An Arabic Slot Grammar Parser

Michael C. McCord

IBM T. J. Watson Research Center
P.O.B. 704
Hawthorne, NY 10532
mcmccord@us.ibm.com

Violetta Cavalli-Sforza
Language Technologies Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
violetta@cs.cmu.edu

Abstract

We describe a Slot Grammar (SG) parser for Arabic, ASG, and new features of SG designed to accommodate Arabic as well as the European languages for which SGs have been built. We focus on the integration of BAMA with ASG, and on a new, expressive SG grammar formalism, SGF, and illustrate how SGF is used to advantage in ASG.

1 Introduction

In this paper we describe an initial version of a Slot Grammar parser, **ASG**, for Arabic. Slot Grammar (SG) (McCord, 1980. 1993) is dependency-oriented, and has the feature that deep structure (via logical predicate arguments) and surface structure are both shown in parse trees.

A new formalism **SGF** (**S**lot **G**rammar **F**ormalism) for SG syntax rules has been developed (McCord, 2006), and the ASG syntax rules are written in SGF. SGF is largely declarative, and can be called "object-oriented" in a sense we will explain. The rules in SGF all have to do with slot filling.

ASG uses BAMA (Buckwalter, 2002), in a version from Qamus, as its morphological analyzer. All the internal processing of ASG is done with the Buckwalter Arabic transliteration – though of course ASG can take real Arabic script (in UTF-8 form) as input. We use BAMA features in the processing (and parse trees), but augmented with other features more unique to ASG. The Penn Arabic Treebank (ATB), which also uses BAMA features, has served as a development guide in the work. But SG is a rule-based system, and there is no automatic training from the ATB.

Prior to this work, SGs had been written for English (McCord), German (Claudia Gdaniec), and for the Romance languages (Esméralda Manandise) Spanish, French, Italian and Portuguese. For handling Arabic, there have been two main new adaptations of SG.

One adaptation is in the treatment of features in the form that BAMA delivers. This treatment includes a *feature lexicon* in ASG, which can specify two kinds of relations among features, which we will describe below. We also take steps to handle the large number of analyses returned by BAMA. Special treatment of features appears as well in the SGF syntax rules. The other main adaptation is in the treatment of clitics, where special things happen in Arabic for proclitics.

Although the basic ideas of SG have not changed in treating Arabic, ASG has been serving as a test bed for the new syntax rule formalism SGF.

Overall, the design of the SG system has become neater by including Arabic as well as the European languages. For instance, the new treatment of features generalizes the existing treatment in the German SG. And the new treatment of clitics will make the treatment of clitics for the Romance languages neater.

In Section 2, we discuss the ASG feature system. Section 3 briefly describes the ASG slot frame lexicon. Sections 4 and 5 deal with syntactic analysis. In Section 6, we discuss current performance of ASG (coverage and speed), and in Section 7, related work.

Proceedings of the 5th Workshop on Important Unresolved Matters, pages 81–88,
Prague, Czech Republic, June 2007.

2 The Feature System

Features for an SG parser for language *X* are specified externally as character strings, listed by the grammar writer in the *feature lexicon* *X*feas.lx (Arfeas.lx for Arabic). Internally, features are represented in two ways, for efficient processing: (1) The features themselves are "atoms", represented by integers. (2) The set of features for a parse node is represented by a bit string, where each feature atom is assigned a bit position. For ASG, these bit strings are currently of length 512. But these internal representations are invisible to the grammar writer.

In the set of features for a node, some subsets can be viewed disjunctively. For instance if a noun is ambiguously singular or plural, it might have both features **sg** and **pl**. This situation occurs very much for Arabic text input because of the ambiguity due to unvocalized script. In order not to choke the parse space, the SG-BAMA interface combines some BAMA analyses, basically ones that have the same stem and POS, so that nodes have disjunctive BAMA features. But agreement rules or slot filler constraints often reduce the ambiguity. Such rules, specified in a perspicuous way in SGF, as we will see below, are implemented internally by intersecting the bit string representations of relevant feature sets.

For ASG, there are two categories of features. One category consists of BAMA compound features like

```
NOUN+NSUFF_FEM_PL+CASE_DEF_ACC
```

(indicating a feminine plural definite accusative noun). Although such features are compound in intent, they are treated as atomic symbols by ASG (as are all features specified in *X*feas.lx).

Features of the other category are more special to ASG. Some of them have to do with syntactic structure (like presence of an overt subject), and others are morphological. Typical morphological features are standard, simple ones that appear in sets of values for attributes like case, number, gender, and definiteness – for example:

```
nom, acc, gen
sg, dual, pl
m, f,
def, indef
```

Besides declaring features, *X*feas.lx can specify relations between features. One way is to specify simple hierarchical relations. An entry of the form

$x < y \ldots z \ldots$

specifies that feature *x* implies features *y* … *z*. This means for instance that if the feature *x* is marked on a node, then a test in the grammar for feature *y* will succeed. Hierarchical information like this is stored internally in bit string arrays and allows efficient processing.

If an entry is of the form

$x < \ldots > \; u \ldots v$

then we say that *x* *extends* the feature set {*u* ... *v*}, and *x* is an *extending* feature. The basic idea is that *x* is a kind of abbreviation for the disjunction of the set {*u* ... *v*}, but *x* may appear on a node independently of that set. We will explain the exact meaning in the section below on the syntax rules. A typical example of an extending feature rule in Arfeas.lx is as follows:

```
gen < >
  NOUN+NSUFF_FEM_DU_GEN
  NOUN+NSUFF_FEM_PL+CASE_DEF_GEN
  NOUN+NSUFF_FEM_PL+CASE_INDEF_GEN
   ...
```

where we list all BAMA compound features that include a genitive subfeature. Rules in the syntax component can test simply for extending features like **gen**, as we will see below. The syntax component does not even mention BAMA features. But this representational scheme allows us to keep BAMA compound features as units -- and this is important, because the morphological analysis (with ambiguities shown) requires such groupings. The internal representation of an extending feature relationship of *x* to {*u* ... *v*} associates with the atom for *x* the disjunction of the bit strings for *u* ... *v*, and the processing is quite efficient.

Although the features in *X*feas.lx are generally morphosyntactic, and have internal atom and bit string position representations in limited storage areas, SG also allows open-ended features, which may be used in the SG lexicon and tested for in the syntax component. These are typically semantic features.

3 The SG Lexicon

Although BAMA contains lexicons for doing Arabic morphological analysis, an SG needs its *SG lexicon* to drive syntactic analysis and help produce parse trees that show (deep) predicate argument structure. The main ingredients associated with index words in an SG lexicon are *sense frames.* A sense frame can specify a part of speech (POS), features (typically semantic features), a *slot frame*, and other ingredients. The most important ingredient is the slot frame, which consists of an ordered list of (*complement*) *slots*. Slots can be thought of as grammatical relations, but also as names for logical arguments for word sense predicates. An example from the ASG lexicon, called Ar.lx, is:

```
Eoniy < v (obj n fin)
```

This says that **`Eoniy`** (عَنِي) is a verb (stem) with a direct object slot (**`obj`**) which can be filled by either an NP (indicated by the **`n`**) or a finite VP (indicated by the **`fin`**). A slot can be either an atomic symbol or a list of the form

($SlotName\ Option_1 \ldots Option_n$)

where the options are terms that specify conditions on the fillers of the slot. If no options are specified, then defaults are used. The **`Eoniy`** (عَنِي) example shows no subject slot, but the default is that every verb has a subject slot (even though it may not be overtly filled). One can specify a subject slot (**`subj`**) if it needs non-default options.

For the index words for ASG, we are currently using vocalized stems – stems as in the ATB, or as produced by BAMA. To produce a starter for Ar.lx, we extracted stems from the ATB, listed by frequency, and associated default sense frames based on the BAMA features in the ATB. Using vocalized stems entails some repetition of sense frames, since there can be more than one vocalized stem for a given word sense.

Index words in the SG lexicon can also be multiwords. Some multiword entries occur in Ar.lx.

Morpholexical analysis for ASG combines BAMA analysis with look-up in Ar.lx. BAMA provides morphological features (BAMA compound features) associated with vocalized stems. Also, an algorithm in ASG separates clitics out of the BAMA analyses and represents them in a form convenient for the parser. The vocalized stems are looked up in Ar.lx, and the sense frames found there (if look-up is successful) are merged with compatible analyses from BAMA. If look-up in Ar.lx fails, then the BAMA analyses can still be used, with default slot frames assigned. In the other direction, look-up in BAMA may fail, and special entries in Ar.lx can cover such words (specifying morphological features as well as slot frames).

4 The Parsing Algorithm

The SG parser is a bottom-up chart parser. Initial chart elements are one-word (or one-multiword) phrases that arise from morpholexical analysis. All further chart elements arise from binary combinations of a *modifier* phrase *M* with a *higher* phrase *H*, where *M* fills a slot *S* in *H*. The slot *S* could be a complement slot which is stored with *H*, having arisen from the lexical slot frame of the word sense head of *H*. Or *S* could be an adjunct slot associated with the POS of *M* in the syntax rule component *X*.gram. In both cases, the conditions for filling *S* are specified in *X*.gram. The parser attaches postmodifiers first, then premodifiers.

Normally, *M* and *H* will be existing adjacent phrases in the chart. But there is an interesting treatment of clitics that is especially relevant for Arabic. The SG data structure for a phrase *P* includes two fields for clitics associated with the head word of *P* – a list of proclitics, and a list of enclitics. Each clitic is itself a (one-word) phrase data structure, ready to be used for slot filling. So the parsing algorithm can combine not only adjacent phrases in the chart in the normal way, but can also combine a phrase with one of its clitics. For Arabic, all enclitics (typically pronouns) for a phrase *P* are attached to *P* (by postmodification) before *P* enters into any other slot filling. On the other side, proclitics (typically conjunctions and prepositions) of *P* are used only as higher phrases where *P* is the modifier. But a proclitic can get "passed upwards" before it is treated as a higher phrase. A non-deterministic option in the parser is that a phrase *M* becomes a premodifier of an adjacent phrase *H* in the chart, and the proclitic list of *M* is passed up to become the proclitic list of *H*. For instance a conjunction like "w"/"wa" [وَ , "and"] might be attached as a proclitic to the first word in a (premodifying) subject of a clause *C*, and the conjunction proclitic gets passed upwards until it finally takes C as a postconjunct modifier.

Although SG is a rule-based system, it does use a numerical scoring system for phrases during parsing. Real numbers are attached to phrases, indicating, roughly, how likely it is that the phrase is a good analysis of what it spans. Partial analyses (phrases) can be pruned out of the chart if their scores are too bad. Also, final parses get ranked by their scores. Scores can arise from rules in the syntax component, in the lexicon, or in the shell. A general rule in the shell is that complement slots are preferred over adjunct slots. The specific values of scores are normally determined by the grammar writer, with regression testing.

5 The ASG Syntax Rule Component

In an SG syntax rule component *X*.gram (Ar.gram for Arabic), the rules are written in the formalism SGF (McCord, 2006). Each rule deals with slot filling, and is either a complement slot rule or an adjunct slot rule. Each rule is of the form

S < *Body*

where *S* is the *index,* which is a complement slot for a complement slot rule, or a POS for an adjunct slot rule. The *Body* is basically a logical expression (in a form we will describe) which is true iff the corresponding slot filling can succeed. The rules can be viewed largely declaratively, even though there are some operators that look like commands.

The rule system is applied by the parsing algorithm when it is looking at specific phrases *M* and *H* that are adjacent or have a clitic relationship, and asking whether *M* can fill a slot in *H*. For a yet unfilled complement slot *S* of *H*, with a chosen slot option, the parser looks for the complement slot rule in *X*.gram indexed by *S,* and applies its body, requiring that to be true before doing the slot filling. And the parser also looks at the POS of *M*, finds the corresponding adjunct slot rule indexed by that POS, and applies its body. In this case, the body determines what the adjunct slot and option are; and it can do so non-deterministically: The body may be a disjunction, with operator ||, of several sub-bodies, which are all tried for insertion of the filled version of *H* into the chart. Complement slot rules can also use the infix operator || for disjunctions of the body on the top level, but in this case the || behaves deterministically – as in an if-then-else.

A simple example of a complement slot rule is the following, for the object of a preposition:

```
objprep <
   ri
   (opt n)
   (mpos noun)
   (extmf gen)
   (removemf nom acc)
   satisfied
```

The body is a sequence of tests which are viewed conjunctively. The first test, `ri`, means that the filler *M* is on the "right" of *H* (a postmodifier). The `opt` test checks that the slot option is `n`, requiring an NP. The next test requires that the filler *M* has POS `noun`. In SGF rules, the letter `m` in operators indicates the filler *M* as an implicit operand, and `h` indicates the higher phrase *H*.

The term `(extmf gen)` is an *ext*ending feature test on *M* for the feature `gen` (genitive). This will succeed iff either `gen` is marked explicitly on *M* or *M* has at least one of the BAMA features associated with `gen` in the extending feature rule for `gen` in Arfeas.lx (see Section 2). The test `(removemf nom acc)` always succeeds, and it will remove explicit occurrences of `nom` or `acc` on *M*, as well as any BAMA features associated with those features by extending feature rules.

Finally, the test `satisfied` succeeds iff *M* has no unfilled obligatory complement slots.

The syntax of the SGF formalism is Cambridge Polish (Lisplike), except for the uses of the binary operators < and ||. There are quite a number of "built-in" operators in SGF, and many of them can take any number of arguments.

Tests in SGF can be nested; some operators, including all the logical operators, can contain other tests as arguments. We mentioned that SGF is "object-oriented" in a certain sense. In any given test, however much embedded, there is always a *phrase in focus*, which is an implicit argument of the test. The phrase in focus can be considered like `this` in object-oriented languages. The default phrase in focus on top-level tests is *M* (the modifier). But some operators can shift the focus to another phrase, and this can happen an unlimited number of times in nested tests. For example, a test of the form

(rmod $Test_1$ ... $Test_n$)

searches the postmodifiers of the current phrase in focus and succeeds iff, for one of them as a new phrase in focus, all of the test arguments are satisfied. This scheme allows for quite compact expressions for searching and testing parse trees.

Now let us look at (a modified form of) an adjunct slot rule in Ar.gram, for adjectives that postmodify nouns:

```
adj <
   ri
   (hf noun)
   (agreef nom acc gen)
   (agreef def indef)
   (if (& (exthf pl) (nhf h))
       /* then */
       (extmf sg f)
       /* else */
       (& (agreef sg pl dual)
          (agreef m f) ) )
   satisfied
   (setslot nadj)
   (setopt aj)
```

So the filler *M* should be an adjective phrase. The first two tests check that *M* postmodifies *H*, and *H* is a noun phrase. The main operator here is **agreef**, which works with a list of extending features. The list of features should consist of the possible values of an attribute like case, number, gender, etc. The **agreef** test will succeed iff *M* and *H* agree along this dimension. For at least one of the argument features, both *M* and *H* should have this feature (as an extending feature). Furthermore, **agreef** takes care of reducing feature ambiguity in *M* and *H* (if it succeeds): If *x* is an argument feature such that one of *M* and *H* has *x* (as an extending feature) but the other does not, then *x* is removed from the other (as an extending feature).

For the **adj** rule at hand, the **if** statement can be interpreted as follows: If *H* (the noun) is plural and not human, then *M* (the adjective) must be singular and feminine; otherwise *M* and *H* must agree in number and gender. The actual current rule in Ar.gram skips the agreement test for plural non-human nouns, because we do not currently have enough marking of the human (**h**) features.

For subject-verb agreement, we have the situation that verbs do not use the same extending feature names as nouns do. (This has to do with corresponding BAMA features.) To handle this, **agreef** can take as arguments *pairs* of features, like **(sg vsg)**, where the first element is checked for *M* (the subj noun), and the second is checked for *H* (the verb). Here is a shortened form of the subject slot rule of ASG, which contains the current subject-verb agreement rule for ASG:

```
subj <
  (opt n)
  (mpos noun)
  (if (mf pron)
   /* then */
    (& (agreef (m vm) (f vf))
       (agreef (sg vsg)
               (pl vpl)
               (dual vdual))
       (agreef (pers1 vpers1)
               (pers2 vpers2)
               (pers3 vpers3)) )
    /* else */
    (& (exthf vpers3)
      (if (| (^ (extmf pl)) (mf h))
         (&
           (agreef (m vm) (f vf))
           (if le
              /* subj before verb */
              (agreef (sg vsg)
                      (pl vpl)
                      (dual vdual))
              /*subj after verb: */
              (exthf vsg) ) ) ) )
  )
```

The agreement part is the outer **if** test, and can be interpreted as follows:

1. If *M* is a pronoun, then *M* agrees with *H* in gender, number and person;
2. else *H* must be 3rd-person and if *M* is non-plural or human, then:
    a. *M* agrees with *H* in gender and
    b. if *M* premodifies *H* then it agrees with *H* in number,
    c. else *H* is singular.

This formulation shows the way we are currently ignoring agreement for plural non-human nouns, until we get human markings on nouns.

Now let us illustrate how an adjunct slot rule can overcome a seeming problem for dependency grammars when there is a "missing head word" for a phrase. Consider n the sentence shown in Figure 1, along with its ASG parse tree.

وهذه ظاهرة شائعة جداً قد يسببها الخوف او الاضطرابات المعوية.

```
wh*h ZAhrp $A}Ep jdAF qd ysbbhA Alxwf Aw AlADTrAbAt AlmEwyp.
```

[This is a very common phenomenon, which may be caused by fear or intestinal disorder.]

```
----------------------------------------------------------------------
o----------- top     wa(111,u,1)      noun pron
`----------- rconj   h`*ihi(1)        noun pron
  `--------- npred   ZAhir(2)         noun sg cn def indef nom f
    `------- nadj    $A}iE(3)         adj sg def indef nom acc gen f
    | `----- adjpost jid~(4)          noun cn indef acc qualnoun
    | .----- vadv    qad(5)           adv
    `-+----- nrel    sab~ib(6,8,113)  verb pronobj
      `----- obj(n)  hA(113)          noun pron acc encliticf
      | .--- lconj   xawof(7)         noun cn def nom acc gen
      `-+--- subj(n) Oaw(8,7,9)       noun pl cn def nom acc f
        `--- rconj   {iDoTirAb(9)     noun pl cn def nom acc gen f
          `- nadj    miEawiy~(10)     adj sg def nom acc gen f
----------------------------------------------------------------------
```

Figure 1. Handling a "missing head word"

Here Arabic does without a form of "be". In the ATB, the parse tree shows an S node with three daughters:

```
(S
  (CONJ wa)
  (NP-SBJ
    (DEM_PRON_F h`*ihi))
  (NP-PRD
    (NP (NOUN… ZAhir+ap+N))
      ... )
)
```

Since the ATB does not use a dependency tree scheme, there is no need for a word acting as a verb head of this S.

In ASG we solve the problem of the "missing head word" by letting the "clause" be a nominal phrase with head **h`*ihi** [هٰذِهِ "this"] (this is the subj in the ATB tree), where the predicate NP fills an adjunct slot **npred** of the head NP. Logically, this is not unreasonable, because adjuncts often predicate logically on the phrase they modify. And a predicate NP for a "be" verb can do just that.

The **npred** rule in Ar.gram is as follows (in abbreviated form):

```
noun <
   ri
   (hf noun)
   (exthf nom)
   (extmf nom)
   (^ (mf propn) (hf propn))
   (nhf ri1 num)
   satisfied
   (^ (lmod lconj (rmod nrel)))
   (removehf acc gen)
   (removemf acc gen)
   (setslot npred)
   (setopt n)
```

The rule is indexed under the POS **noun**, since the **npred** filler *M* is an NP. (Actually the **noun** rule has several other disjunctive components, separated by the operator ||, for other ways NPs can modify other phrases as adjuncts.) So this rule requires that *M* postmodifies *H*, *H* is an NP, both *M* and *H* have extending features **nom**, neither *M* nor *H* is a proper noun, *H* has no postmodfiers, and is not a number, and *H* is satisfied. The test

```
(^ (lmod lconj (rmod nrel)))
```

illustrates two focus-shifting operations (see above). This says that it is not the case that *M* has a preconjunct which has a postmodifying relative clause. Finally, the rule removes the extending features **acc** and **gen** from both *H* and *M*, sets the adjunct slot to **npred**, and sets its option to **n**.

The parse in Figure 1 illustrates several other interesting features of Arabic syntax, for instance the resumptive pronoun in the relative clause (adjunct slot **nrel**). And this pronoun is an enclitic, treated by the ASG methods described in Section 4. (The conjunction "wa" in the tree is marked as a noun, because (coordinating) conjunctions in SG inherit features from their conjuncts. In SG, a phrase's features are carried on its head word.)

6 Performance of ASG

Since SG has its own linguistic choices (including being a dependency grammar), it is difficult to measure ASG automatically against the ATB without considerable conversion efforts. We plan to look into comparisons with the Prague Treebank (Hajič et al., 2006), but have not had time yet. The best approach, however, may be to create a treebank that simply uses the ASG design. The SG system has some tools for doing that – using SG parsing as a starter, and hand-correcting the trees.

For the immediate purposes of getting some idea of where ASG currently stands, we did a short measurement (hand-scored) on 20 untrained-on segments from the ATB chosen at random, scoring only the first (highest-ranked) parse for each segment. The scoring consisted of marking each parse tree node *N* for *correctness* of *N* in the sense that *N* has the correct mother node and the correct POS. (The parser does make an assignment of POS and mother for every word/node, even when there is no complete (segment-spanning) parse for the segment.) Note that correctness of all mother nodes implies correct tree shape. With this measurement, the percentage of correct nodes in the test set was 64%.

On 1,000 sentences from ATB3 of length 13 to 20 words, the percentage of complete parses (phrase analyses that span the whole segment) was 72% (with no guarantee of correctness of these parses).

Speed of ASG analysis seems good. On the 1,000 sentences mentioned above, parsing was at the rate of 2,500 words per second (on a laptop). This is with SGF being used in *interpreted* mode. There is a compiler for SGF (compiling *X*.gram to a C program) that provides about a twofold speedup for syntactic analysis, although the compiler is not currently up-to-date with the latest set of operators for SGF.

For the morpholexical processing part of analysis, the rate was 10,000 words per second. This includes look-up and morphology in BAMA, and look-up in Ar.lx – the complete morpholexical process.

7 Related Work

Surprisingly little information is available regarding existing Arabic parsers and their performance, though some commercial parsers must exist. Until very recently, the focus of published research for Arabic NLP has been on low-level forms of processing, including morphological analysis, part-of-speech tagging, automatic diacriticization, and named entity transliteration; and frequently the term "parsing" in the context of Semitic languages refers to morphological and not syntactic parsing.

One symbolic approach to parsing Arabic (Othman et al., 2003, 2004) uses a unification-based grammar formalism and a chart parser implemented in Prolog. Information in the lexicon on "subject rationality" and "object rationality" is combined with "rationality" features on head nouns and noun phrases to eliminate some of the choices proposed by the morphological analyzer. No information is provided regarding the coverage of the grammar or the performance of the parser.

More performance data is available for two related statistical parsers trained on Arabic treebank data. Bikel's (2004) implementation of the Collins (2003) parser, trained on the Arabic TreeBank 1 (ATB1), reached recall/precision = 75.4/76.0 on sentences of 40 words or less and 72.5/73.4 on all sentences. Kulick et al. (2006) used the Bikel parser on a revised version of the ATB1 with results comparable to Bikel, and then on ATB3, where initial performance dropped slightly. A number of successive improvements allowed the parser to achieve recall/precision = 78.14/80.26 on sentences of 40 words or less and 73.61/75.64 on all sentences. The two most substantial improvements were obtained by changing the handling of punctuation and choosing a tagset that preserves a bit more information than the severely reduced one distributed with the ATB segments.

Other statistical parsers that have been used with Arabic include one trained on a segment of the Prague Arabic Dependency TreeBank (Hajič et al.,

2004) and then used to assist in the annotation of the remainder, but little seems to be published about its performance. The Stanford Parser has been used with Arabic (http://nlp.stanford.edu/downloads/ lex-parser.shtml), but no specific performance information could be found. It is based on the ideas that there are advantages in factoring out the phrase structure tree and the lexical dependency tree models and estimating them separately, and that significant improvements can be achieved without including any lexical dependency information by adding a few linguistically motivated annotations to phrase structure tree models (Klein and Manning, 2002, 2003).

Finally Chiang et al. (2006) used both Bikel's (2002) and Chiang's (2000) parsers to develop different approaches to parsing text in Levantine Arabic based on the Arabic Treebank data.

Even less information was found for parsing of other Semitic Languages (with the exception of http://www.cs.technion.ac.il/~winter/Corpus-Project/project-description.html) and Wintner's (1998) discussion of Hebrew syntax form a computational perspective. However, while the authors are not very familiar with this language, known similarities with Arabic give us reason to believe that some of our work on ASG could be readily reusable for Hebrew SG.

References

Daniel M. Bikel. 2002. Design of a multi-lingual, parallel processing statistical parsing engine. In *Proceedings of International Conference on Human Language Technology Research* (HLT).

Daniel M. Bikel. 2004. *On the Parameter Space of Lexicalized Statistical Parsing Models*. PhD thesis, Department of Computer and Information Sciences, University of Pennsylvania.

Tim Buckwalter. 2002. *Arabic Morphological Analyzer Version 1.0.* Linguistic Data Consortium catalog number LDC2002L49, ISBN 1-58563-257-0.

David Chiang. 2000. Statistical parsing with an automatically-extracted tree adjoining grammar. *In Proceedings of the 38th Meeting of the Association for Computational Linguistics (ACL'00),* Hong Kong, China, 456–463.

David Chiang, Mona Diab, Nizar Habash, Owen Rambow, and Safiullah Sharif. 2006. Parsing Arabic Dialects. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, Trento, Italy, 369–376.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29:589–637.

Jan Hajič, Otakar Smrž, Petr Zemánek, Jan Šnaidauf, and Emanuel Beška. 2004. Prague Arabic Dependency Treebank: Development in Data and Tools. *In Proceedings of NEMLAR 2004*.

Jan Hajič et al. 2006. *Prague Dependency Treebank Version 2.0*. Linguistic Data Consortium catalog number LDC2006T01, ISBN 1-58563-370-4.

Seth Kulick, Ryan Gabbard, and Mitchell Marcus. 2006. Parsing the Arabic Treebank: Analysis and Improvements. In Hajič J. and Nivre, J. (eds.): *Proceedings of the TLT 2006,* pp. 31-42. Institute of Formal and Applied Linguistics, Prague, Czech Republic.

Dan Klein and Christopher D. Manning. 2002. Fast Exact Inference with a Factored Model for Natural Language Parsing. In *Advances in Neural Information Processing Systems* 15 (NIPS 2002).

Dan Klein and Christopher D. Manning. 2003. Accurate Unlexicalized Parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*.

Michael C. McCord. 1980. Slot Grammars. *Computational Linguistics*, 6:31-43.

Michael C. McCord. 1993. Heuristics for Broad-Coverage Natural Language Parsing. In *Proceedings of the ARPA Human Language Technology Workshop*. Morgan-Kaufmann, 127-132.

Michael C. McCord. 2006. *A Formal System for Slot Grammar*. Technical Report RC 23976, IBM T.J. Watson Research Center.

E Othman, K Shaalan, A Rafea. 2003. A Chart Parser for Analyzing Modern Standard Arabic Sentence. In *Proceedings of the MT Summit IX Workshop on Machine Translation*.

E Othman, K Shaalan, and A Rafea. 2004. Towards Resolving Ambiguity in Understanding Arabic Sentences. In *Proceedings of NEMLAR 2004.*

Shuly Wintner. 1998. Towards a linguistically motivated computational grammar for Hebrew. In *Proceedings of the ACL-98 Workshop on Computational Approaches to Semitic Languages*, 82-88.