite nouns; in the second, it can only be added to indefinite singulars. The rule that modifies nouns by quantifiers checks agreement between the feature *agr* of the quantifier and the feature *(head agr)* of the noun.

Cardinals - appear in two forms: "niprad" (such as "loa", three) and "nismak" (such as "lot", three of). A nismak number can only modify definite nouns while niprad numbers can be added to indefinite nouns only.
Example:

koll	lot	ha-	yladim	$elli
all	three-of	the	boys	of-me

all my three boys

```
NP      QUANT     koll
        CARDINAL  3
        ART       ha-
        HEAD      yladim
        QINYAN    PREP      $ell
                  POSSESSOR 'ani      var2207
```

Somek (noun construct) - is a noun phrase and it differs from other adjuncts by modifying *n*'s rather than *np*'s. The *n* should be in "nismak" form. Modifying a noun by a "somek" might change some of the head nouns features: the definiteness and the feature *(head possessive)* of the modified phrase are those of the modifier.

Adjectives and ordinal numbers must agree in gender, number and definiteness with the noun they modify.

Possessives must agree in number, gender and person to a possessive suffix, if one exists. For example, the phrase

bubbato	$ell	ha-	yeld
his-doll	of	the	boy
the boy's doll			

is grammatical, while

*bubbato	$ell	ha-	yalda
his-doll	of	the	girl
?the girl's doll			

is not. The parser binds the possessive suffix to the possessive.

Prepositional phrases can be added to nouns unconditionally.

Infinitival verb phrases and subordinate clauses can serve as apposed clauses. They are allowed only if the value of the feature *(head mashlim)* of the noun is `vpinf`. At most one of the two can appear.

Relative clauses - see section 4.3.2.

A complete noun phrase that cannot be modified or coordinated is shifted to the category *cnp*.

4.1.1 Nominalization

Every definite adjective, whether singular or plural, masculine or feminine, can become a noun: "ha-$obab ha-ze (this naughty [boy])".

ha-	$obab	ha-	ze
the	naughty	the	this
this naughty [boy]			

The rule for nominalization of adjectives is simple (where *cadjp* is a complete adjectival phrase):

```
(<np> <==> (<cadjp>)
      (
       ((x1 head agr def) = yes)
       (x0 = x1)
       ))
```

4.2 Verb Phrases

A verb phrase (VP) is a phrase that can serve as the predicate of a sentence. It is usually headed by a verb, though we consider other predicates (that may be headed by adjectives, prepositions and other parts of speech) as members of this category. Let us first describe predicates whose head is a verb. Verbs' feature structures consist of the following features:

trans - a string representing the verb, followed by its base (uninflected) form.

head - which contains the following fields:

- **agr** - a complex feature containing the number, gender and person of the verb (as in nouns).
- **tense** - possible values are `past`, `present`, `future` or `imperative`.

form - which takes the values `finite` or `infinite`. If *form* = `infinite` then *(head agr)* and *(head tense)* are not defined.

inclsubj - this property, valued `yes` or `no`, determines whether the verb's inflection includes a subject. Its value is defined according to the verb's person, number, gender and tense by means of simple rules. When *inclsubj* = `yes` a VP can become a sentence even if no explicit subject exists.[4] However, subjects can still be added to these verb phrases, except in the case where *form* = `infinite`.

subcat - this feature determines the verb's subcategorization frame. It is explained later on in this section.

For example, that is the feature structure of the verb "naskim (we will agree)":

$$\begin{bmatrix} trans: & "naskim(hiskim)" \\ head: & \begin{bmatrix} agr: & \begin{bmatrix} person: & 1 \\ number: & pl \end{bmatrix} \\ tense: & future \\ form: & finite \\ inclsubj: & yes \end{bmatrix} \\ subcat: & ((l-)(\&im)(subcl)(vpinf)) \end{bmatrix}$$

The category of verbs in the lexicon is *v*. The category that immediately dominates *v* is *vp*; it contains additional details about the verb. The feature *type* determines the phrase's head type (in this case, `vp`); the feature *(head subjtype)* determines the categories that can be possible subjects of the predicate. In this case, its value is `np`. This is the rule for creating *vp* out of *v*:

```
(<vp> <==> (<v>)
      (
       (x0 = x1)
       ((x0 type) = vp)
       ((x0 head subjtype) = np)
       ((x0 struct) <= (build-struct 'build
                                     'pred
                                     `(head ,(x0 trans))))
       ))
```

4.2.1 Verb Phrases that are not Headed by Verbs

A nominal sentence in Hebrew is a sentence whose predicate is headed by an adjective, a noun or a preposition. For each case there is an appropriate rule for generating a *vp* out of the original phrase. The feature *type* of the *vp* is then `adj`, `np` or `pp`, respectively, while the feature *(head subjtype)* is valued `np`. An additional feature of such *vp*'s is *(head copula)* whose value is `allowed` – meaning that a copula can be added to nominal predicates: "ha-yeld gadol" – "ha-yeld hu' gadol (the boy [is] big)". Let us see one example of a rule that creates a *vp* out of an *np*:

```
(<vp> <==> (<cnp>)
      (
       ((x0 type) = np)
       ((x0 head) = (x1 head))
       ((x0 head subjtype) = np)
       ((x0 head copula) = allowed)
       ((x0 struct) <= (build-struct 'build
```

[4] In such cases the '`SUBJ`' field in the parse tree has the value '`intern`'

```
                                           'pred
                                           `(head ,(x1 struct))))
          ))
```

Following are some examples of nominal sentences and their parse trees:
Example:

ha-	yeld	gadol
the	boy	big

the boy is big

```
PRED      SUBJ      NP        ART       ha-
                              HEAD      yeld
          HEAD      ADJ       HEAD      gadol
```

Example:

$im'on	pers	hu'	jar	ha-	xuc
Shimon	Peres	is	minister-of	the	outside

Shimon Peres is the foreign-affairs minister

```
PRED      SUBJ      PROPN     HEAD      $im'on pers
          COPULA    hu'
          HEAD      NP        HEAD      jar
                              SOMEK     NP        ART       ha-
                                                  HEAD      xuc
```

There are other cases in which a predicate is not headed by a *v*. One such case deals with modals. Modals are separately stored in the lexicon under the category *modal*; their features are *gender* and *number*. Consider the feature structure of the modal "carik (should)":

$$\begin{bmatrix} struct: & "carik" \\ head: & \begin{bmatrix} agr: & \begin{bmatrix} number: & sg \\ gender: & m \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

A verb phrase headed by a modal must be modified by an infinitival verb phrase, and therefore its *subcat* feature has the value `vpinf`. The rule for building *vp*'s out of modals is the following:

```
(<vp> <==> (<modal>)
      (
       (x0 = x1)
       ((x0 type) = modal)
       ((x0 head subjtype) = np)
       ((x0 head copula) = allowed)
       ((x0 subcat) <= '(vpinf))
       ((x0 struct) <= (build-struct 'build
                                     'pred
                                     `(head ,(x1 struct))))
       ))
```

Example: "ha-yeld carik lalekt l-ha-gann (the boy should go to the kindergarten)":

ha-	yeld	carik	lalekt	l-	ha-	gann
the	boy	should	to-go	to	the	kindergarten

the boy should go to the kindergarten

```
PRED     SUBJ     NP       ART      ha-
                           HEAD     yeld
         HEAD     carik
         OBJ      PRED     HEAD     lalekt (halak)
                           OBJ      PP       PREP     l-
                                             POBJ     NP       ART      ha-
                                                               HEAD     gann
```

Another case of a *vp* that is not headed by a verb is that of impersonals. An impersonal is a member of a small set of words, most of which serve also as adjectives, that do not have person, gender and number features and cannot be derived. An example is the word “@ob (good)” that is the head of the sentence

@ob	lalekt	l-	ha-	yamm
good	to-go	to	the	sea

it’s good to go to the beach

The subject of such a sentence in Hebrew is an infinitival verb phrase (“to go...”). An optional preposition phrase (headed by the preposition “l- (to)”) can modify the head as an object: “qa$e li lir’ot zo’t (it’s hard for me to see it)”.

qa$e	li	lir’ot	zo’t
hard	to-me	to-see	this

it’s hard for me to see it

Therefore, the value of the subcat feature will be “`l-`”.

```
(<vp> <==> (<impersonal>)
      (
       (x0 = x1)
       ((x0 type) = impers)
       ((x0 head subjtype) = vpinf)
       ((x0 head copula) = allowed)
       ((x0 subcat) <= '(l-))
       ((x0 struct) <= (build-struct 'build
                                     'pred
                                     `(head ,(x1 struct))))
       ))
```

Example: “@ob lalekt l-ha-yamm”:

```
PRED     HEAD     @ob
         SUBJ     PRED     HEAD     lalekt (halak)
                           OBJ      PP       PREP     l-
                                             POBJ     NP       ART      ha-
                                                               HEAD     yamm
```

4.2.2 Modifying Verb Phrases

Four kinds of phrases can modify verb phrases: subject, object, modifier and copula. Each of them can appear in each side of the head of the verb phrase, and the grammar does not specify any constraints on the order of these modifiers, since in general any such order is possible.

Copula can modify a verb phrase only if the value of the feature *(head copula)* is `allowed`. After adding the copula this feature is assigned the value `appeared` so that no additional copula can be added. There are some constraints on adding copula to a verb phrase, depending on the type of the latter. They are described in the following rule:

```
(<vp> <==> (<copula> <vp>)
      (
       ((x2 rcomps) = *undefined*)
       ((x2 head copula) =c allowed)
       (*eor* (
               ((x2 type) =c vp)
               ((x2 head tense) = present)
               ((x2 head agr) = (x1 head agr))
               )
              (
               ((x2 type) =c adj)
               ((x2 head agr) = (x1 head agr))
               )
              (
               ((x2 type) =c modal)
               ((x1 head tense) = (*not* present))
               ((x1 head agr) = (x2 head agr))
               )
              (
               ((x2 type) =c impers)
               ((x1 head agr gender) = m)
               ((x1 head agr number) = sg)
               ((x1 head agr person) = 3)
               ((x1 head tense) = (*not* present))
               )
              (
               ((x2 type) =c pp)
               )
              (
               ((x2 type) =c np)
               )
              )
       (x0 = x2)
       ((x0 head tense) <= (x1 head tense))
       ((x0 struct) <= (build-struct 'add-l
                                     (x2 struct)
                                     `(copula ,(x1 struct))))
       ((x0 head copula) <= 'appeared)
       ((x0 verbfirst) = no)
       ))
```

Subject can be added to a verb phrase in which the feature *(head subj)* is undefined. After adding a subject this feature is defined so that no additional subject can be added. The value of the feature *type* of the subject should be equal to that of the feature *(head subjtype)* of the VP.[5] The subject and the predicate must agree in gender, number and person. The rule for adding a subject to a *vp* is depicted here:

```
(<vp> <==> (<subj> <vp>)
      (
       ((x2 rcomps) = *undefined*)
       (*eor* (
```

[5]Subjects do not have to be NPs, even though this is the most common scheme. We detail the possibilities below.

```
               ((x2 type) =c modal)
               ((x1 head agr) = (x2 head agr))
               )
              (
               ((x2 type) =c adj)
               ((x1 head agr) = (x2 head agr))
               ((x1 head agr def) = yes)
               )
              (
               ((x2 type) =c adj)
               ((x1 head agr) = (x2 head agr))
               ((x1 head agr def) = no)
               ((x2 head copula) =c appeared)
               )
              (
               ((x2 type) =c pp)
               ((x1 head agr def) = yes)
               )
              (
               ((x2 type) =c impers)
               )
              (
               ((x2 type) =c np)
               )
              (
               ((x2 type) = vp)
               ((x2 head form) = finite)
               ((x1 head agr) = (x2 head agr))
               )
              )
       ((x2 head subj) = *undefined*)
       ((x1 type) = (x2 head subjtype))
       (x0 = x2)
       ((x0 head refer-pron) <= (my-union
                                  (x1 head refer-pron)
                                  (x2 head refer-pron)))
       ((x0 head subj) = x1)
       ((x0 verbfirst) = no)
       ((x0 struct) <= (build-struct 'add-l
                                     (x2 struct)
                                     `(subj ,(x1 struct))))
       ))
```

The category *subj* can be derived out of a nominative noun phrase or an infinitival verb phrase:

```
(<subj> <==> (<cnp>)
        (
         ((x1 head case) = nom)
         (x0 = x1)
         ((x0 type) = np)
         ))

(<subj> <==> (<vp>)
```

```
(
 ((x1 head form) = infinite)
 ((x1 type) = vp)
 ((x0 struct) = (x1 struct))
 ((x0 type) = vpinf)
 ))
```

Objects and Modifiers : An object is a modifier which is subcategorized for by the head of the verb phrase. The feature *subcat* of verbs contains a list of possible subcategorization frames. Each frame in the list describes possible types of modifiers that can be added to the verb as objects. For example, the subcategorization list of the verb "badaq (check)" is `(et)(subcl)`; it means that this verb can have a direct object (signed by "'et") or an object that is a subordinate clause. While adding modifiers to a *vp* the parser checks if the modifier matches one of the symbols in the *vp*'s subcat list. If it does, then this modifier is considered an object and the symbol is removed from the list. Otherwise, the modifier is considered an adverbial modifier.

Example:

lo'	naskim	l-	$innuyim	b-	yarden
not	we-shall-agree	to	changes	in	Jordan

we shall not accept changes in Jordan

```
PRED     FADV     lo'
         HEAD     naskim (hiskim)
         OBJ      PP       PREP     l-
                           POBJ     NP       HEAD     $innuyim
         ADVRB    PP       PREP     b-
                           POBJ     PROPN    HEAD     yarden
         SUBJ     (intern)
```

In the above example one can note that the prepositional phrase "l-$innuyim (to changes)" was interpreted as an object, while "b-yarden (in Jordan)" – an adverbial. The difference results from the appearance of the preposition "l-" in the subcat list of the verb "hiskim (agree)".

4.3 Sentences

According to the grammar we defined, a sentence in Hebrew is a verb phrase (that is, a predicate) that has a subject. Since the subject fills a role in the sentence, just like the other verb modifiers, and it can appear anywhere in the predicate, either before or after the head, we didn't find a reason to treat it differently. We represent a sentence, as well as a verb phrase, by the symbol PRED in the output.

The rule for creating a sentence out of a *vp* simply checks if the vp has a subject:

```
(<s> <==> (<vp>)
     (
      ((x1 head form) = finite )
      ((x1 head subj) = *defined*)
      (x0 = x1)
      ))
```

Predicates that are headed by an impersonal do not have to have a subject in order to be considered as sentences. For instance, there is no subject in the sentences

haya	li	qar
was	to-me	cold

I was cold

and

yihye	b-	sedr
will-be	in	order

it will be OK

A subject is optional also if it is contained in the verb's derivation:

```
(<s> <==> (<vp>)
     (
      ((x1 head form) = finite )
      ((x1 head subj) = *undefined*)
      (*eor*
       (
        ((x1 head inclsubj) =c yes)
        )
       (
        ((x1 type) =c impers)
        )
       )
      (x0 = x1)
      ((x0 subj) = intern)
      ((x0 struct) <= (build-struct 'add
                                    (x0 struct)
                                    '(subj "(intern)")))
      ))
```

4.3.1 Subordinate Sentences

Apart from relative clauses, which are described in the next section, all subordinate sentences are derived by the category *subord*. *Subord*'s can function as adverbials, objects or adjuncts. Two types of *subord*'s are defined in the grammar, and they differ in the value of the feature *subordtype*.

Subordinate clauses which function as objects or as adjuncts have `that` as the value of this feature. Their subordinating conjunction is "$e-" or "ki" (that). If the function of a subord is an adverbial, the value of the feature *subordtype* in its feature structure is `advp`. The subordinating conjunction in this case is a member of the set *subordinator*: "ki (because)", " 'im (if)", "kdei $e - (so that)", etc. Some *subordinator*s can be added to infinitival verb phrases rather than to sentences, and there is a special rule in the grammar for that purpose.

4.3.2 Relative Clauses

Relative clauses are subordinate clauses that function as adjuncts of noun phrases. They are related to the head noun by means of a subordinate conjunction and by means of a pronoun in the clause that agrees with the head noun in number, gender and person. This pronoun (the **referring pronoun**) can be omitted in two cases: when the head noun is the subject of the relative clause; or when it is its direct object. The first case is exemplified by "ha-yeld $e-'akal tappux (the boy that ate an apple)"

ha-	yeld	$e-	'akal	tappux
the	boy	that	ate	apple

the boy that ate an apple

and the other by

ha-	tappux	$e-	ha-	yeld	'akal
the	apple	that	the	boy	ate

the apple that the boy ate

In the first case no referring pronoun is allowed, while in the second one might appear. The subordinating conjunction of relative clauses is "$e-" or " '$er" (that). The subordinator "ha- (that)" can appear if the head noun is the subject of the relative clause, the relative clause is in present tense and begins with the verb:

ha-	yeld	ha-	'okel	tappux
the	boy	that	is-eating	apple

the boy that is eating an apple

The subordinating conjunction can be omitted if the relative clause begins with a derived preposition that carries the referring pronoun:

ha-	tappux	'otto	ha-	yeld	'akal
the	apple	which$_{acc,sg,m,3rd}$	the	boy	ate

the apple which the boy ate

The main problem in handling relative clauses is spotting the referring pronoun because it might appear anywhere in the sentence, sometimes very deep in its parse tree:

ha-	yeld	$e-	ha-	yladim	$e-	lo'	racu
the	boy	that	the	kids	that	not	they-wanted
ljaxxeq	'im	'xoto	halku	l-	ha-	yamm	
to-play	with	his-sister	they-went	to	the	sea	

the boy that the kids that didn't want to play with his sister went to the beach

Referring pronouns can be attached to prepositions ("lo" – to him), nouns ("bitto" – his daughter), possessives ("$ello" – of him) or verbs ("ra'uhu" – saw him). While parsing the relative clause the parser doesn't know what the head noun is, and hence all possible referring pronouns have to be collected. At the time of attaching the clause to its head noun, one of the members of this list has to be chosen - according to the agreement conditions we mentioned.

The treatment of relative clauses is rather complex, partly due to some limitations of LRP/C's unification operation. We shall not get into details here[6]; let us list some examples of phrases the parser can deal with correctly. Note that the parser binds the head noun to its proper function in the relative clause in the following examples:

ha-	kaddur	$e-	ha-	yeld	jixxeq	bo
the	ball	that	the	boy	played	with-him

  the ball that the boy played with

```
NP      ART     ha-
        HEAD    kaddur (var3371)
        NREL    SUBORD   $e-
                PRED     SUBJ    NP      ART     ha-
                                         HEAD    yeld
                         HEAD    jixxeq (jixxeq)
                         OBJ     PP      PREP    b-
                                         POBJ    hu'     var3371
```

ha-	yeld	$e-	ra'a	'et	ha-	kelb
the	boy	that	saw	ACC	the	dog

  the boy that saw the dog

```
NP      ART     ha-
        HEAD    yeld (var3772)
```

[6]For a complete discussion, see [20].

```
        NREL    SUBORD  $e-
                PRED    HEAD    ra'a (ra'a)
                        OBJ     PP      PREP    et
                                        POBJ    NP      ART     ha-
                                                        HEAD    kelb
                        SUBJ    var3772
```

ha-	kelb	$e-	ha-	yeld	ra'a
the	dog	that	the	boy	saw

  the dog that the boy saw

```
NP      ART     ha-
        HEAD    kelb (var3813)
        NREL    SUBORD  $e-
                PRED    SUBJ    NP      ART     ha-
                                        HEAD    yeld
                        HEAD    ra'a (ra'a)
                        DOBJ    var3813
```

ha-	kaddur	bo	jixxeq	ha-	yeld
the	ball	in-him	played	the	boy

  the ball with which the boy played

```
NP      ART     ha-
        HEAD    kaddur (var3404)
        NREL    PRED    OBJ     PP      PREP    b-
                                        POBJ    hu'     var3404
                        HEAD    jixxeq (jixxeq)
                        SUBJ    NP      ART     ha-
                                        HEAD    yeld
```

4.3.3 Interrogative Sentences

The grammar we defined deals with questions only partially. Yes/no questions are simple in Hebrew – they are formed by attaching "h -'im (is it)" to a declarative sentence.

Example:

h-'im	ha-	yeld	jixxeq	b-	kaddur
is-it	the	boy	played	in	ball

is it true that the boy played with a ball

```
PRED    YNQ     h-'im
        SUBJ    NP      ART     ha-
                        HEAD    yeld
        HEAD    jixxeq (jixxeq)
        OBJ     PP      PREP    b-
                        POBJ    NP      HEAD    kaddur
```

Wh-questions are more complicated. We list here two examples of such questions that the parser recognizes and parses correctly.

Example:

mi	jixxeq	b-	kaddur
who	played	in	ball

who played with a ball

```
PRED    SUBJ    mi
        HEAD    jixxeq (jixxeq)
        OBJ     PP      PREP    b-
                        POBJ    NP      HEAD    kaddur
```

Example:

b-	ma	jixxeq	ha-	yeld
in	what	played	the	boy

what did the boy play with

```
PRED    OBJ     QUEST   ma
        HEAD    jixxeq (jixxeq)
        SUBJ    NP      ART     ha-
                        HEAD    yeld
```

4.4 Other Categories

Besides the rules for the main categories that were described above, we also defined a set of rules that take care of other simpler categories. These rules deal with prepositional phrases; adjectives, including modifiers of adjectives and comparatives; adverbials, including adverbs, prepositional phrases and subordinate sentences that operate as adverbials; and numerals.

The rules for describing numbers in Hebrew can generate both ordinals and cardinals that are less then 1000. Strings of digits can be parsed as well. For each number, in every representation, the parser produces a numeric value.

Example:

lo	me'ot	'arba‘im	w-	$naym	yladim
three-fo	hundreds	forty	and	two	kids

three hundred and forty two kids

```
NP      CARDINAL  342
        HEAD      yladim
```

Example:

ha-	yeld	ha-	$eni
the	boy	the	second

the second boy

```
NP      ART       ha-
        HEAD      yeld
        ORDINAL   2
```

4.5 Coordination

Coordination is the combination of two phrases that share the same category to a new phrase of this category, usually by means of a conjunction. In Hebrew every number of components can be coordinated, and the rules we defined determine that a conjunction should separate the last component from its predecessor. In order to give a uniform treatment of coordination we defined a macro that operates on a category symbol and a set of LRP/C equations. The application of the macro generates a set of four grammar rules that are interwoven into the grammar as an integral part of it; thus, the macro is essentially a meta-rule.

The four rules that are generated by applying the macro enable the generation of a list of phrases of the same category so that the last element of the list must be preceded by a conjunction. One can describe the rules for coordination of a specific category as if exactly two components are involved; the parser allows longer coordinations through the use of the macro. We defined specific coordination rules for the categories *np*, *vp*, *adjp*, *pp* and *advp*.

4.6 Ambiguity

Since natural languages are not deterministic, a grammar for a natural language cannot be deterministic; many sentences and phrases are ambiguous in respect to the grammar we defined. In cases of ambiguity the parser assigns more than one structure to the sentence. We briefly describe here some of the reasons for ambiguity we encountered while testing the grammar.

- Lexical ambiguity:
  Phonemic script, which we used to represent Hebrew words, is almost deterministic. Still there are some words that are lexically ambiguous under this representation. There are other problems of ambiguity that cannot be resolved by the script. One is the phenomenon of differences cancellation: two derivations of different heads result in the same form. An example is the plurals of "xob (debt)" and "xoba (duty)": "xobot". A different problem is under-specification of the word's properties. The "niprad" form of many nouns in Hebrew matches their "nismak" form; many verbs have one representation in both past and present tense; many verb derivations do not specify a person feature; verbs in present form ("beinoni") can function as nouns; etc.

- PP Attachment
  This problem is common to many parsers of various languages. It results from the fact that the Prepositional Phrases can be attached both to verbs and to nouns. If a PP appears after a verb and a noun phrase, it can be added in two different ways. Each additional PP increases the number of possible parses, so that long sentences can sometimes be assigned more than ten trees.

  For example, consider the parsing of the sentence:

lo'	naskim	l-	$innuyim	b-	yarden
not	we-shall-agree	to	changes	in	Jordan

  we shall not accept changes in Jordan

```
;**** parse 1 ***

PRED     FADV     lo'
         HEAD     naskim (hiskim)
         OBJ      PP       PREP     l-
                           POBJ     NP       HEAD     $innuyim
         ADVRB    PP       PREP     b-
                           POBJ     PROPN    HEAD     yarden
         SUBJ     (intern)

;**** parse 2 ***

PRED     FADV     lo'
         HEAD     naskim (hiskim)
         OBJ      PP      PREP     l-
                          POBJ     NP     HEAD       $innuyim
                                          PPADJUNCT  PP     PREP   b-
                                                            POBJ   PROPN  HEAD  yarden
         SUBJ     (intern)
```

  In the first tree the PP "b-yarden (in Jordan)" is interpreted as an adverbial of the verb, while in the second it is the adjunct of the noun "$innuyim (changes)".

- Word Order:
  Like many other morphologically-rich languages, Hebrew is also a relatively free-word-order language and many possibilities that are invalid in English, for instance, are grammatical in Hebrew. The freedom is complete in the order of verb modifiers – the grammar we defined places no constraints on this order. As a result, a number of parses are acquired in cases where modifiers can be interpreted in more than one way (usually - a noun phrase that can function both as a subject and as a direct object).
  Example:

yeld	ra'a	kelb
boy	saw	dog

a boy saw a dog

```
;**** parse 1 ***

PRED      SUBJ      NP        HEAD      yeld
          HEAD      ra'a (ra'a)
          OBJ       NP        HEAD      kelb

;**** parse 2 ***

PRED      OBJ       NP        HEAD      yeld
          HEAD      ra'a (ra'a)
          SUBJ      NP        HEAD      kelb
```

- Coordination:
  Coordination introduces ambiguity when the boundaries of the coordinated phrases can be defined in more than one way. The scope of the conjunction is then ambiguous.
  Example:

ha-	yeld	w-	ha-	yalda	mi-	xeipa
the	boy	and	the	girl	from	Haifa

the boy and the girl from Haifa

```
;**** parse 1 ***

CONJ      NP        ART       ha-
                    HEAD      yeld
          NP        ART       ha-
                    HEAD      yalda
                    PPADJUNCT PP        PREP      mi-
                                        POBJ      PROPN     HEAD      xeipa

;**** parse 2 ***

CONJ      NP        ART       ha-
                    HEAD      yeld
          NP        ART       ha-
                    HEAD      yalda
          PPADJUNCT PP        PREP      mi-
                              POBJ      PROPN     HEAD      xeipa
```

- Noun-Noun Constructs:
  Noun construct ("smikut") is a phrase whose head is a noun ("nismak") and whose modifier is a noun phrase ("somek"). This definition introduces ambiguity in cases where an additional modifier is added to the phrase that can modify both the "nismak" and the "somek".
  Example:

mo'ect	ha-	mdina	ha-	zmannit
council-of	the	state	the	temporary

the temporary state council

```
;**** parse 1 ***

NP        HEAD      mo'ect
          SOMEK     NP        ART       ha-
                              HEAD      mdina
```

```
          NADJ      ADJ       ART       ha-
                              HEAD      zmannit

;**** parse 2 ***

NP        HEAD      mo‘ect
          SOMEK     NP        ART       ha-
                              HEAD      mdina
                              NADJ      ADJ       ART       ha-
                                                  HEAD      zmannit
```

- Prepositions:
  Many prepositions in Hebrew are derived from nouns: "b-qe$r l- (about / in connection with)", "mi-cadd (from / from the side of)", "b-ydei (by / by the hands of)", etc. Prepositional phrases headed by these prepositions can be interpreted differently, when the preposition is understood literally.
  Example:

ha-	'um	raca	haskama	mi-	cadd	yisra'el
the	UN	wanted	agreement	from	side-of	Israel

  UN wanted agreement from Israel

```
;**** parse 1 ***

PRED      SUBJ      NP        ART       ha-
                              HEAD      'um
          HEAD      raca (raca)
          OBJ       NP        HEAD      haskama
                              PPADJUNCT PP     PREP   mi-cadd
                                               POBJ   PROPN  HEAD   yijra'el

;**** parse 2 ***

PRED      SUBJ      NP        ART       ha-
                              HEAD      'um
          HEAD      raca (raca)
          OBJ       NP        HEAD      haskama
                              PPADJUNCT PP     PREP   mi-
                                               POBJ   NP     HEAD   cadd
                                                             SOMEK  PROPN  HEAD   yijra'el
```

- Other Problems:
  Let us exemplify a special case in which structures in Hebrew can be interpreted in very different ways. A relative clause, whose head is a present form ("beinoni"), can be attached to its head noun by means of the conjunction "ha- (that)". If the head noun isn't definite, and it is both "niprad" and "nismak", an ambiguous structure is created: head + relative clause, or "nismak" + "somek" (noun-noun compund). To understand the next example, compare "yaldei ha-xberim ba-wa‘da (the kids of the committee members)" with "yladim ha-xberim ba-wa‘da (kids that are committee members)".
  Example:

mdinot	ha-	xberot	b-	ha-	'um
states/states-of	that/the	members	in	the	UN

  states that are members of the UN
  the states of the UN members

```
;**** parse 1 ***
```

```
NP        HEAD      mdinot
          SOMEK     NP        ART       ha-
                              HEAD      xberot
                              PPADJUNCT PP        PREP      b-
                                                  POBJ      NP      ART     ha-
                                                                    HEAD    'um

;**** parse 2 ***

NP        HEAD      mdinot (var2051)
          NREL      SUBORD    ha-
                    PRED      HEAD      NP        HEAD      xberot
                                                  PPADJUNCT PP      PREP    b-
                                                                    POBJ    NP      ART     ha-
                                                                                    HEAD    'um
                              SUBJ      (var2051)
```

5 Conclusions

The main target of the work described in this paper was to prepare a good syntactic analyzer for the Hebrew language. While the parser we describe cannot deal with *every* input sentence, it does handle successfully many different phenomena in Hebrew syntax. Hebrew sentences that are relatively simple acquire a correct structure, and the number of structures the parser produces for each sentence is reasonable. Moreover, most of these structures are syntactically correct.

The output of the parser is twofold: it combines the phrase structure and the grammatical function of each constituent. The combined description provides richer information on the structure of the input sentences: in Hebrew, where constituent order is relatively free, phrase structure alone does not suffice for determining grammatical functions.

In order to test the grammar we applied the parser to sentences of a newspaper in simple Hebrew ("Sha'ar", # 632, Oct. 16, 1991). The correct parse was a member of the resulting parses in seventy percent of the sentences. Indeed, if the sentences were taken from a free text (say, an evening paper), this percentage would have decreased; yet it is obvious that the grammar encompasses a wide variety of phenomena in Hebrew.

There are some possible extensions to this work. The main one is to extend the coverage of the grammar. Some phenomena in Hebrew were not covered at all while others could benefit from further treatment. The grammar we defined can handle only phrases and sentences, and extensions towards discourse analysis should be considered. Another important target is to link the parser to other existent systems that deal with Hebrew, especially morphological analyzers. Such a linkage, we believe, would produce a practical system that could be used as the skeleton of many natural language processing applications.

References

[1] Esther Bentur, Aviella Angel, and Danit Segev. Computerized analysis of Hebrew words. *Hebrew Linguistics*, 36:33–38, December 1992. (in Hebrew).

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

[3] M. J. Chayen and Z. Dror. *Introduction to Hebrew Transformational Grammar*. University Publishing Projects Ltd., Jerusalem, 1976. (in Hebrew).

[4] Yaacov Choueka. Response to "computerized analysis of Hebrew words". *Hebrew Linguistics*, 37:87, December 1993. (in Hebrew).

[5] Yaacov Choueka and Yoni Ne'eman. "Nakdan-T", a text vocalizer for modern Hebrew. In *Proceedings of the Fourth Bar-Ilan Symposium on Foundations of Artificial Intelligence*, June 1995.

[6] Lewis Glinert. *The Grammar of Modern Hebrew.* Cambridge University Press, Cambridge, 1989.

[7] Lyor Goldstein. Generation and inflection of the possession inflection of Hebrew nouns. Master's thesis, Technion, Haifa, Israel, 1991. (in Hebrew).

[8] R. Kaplan and J. Bresnan. Lexical functional grammar: A formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, Mass., 1982.

[9] Alon Lavie, Alon Itai, Uzzi Ornan, and Mori Rimon. On the applicability of two-level morphology to the inflection of Hebrew verbs. Technical Report 513, Department of Computer Science, Technion, 32000 Haifa, Israel, 1988.

[10] Michael McCord. A new version of slot grammar. Research Report RC 14506, IBM, 1989.

[11] Uzzi Ornan. *The Simple Sentence.* Academon, Jerusalem, Israel, 1979. (in Hebrew).

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

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

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

[15] S. J. Rosenschein and Stuart M. Shieber. Translating English into logical form. In *Proceedings of the 20th Annual Meeting of the Association for Computational Linguistics*, 1982.

[16] Michal Shany-Klein. Generation and analysis of segolate noun inflection in Hebrew. Master's thesis, Technion, Haifa, Israel, 1990. (in Hebrew).

[17] Stuart M. Shieber. *An Introduction to Unification Based Approaches to Grammar.* CSLI Lecture Notes. CSLI, 1986.

[18] Masaru Tomita. The generalized LR parser/compiler: User's guide. CMT memo, Center for Machine Translation, Carnegie Mellon University, 1988.

[19] Masaru Tomita. The generalized LR parser/compiler v8-4: A software package for practical NL projects. In *Proceedings of the 13th International Symposium on Computational Linguistics*, 1990.

[20] Shuly Wintner. Syntactic analysis of Hebrew sentences. Master's thesis, Technion, Israel Institute of Technology, Haifa, Israel, July 1991. (In Hebrew, abstract in English).

[21] Shuly Wintner and Uzzi Ornan. Computational models for syntactic analysis – their fitness for writing a computational grammar for Hebrew. CIS Report 9103, Center for Intelligent Systems, Technion, Israel Institute of Technology, Haifa 32000 Israel, May 1991.