# The *Xavier* Module – Information Processing of Treebanks

**Sanghoun Song*, Jieun Jeon**

Department of Linguistics, Korea University, Anam-dong Sungbuk-gu SEOUL South Korea
*Correspondence should be addressed to yooseon21@korea.ac.kr

**Abstract**

This paper aims to introduce the *Xavier* module, a program package to process Treebanks (in particular, the *Sejong Korean Treebank*). In this paper, the procedure of implementing *Xavier* is discussed, and main usage of the program is also provided. Though this paper focuses on the *Sejong Korean Treebank*, *Xavier* is also applicable to other Treebanks, such as the *Penn Treebanks*, because it has been designed data-independently. In addition, *Xavier* is freely available, so those who want to study linguistic phenomena based on Treebanks can take advantage of *Xavier*.

## The Goal

*Xavier* is designed for extracting automatically linguistic information from Treebanks. Language resources, such as Treebanks, have become a greater priority in the study of language as well as in the implementing natural language processing system. Although there are so many resources, tools for the resources are relatively small number. Currently available tools for the *Sejong Korean Treebank* (henceforth SKT), such as *Hanmaru*[1], are limited in its functions. For example, *Hanmaru* cannot extract the whole CFG (Context Free Grammar) rules from treebanks so that field workers cannot offer a comprehensive explanation of characteristics of a language in treebanks. Besides, *Hanmaru* which extracts concordance only at the level of one phrase cannot apply context information to the concordance search easily. Therefore, in order to extract large scale data from the Treebanks, a more powerful tool is necessary. *Xavier* can furnish the research environment to those who need a general, multi-purpose extraction tool.

[1] *Hanmaru* has been implemented as a basic search program for the *Sejong* corpora, such as POS-tagged corpora, treebanks, etc. This program also makes use of the triangle-structure in order to search treebanks. If field workers want to find out simple examples from treebanks, *Hanmaru* may provide a good solution to the purpose. For more information about *Hanmaru*, visit the relevant homepage (http://www.sejong.or.kr).

## Implementation

*Xavier* is coded in the ANSI C++ language, thus is very fast in its execution speed. *Xavier* processes data in Treebanks on the basis of the Parse-Tree algorithm. Data structure of the Parse-Tree algorithm consists of three elements; the mother node (henceforth MN), the left daughter node (henceforth LDN), and the right daughter node (henceforth RDN). The Figure 1 represents a typical Parse-Tree structure; the first S is the MN of its LDN AP and its RDN S. The second S is also the MN of NP_SBJ and VP at the same time. In brief, every node is linked to the head node in the hierarchical binary form.

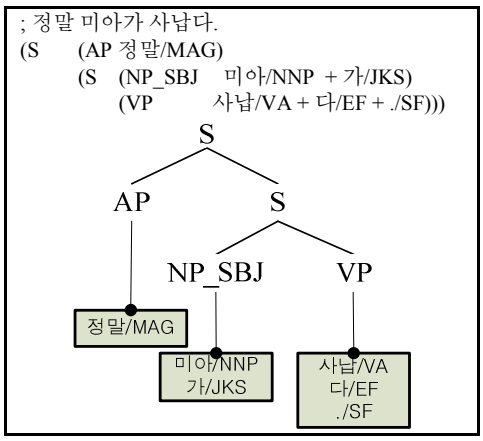

Figure 1: A sample of Parse-Tree

The algorithm which builds up a Parse-Tree is given in the below Algorithm 1. This algorithm builds up a Parse-Tree structure, node by node. If there is a new node (`n` represents the new node in the below) which is not yet processed (line 1), first of all, both the LDN of the node and the RDN of

the node are assigned a NULL value (line 2). If the node is not a terminal node, which belongs to a lexical unit (e.g. VP → 사납/VA+다/EF+.SF in Figure 1), the RDN of the node and the LDN of the node are assigned a value which popped from the stack[2] in order (line 4 and 5). Since there can be a node without its RDN (line 6), in this case, the algorithm substitutes the LDN with the RDN (line 7) and assigns a NULL value to the RDN (line 8). Lastly, this algorithm pushes the node processed so far into the stack (line 9) in order to link with other nodes.

```
1: parse_tree(n):
2:   n→left = n→right = NIL:
3:   if n is not a terminal node:
4:     n→right = pop()
5:     n→left = pop()
6:     if n→left is NIL:
7:       n→left = n→right
8:       n→right = NIL
9:   push(n)
```

Algorithm 1: Parse-Tree

We made each component of *Xavier* with a separate module so that it can be reused for other purposes.

## The Specification of *Xavier* ver. 1.0

Major functions of *Xavier* (ver. 1.0) are as follows.

### Concordance

First of all, *Xavier* extracts concordances that a user wants to find out, based on triangle-table structure. The noticeable feature of *Xavier*, which is distinct from the other concordance program, is that *Xavier* provides context-based extraction. For instance, the rewriting rules 'S→AP S' and 'S→NP_SBJ VP' are applied to Figure 1 in the above. If a user wants to search the whole sentence in which sentential adverbs appear and the main clause has an overt subject, *Xavier* allows him or her to extract context-based concordance, such as '[S→AP [S→NP_SBJ VP]]', from Treebanks. Moreover, *Xavier* can also handle embedded triangle-table structure, such as '[S→[NP_SBJ→VP_MOD NP_SBJ] [VP→VP VP]]', which would offer researchers more chances to put Treebanks in practical use.

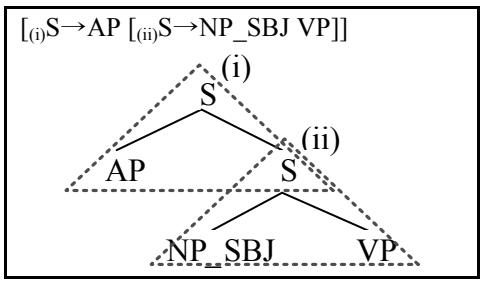


Figure 2: Overlapped triangle-table structures

### CFG Rules

*Xavier* also can function as an automatic analyzer of Treebanks; *Xavier* extracts the CFG rules in SKT.

Table 1: Top 5 of CFG rules

| CFG | Freq. | Prop. |
|---|---|---|
| S→NP_SBJ VP | 49086 | 6.34% |
| VP→VP VP | 46057 | 5.95% |
| VP→NP_OBJ VP | 39664 | 5.12% |
| VP→NP_AJT VP | 36926 | 4.77% |
| VP_MOD→NP_AJT VP_MOD | 27819 | 3.59% |

### Subcategorization Frames

*Xavier* can build up a machine-readable dictionary of the subcategorization frames automatically. The lexical information of verbal items, such as verbs or adjectives, plays an important role in syntactic parsing, because the structure of a sentence mainly hinges on the type of verbal items. Therefore, how to acquire information about verbal items has significant relation to the grammatical study. However, it falls under painstaking work to describe linguistic information about verbal items, such as subcategorization frames, only by hand. Therefore, it is necessary to build up linguistic information about verbal items automatically on the basis of corpora (Manning 1993, Sarkar and Zeman 2000). *Xavier* can extract subcategorization frames of verbal items from treebanks in a statistical and unsupervised way.

[2] Technically speaking, the Parse-Tree algorithm is grounded upon a stack on the principle of Last In First Out (LIFO). The stack has two basic operations; push and pop. The former adds a new node to the last of the stack, and the latter removes and returns the last node of the stack.

### *Xavier* as a Transducer

In addition, *Xavier* can be used as a transducer which converts one annotation-styled linguistic data into the other styled data automatically; for instance, from the *Sejong Korean Treebank*'s annotation to the *Penn Korean Treenbank* (hereafter PKT)'s annotation, and vice versa. Although both of them treat Korean data, their style of annotation are different from each other, respectively. For instance, the empty categories of a sentence are specified in PKT, while there is no empty category in SKT. To take one more example, oblique cases can be tagged as a complement of verbs or adjectives in PKT, whereas SKT in which stringent criteria are applied to distinguish a complement does not cover oblique cases by and large. In order to convert one-styled annotation into the other-styled annotation, we employed about three thousand conversion rules, and introduced some heuristic approaches into the procedure of transduction.

## Evaluation

In the case of speed, *Xavier* can process approximately six thousand of sentences per second in the condition of CPU 1.8Ghz and RAM 512MB, which is equivalent to an official personal computer as of May, 2008. That means the speed of *Xavier* is relatively fine.

In order to check the feasibility of *Xavier*, we compared the result of *Xavier* with the result of *Hanmaru* and the result by hand. The procedure of evaluation is divided into two methods; 'Precision' and 'Recall'. 'Precision' is a measure to represent how much the result is correct. On the other hand, 'Recall' is a measure to prove how the result is extracted completely.

At first, we extracted some concordances from treebanks using *Xavier* as well as *Hanmaru* in order to compare the results with each other. This comparison is for the so-called 'Precision' of *Xavier*. Next, we selected a subset of the whole treebanks, and counted the frequency of some specific categories. We compare this measure with the result of *Xavier*. This comparison is for the so-called 'Recall' of *Xavier*.

The related formulas presented in Manning and Schütze (1999) are given below.

$$precision = \frac{tp}{tp + fp}$$

Formula 1: Precision

$$recall = \frac{tp}{tp + fn}$$

Formula 2: Recall

In accordance with these formulas, the Precision of *Xavier*'s search is 100%, and the Recall is 99.5%.

## Using *Xavier*

The *Xavier* module will be made available on the Internet. *Xavier* ver. 1.0 will be distributed in the form of DLL (Dynamic Linked Library); therefore, this module can be imported in the various programming languages, such as C/C++, Perl, or Python, and can be embedded in other software.

An interface for embedding in the C/C++ language is given in Figure 3. For example, if a user wants to extract the whole CFG rules from treebanks, the user have to import the first function into his or her code. If the user links his or her own code with the *Xavier* DLL properly, it works. A sample code for the C/C++ programming language is also given in Figure 4.

As for other programming languages, such as Perl, Python, or JAVA, we made use of the so-called SWIG (Simplified Wrapper and Interface Generator) program. SWIG is a software development tool that generates wrapper code to make C/C++ code accessible from other languages. (http://www.swig.org) Thus, SWIG connects programs written in C/C++ with programming languages such as Perl, Python, JAVA, PHP, etc.

```
__declspec(dllexport) bool GetCFG
        (char  files[256][256], const char * output_prefix);
__declspec(dllexport) bool GetConcordance
        (char  files[256][256],  const char * output_prefix ,
         const char * mn, const char * ldn, const char * rdn);
… … …
```

Figure 3: The interface of *Xavier* DLL (XavierDllExport.h)

```
#include "XavierDllExport.h"
void main(void)
{
    char files[256][256] = {"BGAA0001.txt", "BGAA0164.txt"};
    const char * output_prefix = "output_";
    GetCFG(files,  output_prefix );
}
```

Figure 4: Using *Xavier* in the C/C++ language

```
%module XavierSwig
%{
#include "XAVIER.h"
extern int GetCFG(const char * input, const char * output);
%}
extern int GetCFG(const char * input, const char * output);
```

Figure 5: Interface Declaration of *Xavier*

```
use XavierSwig;
XavierSwig::GetCFG("BGAA0001.txt", "output.txt");
```

Figure 6: Usage Example for Perl

```
import XavierSwig
XavierSwig.GetCFG("BGAA0001.txt", "output.txt")
```

Figure 7: Usage Example for Python

Using SWIG, we could provide a way to take advantage of *Xavier*, irrespective of user's programming environments. Figure 5 refers to interface declaration of *Xavier* for the wrapper of SWIG.[3] Usage examples for Perl or Python are also given in Figure 6 and 7.[4]

There is also a standalone version which is linked with the above *Xavier* DLL, which is now in the style of simple console program. The more convenient standalone version, such as a program with GUI interface, will be provided as well in future releases.

## Acknowledgments

We would like to return thanks to Prof. Jae-Woong Choe, who helped this study forward. We also want to appreciate the comments of anonymous readers. Of course, all errors are our responsibility.

[3] For more information about using SWIG, see the development documentation of SWIG.

[4] If a user wants to make use of the *Xavier* module wrapped by SWIG, he or she should embed the proper *Xavier* module into the corresponding library folder. The release version of *Xavier* will include the relevant documentation.

## References

Choe, J., Song S., & Jeon, J. (2008) Probabilistic Context-Free Grammar Rules based on Sejong Korean Treebank. *Language and Information*, *9*, 87-139.

Manning, C. D. & Schütze, H. (1999) *Foundations of Statistical Natural Language Processing*. Cambridge: The MIT Press.

Manning, C. D. (1999) Automatic acquisition of a large subcategorization dictionary from corpora. *Proceedings of the 31st annual meeting on Association for Computational Linguistics* (pp. 235-242). Columbus, Ohio: ACL.

Sarkar, A. & Zeman, D. (2000) Automatic Extraction of Subcategorization Frames for Czech. *Proceedings of the COLING-2000* (pp. 691-697). Saarbrücken , Germany.