From Prosodic Trees to Syntactic Trees

Andi Wu
GrapeCity Inc.
andi.wu@grapecity.com

Kirk Lowery
Westminster Hebrew Institute
klowery@whi.wts.edu

Abstract

This paper describes an ongoing effort to parse the Hebrew Bible. The parser consults the bracketing information extracted from the cantillation marks of the Masoetic text. We first constructed a cantillation treebank which encodes the prosodic structures of the text. It was found that many of the prosodic boundaries in the cantillation trees correspond, directly or indirectly, to the phrase boundaries of the syntactic trees we are trying to build. All the useful boundary information was then extracted to help the parser make syntactic decisions, either serving as hard constraints in rule application or used probabilistically in tree ranking. This has greatly improved the accuracy and efficiency of the parser and reduced the amount of manual work in building a Hebrew treebank.

Introduction

The text of the Hebrew Bible (HB) has been carefully studied throughout the centuries, with detailed lexical, phonological and morphological analysis available for every verse of HB. However, very few attempts have been made at a verse-by-verse syntactic analysis. The only known effort in this direction is the Hebrew parser built by George Yaeger (Yaeger 1998, 2002), but the analysis is still incomplete in the sense that not all syntactic units are recognized and the accuracy of the trees are yet to be checked.

Since a detailed syntactic analysis of HB is of interest to both linguistic and biblical studies, we launched a project to build a treebank of the Hebrew Bible. In this project, the trees are automatically generated by a parser and then manually checked in a tree editor. Once a tree has been edited or approved, its phrase boundaries are recorded in a database. When the same verse is parsed again, the existing brackets will force the parser to produce trees whose brackets are exactly the same as those of the manually approved trees. Compared with traditional approaches to treebanking where the correct structure is preserved in a set of tree files, our approach has much more agility. In the event of design/format changes, we can automatically regenerate the trees according to the new specifications without manually touching the trees. The bracketing information will persist through the updates and the basic structure of the trees will remain correct regardless of the changes in the details of trees. We call this a "dynamic treebank" where, instead of maintaining a set of trees, we maintain a parser/grammar, a dictionary, a set of sentences, and a database of bracketing information. The trees can be generated at any time.

Since our parser/grammar can consult known phrase boundaries to build trees, its performance can be greatly improved if large amounts of bracketing information are available. Human inspection and correction can provide those boundaries, but the amount of manual work can be reduced significantly if there is an existing source of bracketing information for us to use. Fortunately, a great deal of such information can be obtained from the cantillation marks of the Hebrew text.

1 The cantillation treebank

1.1 Cantillation marks

The text of HB has been systematically annotated for more than a thousand years. By the end of the 9th century, a group of Jewish scholars known as the Masoretes had developed a system for

Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions, pages 898–904,
Sydney, July 2006.

marking the structures of the Bible verses. The system contains a set of cantillation marks [1] which indicate the division and subdivision of each verse, very much like the punctuation marks or the brackets we use to mark constituent structures. At that time, those cantillation marks were intended to record the correct way of reading or chanting the Hebrew text: how to group words into phrases and where to put pauses between intonational units. In the eyes of modern linguists, the hierarchical structures thus marked can be best understood as a prosodic representation of the verses (Dresher 1994).

There are two types of cantillation marks: the conjunctive marks which group multiple words into single units and the disjunctive marks which divide and subdivide a verse in a binary fashion. The marking of Genesis 1:1, for example, is equivalent to the bracketing shown below. (English words are used here in place of Hebrew to make it easier for non-Hebrew-speakers to understand. OM stands for object marker.)

```
( ( ( In beginning )
    ( created God )
  )
  ( ( OM
      ( the heavens )
    )
    (  ( and OM )
       ( the earth )
    )
  )
)
```

This analysis resembles the prosodic structure in Selkirk (1984) and the performance structure in Gee and Grosjean (1983).

1.2. Parsing the prosodic structure

The cantillation system in the Mesoretic text is a very complex one with dozens of diacritic symbols and complicated annotation rules. As a result, only a few trained scholars can decipher them and their practical use has been very limited. In order to make the information encoded by this system more accessible to both humans and machines, we built a treebank where the prosodic structures of HB verses are explicitly represented as trees in XML format (Wu & Lowery, 2006).

There have been quite a few studies of the Masoretic cantillation system. After reviewing the existing analyses, such as Wickes (1881), Price (1990), Richter (2004) and Jacobson (2002), we adopted the binary analysis of British and Foreign Bible Society (BFBS 2002) which is based on the principle of dichotomy of Wicks (1881). The binary trees thus generated are best for extracting brackets that are syntactically significant.

We found all the binary rules that underlie the annotation and coded them in a context-free grammar. This CFG was then used by the parser to automatically generate the prosodic trees. The input text to the parser was the MORPH database developed by Groves & Lowery (2006) where the the cantillation marks are represented as numbers in its Michigan Code text.

The following is the prosodic tree generated for Genesis 1:1, displayed in English glosses in the tree editor (going right-to-left according to the writing convention of Hebrew):

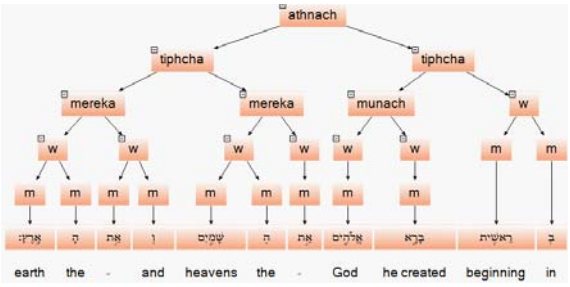

Figure 1

The node labels “athnach”, “tiphcha”, “mereka” and “munach” in this tree are names of the cantillation marks that indicate the types of boundaries between the two chunks they dominate. Different types of boundaries have different (relative) boundary strengths. The “m” nodes are morphemes and the “w” nodes are words.

[1] The cantillation marks show how a text is to be sung. See http://en.wikipedia.org/wiki/Cantillation.

1.3. A complete prosodic treebank

Since the Mesoretic annotation is supposed to mark the structure of every verse unambiguously, we expect to parse every verse successfully with exactly one tree assigned to it, given that (1) the annotation is perfectly correct and (2) the CFG grammars correctly encoded the annotation rules. The actual results were close to our expectation: all the 23213 verses were successfully parsed, of which 23099 received exactly one complete tree. The success rate is 99.5 percent. The 174 verses that received multiple parse trees all have words that carry more than one cantillation mark. This can of course create boundary ambiguities and result in multiple parse trees. We have good reasons to believe that the grammars we used are correct. We would have failed to parse some verses if the grammars had been incomplete and we would have gotten multiple trees for a much greater number of verses if the grammars had been ambiguous.

2 Phrase boundary extraction

Now that a cantillation treebank is available, we can get brackets from those trees and use them in syntactic parsing. Although prosodic structures are not syntactic structures, they do correspond to each other in some systematic ways. Just as there are ways to transform syntactic structures to prosodic structures (e.g. Abney 1992), prosodic structures can also provide clues to syntactic structures. As we have discovered, some of the brackets in the cantillation trees can be directly mapped to syntactic boundaries, some can be mapped after some adjustment, and some have no syntactic significance at all.

2.1 Direct correspondences

Direct correspondences are most likely to be found at the clausal level. Almost all the clause boundaries can be found in the cantillation trees. Take Genesis 1:3 as an example:

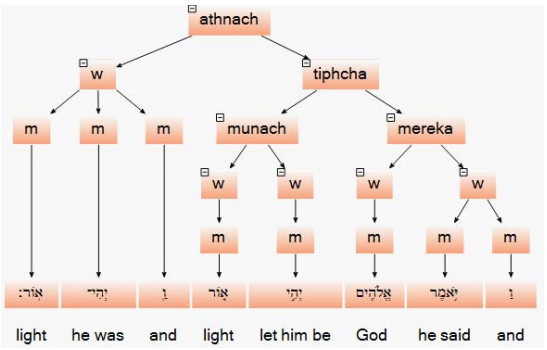

Figure 2

Here, the verse is first divided into two clauses: “God said let there be light” and “there was light”. The first clause is further divided into “God said” and “let there be light”. Such bracketing will prevent the wrong analysis where “let there be light” and “there was light” are conjoined to serve as the object of “God said”. Given the fact that there are no punctuation marks in HB, it is very difficult for the parser to rule out the wrong parse without the help of the cantillatioin information.

Coordination is another area where the cantillation brackets are of great help. The syntactic ambiguity associated with coordination is well-known, but the ambiguity can often be resolved with help of prosodic cues. This is indeed what we find in the cantillation treebank. In Genesis 24:35, for example, we find the following sequence of words: “male servants and maid servants and camels and donkeys”. Common sense tells us that there are only two possible analyses for this sequence: (1) a flat structure where the four NPs are sisters, or (2) “male servant” conjoins with “maid servant, “camels” conjoins with “donkeys”, and then the two conjoined NPs are further conjoined as sisters. However, the computer is faced with 14 different choices. Fortunately, the cantillation tree can help us pick the correct structure:

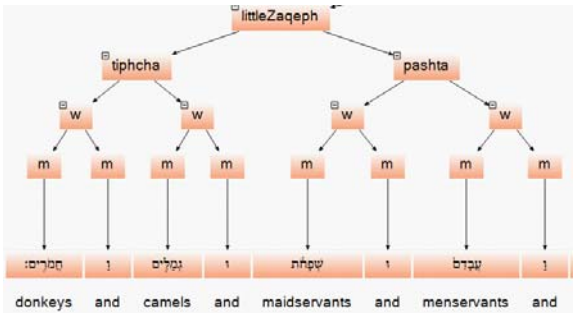

Figure 3

The brackets extracted from this tree will force the parser to produce only the second analysis above.

Good correspondences are also found for most base NPs and PPs. Here is an example from Genesis 1:4, which means "God separated the light from the darkness":

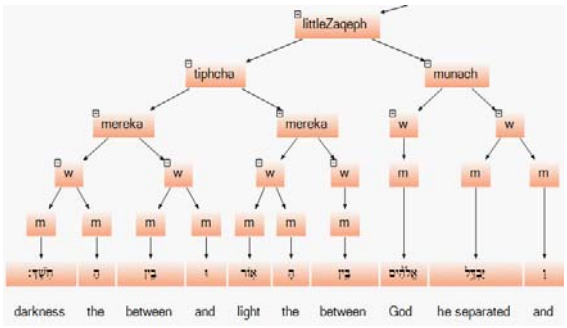

Figure 4

As we can see, the noun phrases and prepositional phrases all have corresponding brackets in this tree.

2.2 Indirect correspondences

Now we turn to prosodic structures that can be adjusted to correspond to syntactic structures. They usually involve the use of function words such as conjunctions, prepositions and determiners. Syntactically, these words are supposed to be attached to complete NPs, often resulting in trees where those single words are sisters to large NP chunks. Such "unbalanced" trees are rarely found in prosodic structures, however, where a sentence tends to be divided into chunks of similar length for better rhythm and flow of speech.

This is certainly the case in the HB cantillation treebank. It must have already been noticed in the example trees we have seen so far that the conjunction "and" is always attached to the word that immediately follows it. As a matter of fact, the conjunction and the following word are often treated as a single word for phonological reasons.

Prepositions are also traditionally treated as part of the following word. It is therefore not a surprise to find trees of the following kind:

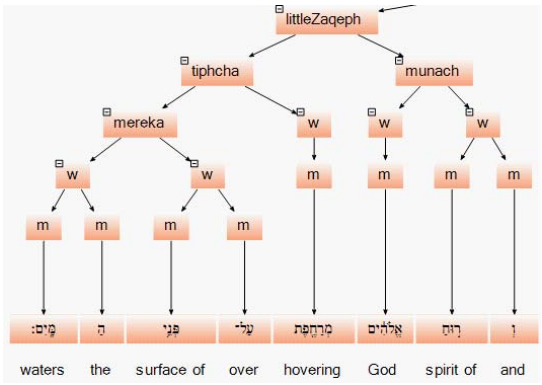

Figure 5

In this tree, the preposition "over" is attached to "surface of" instead of "surface of the waters". We also see the conjunction "and" is attached to "spirit of" rather than to the whole clause.

A similar situation is found for determiners, as can be seen in this sub-tree where "every of" is attached to "crawler of" instead of "crawler of the ground".

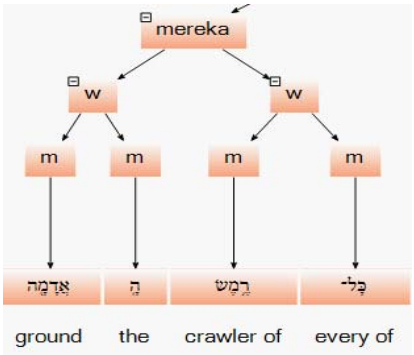

Figure 6

In all these cases, the differences between prosodic structures and syntactic structures are systematic and predictable. All of them can be adjusted to correspond better to syntactic structures by raising the function words out of their current positions and re-attach them to some higher nodes.

2.3 Extracting the boundaries

In the bracket extraction phase, we go through all the sub-trees and get their beginning and ending positions in the form of *(begin, end)*. Given the tree in Figure 6, for example, we can extract the following brackets: (*n*, *n+3*), (*n*, *n+1*), (*n+2*, *n+3*), where *n* is the position of the first word in the sub-tree.

For cases of indirect correspondence discussed in 2.2, we automatically adjust the brackets by removing the ones around the function word and its following word and adding a pair of brackets that start from the word following the function word and end in the last word of the phrase. After this adjustment, the brackets extracted from Figure 6 will become (*n*, *n+3*), *(n+1*, *n+3*) and (*n+2*, *n+3*). This in effect transforms this tree to the one in Figure 7 which corresponds better to its syntactic structure:

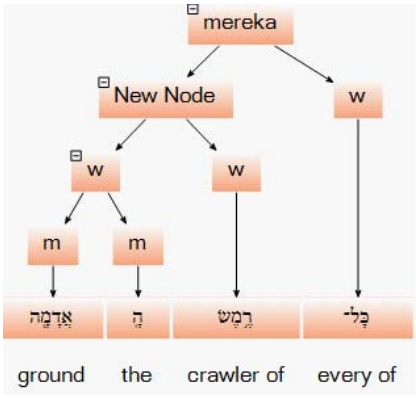

Figure 7

For trees that start with “and”, we detach “and” and re-attach it to the highest node that covers the phrase starting with “and”. After this and other adjustments, the brackets we extract from Figure 5 will be:

(*n*, *n+7*)
(*n+1*, *n+7*)
(*n+1*, *n+2*)
(*n+3*, *n+7*)
(*n+4*, *n+7*)
(*n+5*, *n+7*)
(*n+6*, *n+7*)

These brackets transform the tree into the one in Figure 8:

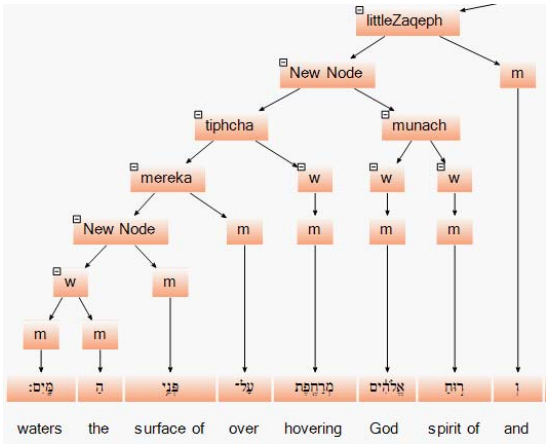

Figure 8

The cantillation trees also contain brackets that are not related to syntactic structures at all. Since it is difficult to identify those useless brackets automatically, we just leave them alone and let them be extracted anyway. Fortunately, as we will see in the next section, the parser does not depend completely on the extracted bracketing information. The useless brackets can simply be ignored in the parsing process.

3 Building a syntactic treebank

As we mentioned earlier, we use a parser to generate the treebank. This parser uses an augmented context-free grammar that encodes the grammatical knowledge of Biblical Hebrew. Each rule in this grammar has a number of grammatical conditions which must be satisfied in order for the rule to apply. In addition, it may have a bracketing condition which can either block the application of a rule or force a rule to apply.

Besides serving as conditions in rule application,

the bracketing information is also used to rank trees in cases where more than one tree is generated.

3.1 Brackets as rule conditions

Bracketing information is used in some grammar rules to guide the parser in making syntactic decisions. In those rules, we have conditions that look at the beginning position and ending position of the sub-tree to be produced by the rule and check to see if those bracket positions are found in our phrase boundary database. The sub-tree will be built only if the bracketing conditions are satisfied.

There are two types of bracketing conditions. One type serves as the necessary *and* sufficient condition for rule application. These conditions work in *disjunction* with grammatical conditions. A rule will apply when either the grammatical conditions or the bracketing conditions are satisfied. This is where the bracketing condition can force a rule to apply regardless of the grammatical conditions. The brackets consulted by this kind of conditions must be the manually approved ones or the automatically extracted ones that are highly reliable. Such conditions make it possible for us to override grammatical conditions that are too strict and build the structures that are known to be correct.

The other type of bracketing conditions serves as the necessary conditions only. They work in *conjunction* with grammatical conditions to determine the applicability of a rule. The main function of those bracketing conditions is to block structures that the grammatical conditions fail to block because of lack of information. However, they cannot force a rule to apply. The sub-tree to be produced the rule will be built only if both the grammatical conditions and the bracketing conditions are met.

The overall design of the rules and conditions are meant to build a linguistically motivated Hebrew grammar that is independent of the cantillation treebank while making use of its prosodic information.

3.2 Brackets for tree ranking

The use of bracketing conditions greatly reduces the number of trees the parser generates. In fact, many verses yield a single parse only. However, there are still cases where multiple trees are generated. In those cases, we use the bracketing information to help rank the trees.

During tree ranking, the brackets of each tree are compared with the brackets in the cantillation trees to find the number of mismatches. Trees that have fewer mismatches are ranked higher than trees that have more mismatches. In most cases, the top-ranking tree is the correct parse.

Theoretically, it should be possible to remove all the bracketing conditions from the rules, let the parser produce all possible trees, and use the bracketing information solely at the tree-ranking stage to select the correct trees. We can even use machine learning techniques to build a statistical parser. However, a Treebank of the Bible requires 100% accuracy but none of the statistical models are capable of that standard yet. As long as 100% accuracy is not guaranteed, manual checking will be required to fix all the individual errors. Such case-by-case fixes are easy to do in our current approach but are very difficult in statistical models.

3.3 Evaluation

Since only a very small fraction of the trees generated by our parser have been manually verified, there is not yet a complete golden standard to objectively evaluate the accuracy of the parser. However, some observations are obvious:

(1) The parsing process can become intractable without the bracketing conditions. We tried parsing with those conditions removed from the rules to see how many more trees we will get. It turned out that parsing became so slow that we had to terminate it before it was finished. This shows that the bracketing conditions are playing an indispensable role in making syntactic decisions.

(2) The number of edits needed to correct the trees in manual checking is minimal. Most trees generated by the machine are basically correct and only a few touches are necessary to make them perfect.
(3) The boundary information extracted from the cantillation tree could take a long time to create if done by hand, and a great deal of manual work is saved by using the brackets from the cantillation treebank.

Conclusion

In this paper, we have demonstrated the use of prosodic information in syntactic parsing in a treebanking project. There are correlations between prosodic structures and syntactic structures. By using a parser that consults the prosodic phrase boundaries, the cost of building the treebank can be minimized.

References

Abney, S (1992) Prosodic Structure, Performance Structure and Phrase Structure. In *Proceedings, Speech and Natural. Language Workshop*, pp. 425-428.

BFBS (2002) The Masoretes and the Punctuation of Biblical Hebrew. British & Foreign Bible Society, Machine Assisted Translation Team.

Dresher, B.E. (1994) The Prosodic Basis of the Tiberian Hebrew System of Accents. In *Language*, Vol. 70, No 1, pp. 1-52.

Gee J. P. & F. Grosjean (2002) The Masoretes and the Punctuation of Biblical Hebrew. British & Foreign Bible Society, Machine Assisted Translation Team.

Groves, A & K. Lowery, eds. (2006). *The Westminster Hebrew Bible Morphology Database*. Philadelphia: Westminster Hebrew Institute.

Jacobson, J.R. (2002) *Chanting the Hebrew Bible*. The Jewish Publication Society, Philadelphia..

Price, J. (1990) *The Syntax of Masoretic Accents in the Hebrew Bible*. The Edwin Mellen Press, Lewiston/Queenston/Lampeter.

Richter, H (2004) Hebrew Cantillation Marks and Their Encoding. Published at http://www.lrz-muenchen.de/~hr/teamim/

Selkirk, E. (1984) *Phonology and Syntax: The Relation between Sound and Structure*. Cambridge, MA: MIT Press.

Wickes, W. (1881) *Two Treatises on the Accentuation of the Old Testament*. Reprint by KTAV, New York, 1970..

Wu, A & K. Lowery (2006) A Hebrew Tree Bank Based on Cantillation Marks. In *Proceedings of LREC 2006*.

Yaeger, G (1998) Layered Parsing: a Principled Bottom-up Parsing Formalism for Classical Biblical Hebrew, a working paper, ASTER Institute, Point Pleasant, NJ.

Yaeger, G (2002) A Layered Parser Implementation of a Schema of Clause Types in Classical Biblical Hebrew, SBL Conference, Toronto, Ontario, Canada.