Tool for robust stochastic parsing using optimal maximum coverage [*]

Vladimír Kadlec [1] and Jean-Cédric Chappelier [2] and Martin Rajman [2]
[1] Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic;
[2] I&C - IIF, EPFL, 1015 Lausanne, Switzerland
E-mail: `xkadlec@fi.muni.cz`,
`{jean-cedric.chappelier,martin.rajman}@epfl.ch`

December 23, 2004

Abstract

This report presents a robust syntactic parser that is able to return a "correct" derivation tree even if the grammar cannot generate the input sentence. The following two steps solution is proposed: the finest corresponding most probable optimal maximum coverage is generated first, then the trees from this coverage are glued into one resulting tree. We discuss the implementation of this method with the SLP toolkit and `libkp` library.

There are many NLP applications (e.g. with speech recognition or dialog systems) where it is difficult to find a context free grammar (CFG) that generates a sufficient subset of the processed language (under-generation problem). In addition, when the coverage of the grammar is improved, the accuracy usually decreases. Therefore our goal is to to develop a robust syntactic parser that is able to return a "correct" derivation tree even if the

[*]This report was also submitted as Technical Report No. FIMU-RS-2005-05 at the Faculty of Informatics, Masaryk University, Brno

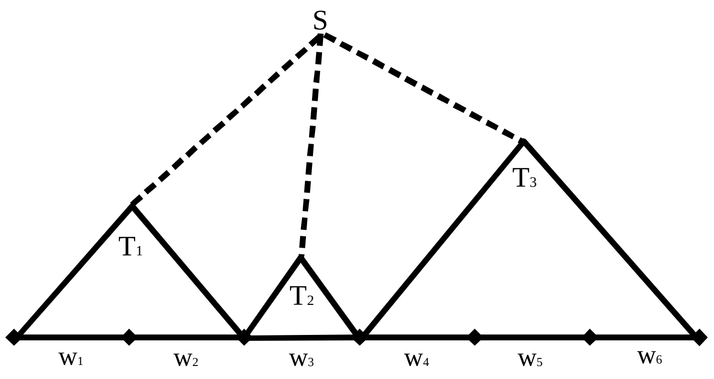


Figure 1: Glued trees.

grammar cannot generate the input sentence. The definition of correctness is strongly dependent on the target application and our framework allows to change the correctness criteria to fit various application needs. We propose the following two steps solution:

- for the sentence to analyze, the finest corresponding most probable optimal maximum coverage (see sections 1 and 2) is generated first,
- then the possibly partial trees from this coverage are glued into one resulting tree (see section 3).

Figure 1 shows a simple example of a possible result from the robust parsing mechanism. The implementation of the robust parser is discussed in section 4.

1 Coverage

For a given sentence a *coverage*, with respect to an input grammar G, is a sequence of non-overlapping, possibly partial, derivation trees, such that the concatenation of the leaves of these trees corresponds to the whole input sentence.

Notice that the fact of restriction the coverages to derivation trees (i.e. trees verifying the left most nonterminal rewrtiting convention) excludes coverages such a “coverage” in figure 2.

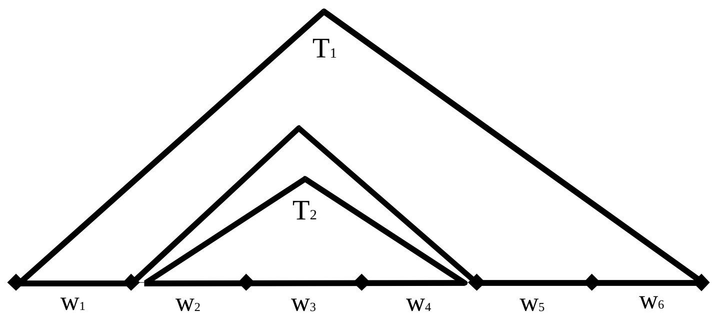


Figure 2: Partial trees, that can not be composed into a coverage.

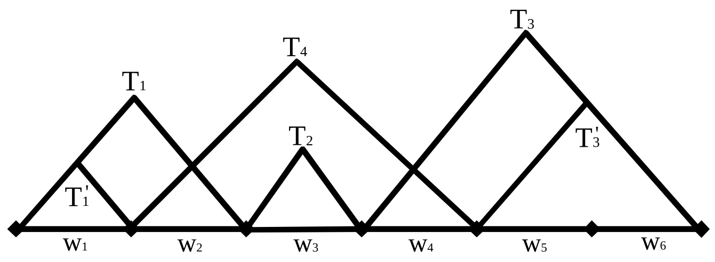


Figure 3: Partial derivation trees. Some of them (e.g. T_1, T_2, T_3 and $T_1^{'}, T_4, T_3^{'}$) can be composed into a coverage.

For an arbitrary derivation tree T, the foliage $f(T)$ is defined as the sequence of the leaves of T. So for a coverage $C = (T_1, T_2, ..., T_k)$ of the input sentence $w_1, w_2, ..., w_n$, we have:

$$f(T_1), f(T_2), ..., f(T_k) = w_1, w_2, ..., w_n.$$

In other words, if we define $f_i(T)$ as i-th leaf of T, $f_{last}(T)$ as the last leaf of T then for coverage $C = (T_1, T_2, ...T_k)$ of the input sentence $w_1, w_2, ..., w_n$, we have:

$$f_1(T_1) = w_1,\ f_{last}(T_k) = w_n \text{ and}$$
$$\text{if } f_{last}(T_i) = w_j \text{ for some } 1 \leq i < k \text{ and } 1 \leq j < n \text{ then } f_1(T_{i+1}) = w_{j+1}.$$

Figure 3 shows a coverage $C = (T_1, T_2, T_3)$ consisting of trees T_1, T_2 and T_3. If there are $T_1^{'}$ and $T_3^{'}$, $T_1^{'}$ is a subtree of tree T_1 and $T_3^{'}$ is a subtree of T_3, then we also have coverage $C^{'} = (T_1^{'}, T_4, T_3^{'})$. Conversely $(T_1, T_3^{'})$ and (T_1, T_4, T_3) are not coverages.

If there are no unknown words in the input sentence, then at least one trivial coverage is obtained, consisting of the trees that all use only lexical rules (i.e. one rule per tree).

1.1 Maximum coverage

A maximum coverage (m-coverage) is a coverage that is maximum with respect to the partial order relation $\leq$, defined as reflexive and transitive closure of the hereafter defined subsumed relation $\prec$.

The relation $\prec$ is a relation over coverages such that, for any coverages C and $C^{'}$:

$$C^{'} \prec C \text{ iff } \exists i, j, k,\ 1 \leq i \leq k, 1 \leq j \text{ and there exists rule } r \text{ in the grammar } G \text{ such that } C = (T_1, ..., T_i, ..., T_k),\ C^{'} = (T_1, ...T_{i-1}, T_1^{'}, T_2^{'}, ..., T_j^{'}, T_{i+1}, ..., T_k) \text{ and } T_i = r \circ T_1^{'} \circ T_2^{'} ... \circ T_j^{'},$$

i.e. if there exists a sub-sequence of trees in $C^{'}$ that can be connected by rule r and the resulting tree is element of C, the other trees in $C^{'}$ being the same as in C. Notice that the rule r can be a unary rule.

The relation $\leq$ is defined as the reflexive and transitive closure of the relation $\prec$. The relation $\leq$ is also antisymmetric. If $C^{'} \leq C$ and $C \leq C^{'}$ then:

- If $|C|$ denotes number of trees in the coverage C then $|C^{'}| \leq |C|$ and $|C| \leq |C^{'}|$, so $|C^{'}| = |C|$.
- If $C^{'} \prec C$ then $\exists T, T^{'}, T \in C, T^{'} \in C^{'}$ such that $T = r_1 \circ T^{'}$ for some unary rule r_1 from grammar G. If also $C \prec C^{'}$ then $T^{'} = r_2 \circ T$. But this is not possible, because $T = r_1 \circ T^{'}$. Notice that all the remaining corresponding trees in C and $C^{'}$ have to be the same. Thus $C^{'} \not\prec C$ and $C \not\prec C^{'}$. And also $C \equiv C^{'}$, because the relation $\leq$ is reflexive closure of the relation $\prec$.

As the relation $\leq$ is reflexive, transitive and antisymmetric, it also corresponds to a partial order on the set of all coverages of a given input sentence. A maximum coverage (m-coverage) is a coverage that is maximum with respect to the $\leq$ relation.

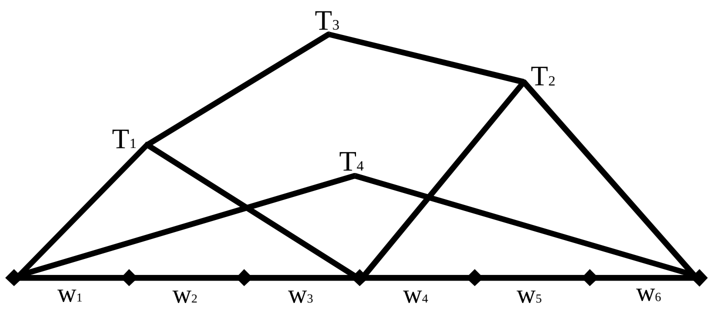


Figure 4: An example to illustrate a maximum coverage.

The coverage $C_1 = (T_3)$ in figure 4 is m-coverage. The coverage $C_2 = (T_1, T_2)$ is not maximum, because $C_2 \leq C_1$. There is also another m-coverage $C_3 = (T_4)$. Notice that C_1 and C_3 are not comparable by $\leq$ relation. If there is a successful parse (a single derivation tree that covers whole input sentence) then there are as many m-coverages as full parse trees and every m-coverage contains only one tree.

1.2 Optimal m-coverage

In addition to maximality, we focus on *optimal* m-coverage, where optimality is defined with respect to different measures. In contrast to maximality, which is generally defined for the coevrages, the choice of a optimality measure depends on the target application.

We propose the following two measures:

- the first optimality measure S_1 relates to the average width (number of leaves) of the derivation trees in the coverage. For an m-coverage $C = (T_1, T_2, ...T_k)$ of input sentence $w_1, w_2, ..., w_n$, $n > 1$, we define

  $$S_1(C) = \frac{1}{n-1}(\frac{n}{k} - 1).$$

  Notice that $0 \leq S_1(C) \leq 1$ and $\frac{n}{k}$ is the average width of the derivation trees in the coverage. With this measure, the value of a trivial coverage (i.e. exclusively made of lexical rules) is 0 and the value of a successful full parse is 1.

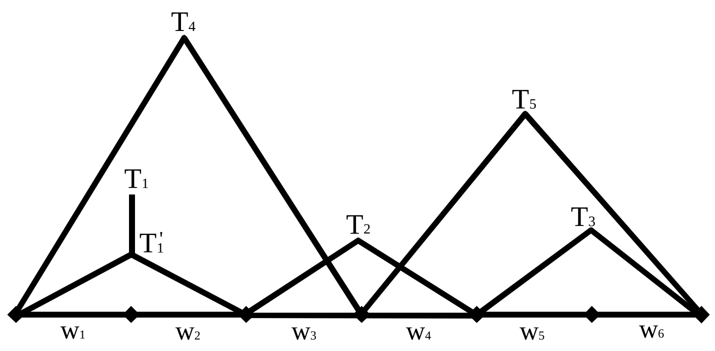


Figure 5: An example to illustrate the notion of optimal m-coverage.

- The second measure favours coverages with the widest trees (trees with the largest number of leaves). We define

  $$l_{max}(C) = \max_{T \in C} |f(T)|$$

  and

  $$S_2(C) = \tfrac{1}{n-1}(l_{max}(C) - 1)$$

  for number of input words $n > 1$. Similarly to S_1, $0 \leq S_2(C) \leq 1$, and the value obtained for a trivial coverage is 0 and the value of a successful full parse is 1.

Several other optimality measures could be defined. For instance, an optimality measure might be sensitive to the internal structure of the trees in a coverage, e.g. count the number of nodes in trees. These additional criteria can be used in a combination with measures S_1 and S_2.

Figure 5 illustrates m-coverages $C_1 = (T_1, T_2, T_3)$ and $C_2 = (T_4, T_5)$. The coverage $C_1' = (T_1', T_2, T_3)$ is not m-coverage. The coverage C_2 is more optimal for the measure S_1 ($S_1(C_1) < S_1(C_2)$), but it is less optimal for the measure S_2 ($S_2(C_2) < S_2(C_1)$). Notice that the coverages C_1 and C_2 are not comparable with the $\leq$ relation.

1.3 Probability of a coverage

The probability of a coverage is defined as the product of the probabilities of the trees it contains, i.e. for a coverage C we define

$$p(C) = \prod_{T \in C} p(T).$$

Notice that, by construction, the probability of any coverage is always less than or equal to the probability of the corresponding trivial coverage. The probability of a coverage can be viewed as another optimality measure. So the most probable coverages can be found in the same way as optimal m-coverages. But, usually we find all optimal m-coverages (OMC) first (optimal with respect to some other measure then probability) and then the most probable one is chosen. Both OMC and most probable OMC are not necessarily unique.

2 Finding optimal m-coverage

We use a parsing algorithm that produces all possible incomplete parses (i.e. whenever there exists a derivation tree that covers the part of the given input sentence, the algorithm produce that tree). This condition is usually satisfied by bottom-up parsers. Then, the incomplete parses can be combined to find the maximum coverage(s).

The described algorithm finds OMC with respect to the measure S_1 (the average width of the derivation trees in the coverage), but it can be easily adapted to different optimality measures.

All operations are applied to a set of Earley's items [Ear70]. In particular, no changes are made during the parsing phase (except some initialization of internal structures for better efficiency of the algorithm).

The Dijkstra's algorithm for shortest path problem in graphs is used to find OMCs with respect to the measure S_1. The input graph for the Dijkstra's algorithm consists of weighted edges and vertices. The edges are Earley's items and the weight of each edge is 1. The vertices are word positions, thus for n input words we have $n + 1$ vertices. Whenever the Dijkstra's algorithm finds paths with equal length (i.e. identical number of items), we use the probability to select the most probable ones. Notice that, if we assume that there are no unknown words, there exists at least one path from position 0 to n corresponding to the trivial coverage.

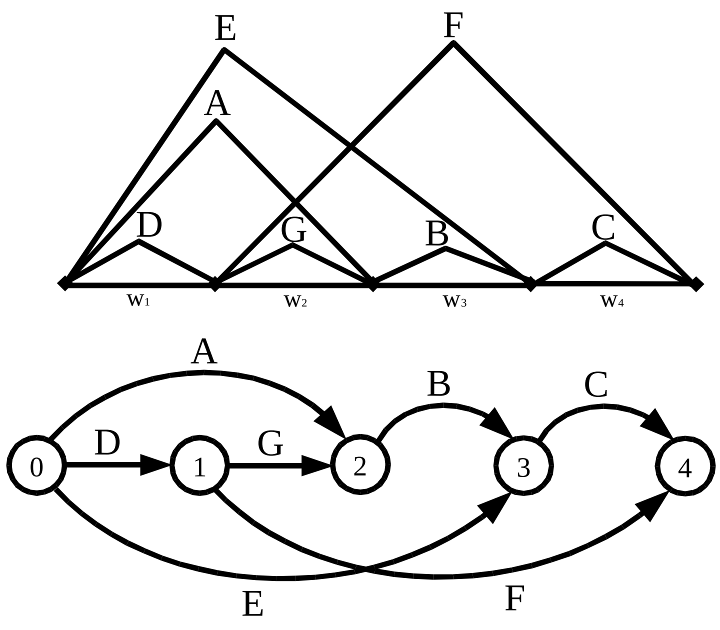


Figure 6: The input graph for the Dijkstra's algorithm and the corresponding derivation trees.

Figure 6 illustrates an example of the input graph for the Dijkstra's algorithm for Earley's items $[A, 0, 2]$, $[B, 2, 3]$, $[C, 3, 4]$, $[D, 0, 1]$, $[E, 0, 3]$, $[F, 1, 4]$ and $[G, 1, 2]$. The shortest paths are $[E, 0, 3]$, $[C, 3, 4]$ and $[D, 0, 1]$, $[F, 1, 4]$. The paths correspond to two optimal m-coverages with two trees in each coverage.

The output of the algorithm is a list of Earley's items. The Earley's item can represent several derivation trees and, to get an OMC, the most probable tree from each item is selected. The resulting OMC is not unique because there can be several trees with the same probability.

3 Gluing

The intended result for our robust parser is a derivation tree covering the whole input sentence. For this reason our goal is to connect (glue) the trees present in the OMC to construct a single one.

3.1 Gluing with new rules

The gluing can be realized by adding new rule(s) to the grammar. We impose the constraint that the new rules use new non-terminals and just connect the roots of the trees together. The probability of such rules is set to 1. Notice that there might be several other ways of constructing a unique tree and therefore our choice mainly rely on technical reasons.

Figure 7 shows example with new rules $S \rightarrow X^L$, $X^L \rightarrow X^L X$, $X^L \rightarrow X$, $X \rightarrow A_i$, where S is the root of the grammar, X and X^L are new non-terminals and A_i is the root of the i-th tree in the coverage (we have three trees in this example). The dotted lines represent newly added rules.

3.2 Gluing by means of mapping non-terminals

Another possibility is to create the top nodes of the resulting tree by the top-down parsing algorithm and then to glue these top nodes with the selected coverage. Notice that, for reasonable grammars, the tree with the following properties can be generated:

- the root is equal to the root of the grammar
- the number of leaves is equal to the number of trees in the coverage.

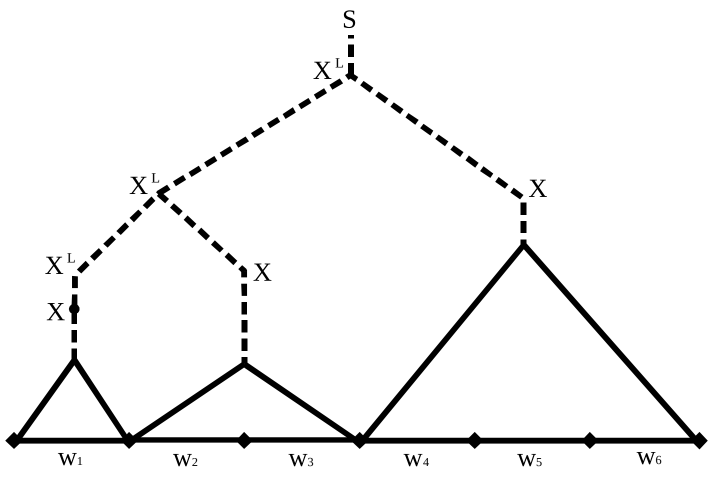


Figure 7: Gluing with new rules added to the grammar. Bottom bold trees are in OMC.

So, in this case, the gluing would be only a formula how to connect two non-terminals. This approach is illustrated in figure 8. The dotted lines represent the mapping function.

We did not implemented this method, because there are many remaining unsolved problems. The main challenge is to find out how to generate the top nodes with respect to the input sentence. A possible track to explore is to consider approaches deriven from head-corner parsing algorithm.

4 The implementation and tools

The SLP toolkit [CR98] is used to implement the above mentioned ideas. The SLP toolkit provides fast and robust bottom-up chart parsing algorithm derived from Earley's chart parsing [Ear70] and CYK [Kas65, You67, AU72, GHR80]. We plan to create an interface between the SLP toolkit and the `libkp` [KS03] to allow the sharing of the results from these tools.

The tools presented in the next sections are already implemented. We also implemented the algorithm which finds all coverages. The other methods suggested here are planned for the further development. Our goal is to integrate the robust algorithm in the SLP toolkit and the `libkp` library.

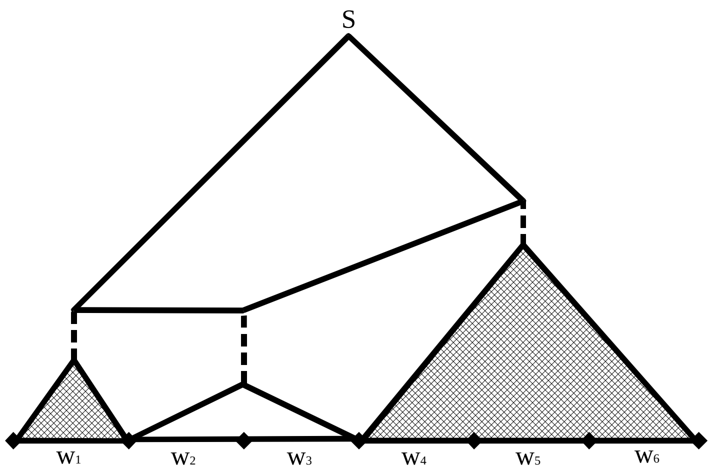


Figure 8: Gluing by means of mapping non-terminals. Bottom hatched trees are in OMC.

4.1 Connection between `libkp` and SLP toolkit

The main difference between `libkp` and SLP toolkit is that `libkp` uses a CFG augmented by semantic actions (contextual constraints). In the following we describe a method how, for a given sentence and CFG with actions, generate an equivalent CFG (without actions), i.e. that both grammars produces the same parse trees.

4.1.1 Evaluating contextual constraints in `libkp`

In `libkp` every grammar rule has zero, one or more embedded semantic actions. The actions are computed bottom-up[1] (like in `bison` [CSH02]). These actions serve the purpose of:

- computing a value used by another action at the higher level;
- throwing out incorrect derivation trees.

For example, the following grammar rule for genitive constructions in Czech has three semantic actions:

[1]Notice that we share derivation sub-trees and their values.

```
npnl -> np np +0.0784671532846715
  test_gen ( $$ $2 )
  prop_all ( $$ $1 )
  depends:1 ( $$ $1 $2 )
```

The first line contains a grammar rule with the probability obtained from a corpus. The contextual constraints are listed on the new lines. The number after the colon represents the internal type of the action. We can turn on or off the evaluation of actions with specified type. The `$$` parameter represents the return value. The `$n` parameter is a variable where we store a value of n-th nonterminal of the rule. Notice that the presented notation does not have to be used directly by users. It can be generated automatically from the meta-grammar format [SH00].

4.1.2 The representation of the values

It was shown that parsing is in general NP-complete if grammars are allowed to have agreement features [BBR87].

The pruning constraints in `libkp` are weaker than general feature structures. It allows us to have an efficient implementation with the following properties. A node in the derivation tree has only limited number of values (e.g. the cardinality of the set for noun groups in our system is max. 56 [SH00]).

We use a chart based parsing algorithm and the results of the parsing process is stored in a packed shared forest of Earley's items [Ear70]. To compute values we build a new forest of values instead of pruning the original packed shared forest. The worst-case time complexity for one node in the forest of values is therefore 56^δ, where δ is the length of the longest right-hand side grammar rule. Notice that this complexity is independent of the number of words in the input sentence.

Values in the forest of values are linked with Earley's items. Each item contains a single linked list of its values. Each value has a reference to its item. The value holds a single linked list of its children. The child is a one dimensional array of values. This array represents one combination of values that leads to the parent value. There can be more combinations of values that leads to the same value, e.g. $2-1$ and $3-2$ in a rule for a minus operator in a grammar for arithmetic expressions in figure 9.

The i-th cell of the array contains a reference to a value from i-th symbol on the RHS of the corresponding grammar rule. Notice that i-th symbol

```
e -> e "+" e
add ( $$ $1 $3 )

e -> e "-" e
sub ( $$ $1 $3 )

e -> e "/" e
is_not_zero ( $3 )
div ( $$ $1 $3 )

e -> NUMBER
value_of ( $$ $1 )
```

Figure 9: Grammar with contextual constraints.

has not to be used to compute the parent value, e.g. the symbol `"-"` in the example in figure 9. We only use reference to the item from such unused cell.

4.1.3 Generation of a grammar with values

We use the following procedure for every inactive item $[i, j, A \rightarrow X_1 X_2 ... X_n \bullet]$ in the chart:

- for every value v in the item, we generate the rule: $A \rightarrow A\_value$, where *value* is an unique textual representation of the value v.
- for every child of the value v, we generate the rule: $A\_value \rightarrow X_1' X_2' ... X_n'$, where X_i' is:
  - $X_i\_value_i$ if a value $value_i$ from i-th nonterminal is used to compute the value v.
  - X_i otherwise.

Duplicate rules are removed.

Figure 10 shows the generated grammar for the input `2 / 1 - 1` and the grammar with actions from figure 9. Notice, that the input has two derivations trees in the original grammar (if the actions are omitted), but the corresponding generated grammar gives us only one derivation tree, because of the `is_not_zero` action.

```
e -> e_0
e -> e_1
e -> e_2
e_0 -> e_1 "-" e_1
e_1 -> NUMBER
e_1 -> e_2 "-" e_1
e_2 -> NUMBER
e_2 -> e_2 "/" e_1
```

Figure 10: Generated grammar with values for the input `2 / 1 - 1`.

4.2 Tools

Because our experiments were based on comparing trees by hand, we developed the following utilities, that simplifies the work with trees from a corpora.

4.2.1 A tree with holes

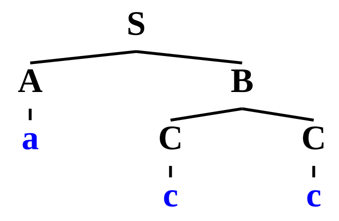
The `tree_with_holes` utility helps user to check whether a given tree can be generated by the grammar. If it can not be, then missing rules are marked, so they can be easily detected.

The `tree_with_holes` utility prints, for any given tree, a tree with marked nodes. The node in the input tree is marked if it is a part of non-grammatical rule. The root of such non-grammatical rule is marked with `(X)` and leafs are marked with `(X0)`.
Example:

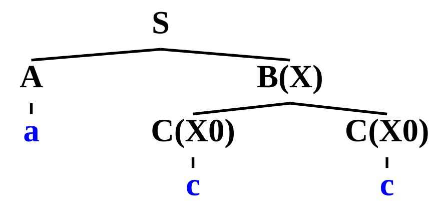
- Grammar:

  ```
  S -> A B
  A -> 'a'
  B -> C D
  C -> 'c'
  D -> 'd'
  ```

- Input tree:



- Output tree:



- The non-grammatical rule is $B \rightarrow CC$.

4.2.2 The number of ties with the most probable parse

The number of trees with the probability equal to the probability of the most probable parse is computed by the `anagram` utility (from SLP Toolkit) during the parsing process. The computation is almost the same as for a number of all trees. The only difference is when new sub-derivations are found. For the number of all trees, the number of the new derivations is just added to a current total number of trees. For the number of ties, we look at the maximum probability of the new derivations. If the maximum probability is the same as a current maximum probability, then the number of ties from the new derivations is added to the current number of ties. If the current maximum probability is lower then the maximum probability of the new derivations, then the current number of ties is replaced with the number of ties from the new derivations (and we also replace the current maximum probability). Otherwise we do not change anything.

Numbers with a floating point are used to represent the probabilities. Because we work with these floating point numbers during computation the number of ties, we should avoid errors that comes from the fact that floating point numbers represents only finite subset of the real numbers (e.g. the multiplication is not associative operation on the floating point numbers).

However, a logarithmic representation of the probabilities is used. Thus the problematic small numbers are big numbers in our representation. So usually no special arithmetic is needed to compute the probabilities.

4.2.3 Is a given tree the most probable tree?

This tool is a simple shell script. First of all we check, with the utility `tree_with_holes`, if the input tree can be generated by the grammar. If yes, the input sentence is extracted from the input tree. In the next step all trees with probability equal to the most probable parse are printed for the given input sentence. Notice that the number of these trees can be exponential with respect of the length of the input. Then we test if our tree appears in these trees.

5 Conclusion

In this report we presented our approaches to the robust stochastic parsing. We introduced the optimal maximum coverage framework and several measures for the optimality of the parser. Our definition of the maximality is independent of the target application. On the other hand, the choice of an optimality measure is strongly application dependent.

We proposed the algorithm that finds OMC (with respect to the measure average width of derivation trees) efficiently. The implementation of this algorithm in SLP toolkit was successfully used by Marita Ailomaa. The results of her experiments are published in [Ail04]. In the future, the interface between `libkp` and SLP toolkit will allow us to integrate contextual constraints into our robust parser.

References

[Ail04] Marita Ailomaa. Two approaches to robust stochastic parsing. Master's thesis, The Swiss Federal Institute of Technology, Lausanne, Switzerland, 2004.

[AU72] A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation and Compiling*, volume I: Parsing. Prentice-Hall, Englewood Cliffs, N.J., 1972.

[BBR87] G. E. Barton, R. C. Berwick, and E. S. Ristad. *Computational complexity and natural language.* MIT Press, Cambridge, Massachusetts, 1987.

[CR98] J.-C. Chappelier and M. Rajman. A generalized CYK algorithm for parsing stochastic CFG. In *TAPD'98 Workshop*, pages 133–137, Paris, France, 1998. (`http://slptk.sourceforge.net`).

[CSH02] Robert Corbett, Richard Stallman, and Wilfred Hansen. Bison parser generator, user manual, version 1.35, 2002. (`http://www.gnu.org/software/bison/bison.html`).

[Ear70] J. Earley. An efficient context-free parsing algorithm. In *Communications of the ACM*, volume 13, pages 94–102, 1970.

[GHR80] S.L. Graham, M.A. Harrison, and W.L. Ruzzo. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2(3):415–462, 1980.

[Kas65] T. Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. In *Technical report AF CRL-65-758*, Bedford, Massachusetts, 1965. Air Force Cambridge Research Laboratory.

[KS03] V. Kadlec and P. Smrž. PACE - parser comparison and evaluation. In *Proceedings of the 8th International Workshop on Parsing Technologies, IWPT 2003*, pages 211–212, Le Chesnay Cedex, France, 2003. INRIA, Domaine de Voluceau, Rocquencourt.

[SH00] P. Smrž and A. Horák. Large scale parsing of Czech. In *Proceedings of Efficiency in Large-Scale Parsing Systems Workshop, COLING'2000*, pages 43–50, Saarbrucken: Universitaet des Saarlandes, 2000.

[You67] D.H. Younger. Recognition of context-free languages in time n^3. *Inf. Control*, 10(2):189–208, 1967.