Dependency Analyser Configurable by Measures

Tomáš Holan

Department of Software and Computer Science Education, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic,
holan@ksvi.ms.mff.cuni.cz

Abstract. In this paper we present a dependency analyser able to compute syntax recognition and analysis according to dependency grammars. Analyser is able to deal with nonprojective constructions, it has means to express the level of word-order freedom and its limitations. Level of word-order freedom and level of robustness (correctness) of sentences can be given as parameters of the analysis. Data and grammar definition laguages are also presented.

1 Introduction

A syntactic analysis is a part of many tasks of computational linguistics like e.g. machine translation, natural-language communication, grammar-checking or full-text search.

One of the big problems of syntax analysis of natural languages is a complexity of the computing and high number of results caused by ambiguity of natural language words and/or ambiguity in syntaxe. This complexity grows even more if we want to use some form of relaxation of language and to allow to analyse sentences containing some errors.

Methods used to solve problem of complexity are disambiguation, pruning or heuristics that less or more help to find the "right" tree/derivation for the sentence.

In our case where analysis is used as a part of grammar-checker or as a testing/debugging tool supporting work on the grammar of natural language, these methods are not acceptable because we need to get *all* trees/derivations of the sentence that can be found.

So we use other means to decrease time and space complexity of analysis.

Those means are measures of word-order complexity ((non)projectivity), measures of robustness (number of allowed errors) and the analyser computing syntax analysis limited by the given values of those measures.

2 Definitions

We will introduce concepts of D- and DR-trees, measures of (non)projectivity, D-grammars and grammars with errors. We will proceed in a rather informal way, exact definitions can be found e.g. in [12].

2.1 Trees, measures

Dependency structure of the sentence is expresed by *dependency tree (D-tree)*, where edges oriented from leaves to the root show a dependencies between the words. Nodes contain *symbol*, *horizontal index* (number of the word in the sentence), *vertical index* (level, distance from the node to the root) and *dominancy index* (horizontal index of the node this node depends on).

To demonstrate order of reduction or order of creation of dependencies we use a *DR-tree* (delete-rewrite-tree), where any vertical edge mark a rewrition of dominant symbol and any obligue edge corresponds to deletion of dependent symbol during a reduction.

D- and DR-tres are related to each other by the concept of *contraction*. We say that the D-tree T is the *contraction of the DR-tree* TT if obligue edges join the same pairs of horizontal positions inside the sentence (corresponding nodes have the same value of dominancy index).

For both D- and DR-trees we can define a $Cov(u,T)$ — *coverage of a node* u *in the tree* T as a set of all *horizontal indices* of all the nodes from which a path (bottom up, including empty path) leads to u.

If $Cov(u,T) = \{i_1, i_2, \ldots, i_m\}, i_1 < i_2 < \ldots < i_{m-1} < i_m$, we say that the pair (i_j, i_{j+1}) forms a *gap in* $Cov(u,T)$, iff $i_{j+1} - i_j > 1$.

Measure $dNg(u,T)$ we define as a number of gaps in the $Cov(u,T)$, $dNg(T)$ is the maximum of $dNg(u,T)$ for all nodes u of the tree T. Value $dNg(T)$ we call the *measure of nonprojectivity of the dependecy tree* T; we say that dependency tree T is *projective* if $dNg(T) = 0$, in other case we say that tree T is *nonprojective*.

This definition of (non)projectivity gives us the same border between the sets of projective and non-projective trees as other definitions based e.g. on crossing of edges (see [1], [2]) but it is easy to evaluate during the process of bottom-up analysis.

By the same way we define $Ng(u,T)$ as a number of gaps in coverage of the node of DR-tree and $Ng(T)$ as the *measure of nonprojectivity of DR-tree* T. We say that DR-tree T is *projective* iff $Ng(T) = 0$.

2.2 Dependency grammars

In [9] we have introduced a class of formal grammars, *Robust Free-Order Dependency Grammars (RFODG's)* as a formal foundation to the way we were developing a grammar-checker for Czech, a natural language with a considerable level of word-order freedom. In this text we will call that grammars simply *dependency grammars (DG)*.

Dependency grammar (D-grammar) is a quadruple $G = (T, N, S_t, P)$ where T is the set of terminals, N is the set of nonterminals, $S_t \subseteq (T \cup N)$ is the set of starting symbols and P is the set of rewriting rules of two forms: $A \rightarrow_X BC$ and $A \rightarrow B$, where $A, B, C \in (T \cup N)$ and $X \in \{L, R\}$. The letters L (R) in the subscripts of the rules mean that the first (second) symbol on the right-hand

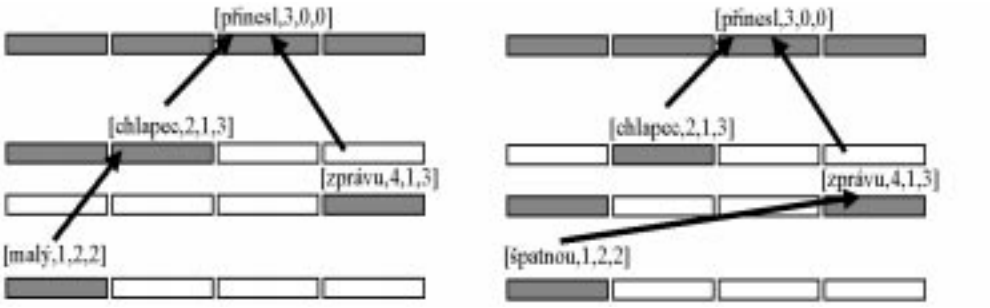

Figure 1. Examples of projective (dNg=0) and non-projective (dNg=1) D-trees with marked coverages of their nodes

side of the rule is considered *dominant*, and the other *dependent*. If a rule has only one symbol on its right-hand side, we consider the symbol to be *dominant*.

The rule has the following meaning (for reduction): The dependent symbol is deleted (if there is one on the right-hand side of the rule), and the dominant one is rewritten (replaced) by the symbol standing on the left-hand side of the rule. The rules $A \rightarrow_L BC$, $A \rightarrow_R BC$ can be used for a reduction of a sentence (sentential form) w for any of the occurrences of the symbols B, C in w, where B precedes (not necessarily immediately) C in w. The rule $A \rightarrow B$ can during reduction rewrite any occurrence of B in w.

We say that a DR-tree T is a DR-tree according to dependency grammar G if the symbol of the root node belongs to the set of starting symbols of grammar G and for every non-leave node it is possible to find rule of the grammar G describing relation between the node and its daughters (dominant–vertical edge, dependent–obligue edge, order of daughters (left-right)). In the case of DR-tree according to any dependency grammar we can talk about *rule assigned to the edge*.

We can refine the expressing power of D-gramars by introducing the *local limitations of non-projectivity*:

The *set of limitations of non-projectivity of symbols* is a set $Cs = \{[A, i] | A \in (N \cup T), i \geq 0\}$. The pair $G_{Cs} = (G,\text{Cs})$ is called *1-DG grammar*. We say that DR-tree T is a *DR-tree according to 1-DG* (G, Cs) iff it is a DR-tree according to D-grammar G and for every node u of T it holds that if A is the symbol of node u and if $[A, x] \in Cs$, then $Ng(u, T) \leq x$.

1-DG grammars can describe all context-free languages without empty words and also some non-context free languages, e.g. language $\{a^n b^n c^n | n > 0\}$.

2.3 Grammars with Erorrs

For the purpose of searching of syntactic errors it could be possibble to use single grammar to recognize correct sentences and other sentences mark as incorrect.

We hope that we have to make difference between error in the sentence and error in (or insufficience of) the grammar. Therefore we use two grammars, *positive grammar* and *extended grammar*, to clasify sentences into three classes: *correct sentences* i.e. recognised by positive grammar, *incorrect sentences* i.e. recognised by extended grammar only (sentence contains some known errors like e.g. broken disagreement condition) and *unclassified sentences* i.e. not recognised by any grammar. The pair (*positive grammar*, *extended grammar*) is called *grammar with errors*.

Rules of a positive grammar (*positive rules*) form subset of rules of extended grammar, rules present in extended grammar only we call *negative rules*.

We define $Ne(T)$ *measure of robustness (incorrectness)* of a DR-tree according to D-gramar as a number of applications of negative rules inside the tree T. By the same way we can define $dNe(T)$, the measure of robustness of D-tree T.

Formalism used by our analyser for grammar definition allows to describe positive and extended grammars both in one by placing tags "error rule" into the rule, it is used mostly in the case of (not) fulfilling of some constraints.

3 Analyser

There are many known dependency analysers and formalisms of expressing dependency grammars, see e.g. [4], [5] or [7]. They are suitable for various languages (mostly english) and for the various purposes.

Our approach differs in the way of expressing rules, in dealing with errorneous and nonprojective sentences and mainly in posibilities to expres limitations of analysis by measures.

Whole tool consists from three parts: *data (sentence) definition language*, *grammar definition language* and *configurable analyser*.

3.1 Language for definition of sentences

It is a language for definition of input data — morphologically and lexically processed sentences of natural language. Words can have only simple attributes except one complex attribute for a valency frame. The language also allows an efficient way of notation of morphological and lexical ambiguity.

Next example shows the (simplified) definition of the Czech word "kvtiny" (flowers) in the language for the definition of sentences.

```
kvetiny
LEXF: kvetina
WCL: noun
SYNTCL: noun
GENDER: fem
?
   NUM: pl
```

```
   CASE: ? voc , acc , nom !
,
   NUM: sg
   CASE: gen
!
END
```

This example illustrates usage of the branching on the level of whole sets of attributes (branchning after attribute `GENDER`) and also branching on the level of values (values of the attribute `CASE`). Altogether this definition corresponds to four unambiguous definitions. Question mark allways denotes beginning of branching, commas separate variants and exclamation mark denotes the end of branching. Branchings can be nested, the resulting set of descriptions is the set of all possible combinations (cartesian product).

3.2 Language for definition of grammars

This language serves for definition of grammars with errors. Meta-rules of a grammar are described as a sequences of commands checking the conditions of applicability of the meta-rule and forming the instance of the meta-rule for given right-hand side symbols. Variables used inside the metarule are: `X` for the left-hand symbol of the rule, `A,B` for the first and second right-hand symbol of the rule and `P`, temporary variable for an element of the frameset.

Head of a meta-rule can contain attributes declaring limitation of the meta-rule to projective use only (PROJECTIVE), limitation of non-projectivity of the meta-rule to closest dominant symbol (CLOSEST) or negative-only meta-rules (NEGATIVE).

Commands forming meta-rules are

- *hard-condition* declaring the value of the attribute needed for application of the meta-rule, e.g.
  `A.SYNTCL=noun`
- *soft-condition* declaring the value of the attribute needed for application of the positive meta-rule and name of the error in the case of breaking that condition, e.g.
  `A.CASE ? B.CASE CaseDisagreement`
- *initialization* declaring which of symbols A, B will be dominant and initializing rewritted dominant symbol (left-hand side of the rule), e.g.
  `X := B`
- *assignment* setting the value of the attribute of (rewrited) dominant symbol (left hand side of the rule), e.g.
  `X.HasRightGenitive := yes`
- *selection from the frameset* selects one element of the specified frameset into variable `P`, rest of the rule continues separately for all elements of the frameset, e.g.
  `P in A.FRAMESET`

- *deletion from the frameset* deletes selected element from the specified frameset, e.g.
  `DELETE P from A.FRAMESET`
- *other statements* — `OK` (successfull end), `FAIL` (unsuccesfull end), `ERROR name` (place error sign).
- *conditional statement* if-then-else-endif allowing branching of the meta-rule, e.g.

```
IF A.SYNTCL=noun THEN X.a := n
                 ELSE
   IF A.SYNTCL=pronoun THEN X.a := p
                       ELSE FAIL
   ENDIF
ENDIF
```

Following example presents the simplified definition of the meta-rule for adjoining the adjective as the left congruent attribute to the noun:

```
METARULE LeftCongruentAttribute
   A.SYNTCL=ADJ
   B.SYNTCL=NOUN
   A.GENDER ? B.GENDER GenderDisagreement
   A.CASE   ? B.CASE   CaseDisagreement
   A.NUMBER ? B.NUMBER NumberDisagreement
   X:=B
   OK
END_METARULE
```

If interpretation of the metarule succeeds and all soft-conditions was fulfilled then we get rule of the positive grammar. If any of soft conditions is not true then interpretation of metarule continues but resulting rule will be negative rule of extended grammar with appropriate error code.

The set of attributes and set of values are not fixed, author of grammar can use her/his own attributes and values. Dictionaries of attributes and values are created during loading of grammar and sentences.

3.3 Configurable analyser

Analyser provides computation of all DR- and D-trees for a given sentence according to a given grammar fulfilling the limitations of the measures Ng, dNg and Ne. Results are presented on different levels, from the list of items of parsing to the sets of DR-trees or D-trees.

Output is a list of DR-trees and their shapes and list of D-trees and their shapes. Those list are interpretted by the TreeView utility.

Analyser is based on bottom-up parsing, working with items presenting (sub)trees. Item contains symbol and horizontal position of a root of the subtree, its coverage and references to daughter items. Analyser tries to combine

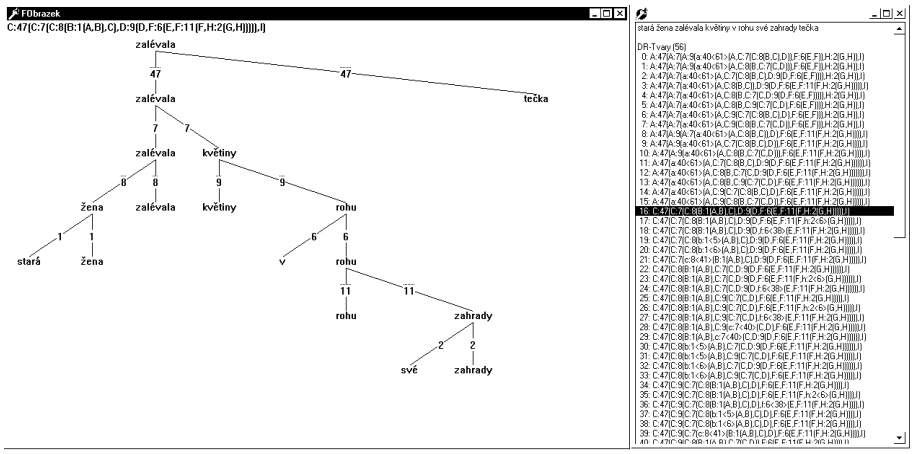

Figure 2. Example of output viewed by TreeView. It shows the tree selected from the list of all trees (window on the right side).

two items i.e. it checks disjunctivity of their coverages, compute their union as a coverage of new item, check if it fullfill given restrictions of nonprojectivity (measure Ng) and of robustness (measure Ne), then it tries to interpret metarules on root symbols of items (by horizontal indices left (A) and right (B)), if metarule succeed (OK) then it creates a new item.

For any new item created analyser checks whether there already exists an item with the same symbol, horizontal position of a root and coverage. If it exists then new item is marked as *duplicite item* and it will not be used for creating of new items. Duplicite items are used later to find all trees for the sentence.

4 Conclusion

The main aim of this contribution was to present dependency analyser configurable by measures.

Described analyser can analyse non-projective sentences, it can deal with errors (extended grammar), it can use global limits of non-projectivity and correctness, given as a parameters of analysis. It can also use local limits of non-projectivity of single metarules, it compute full set of trees, without pruning, as it is needed during testing a natural language grammar.

The analyser was used for pilot implementation of a grammar-checker [8], as a base of the robust analyser of Czech [11] and of a machine aided translation tool [10] and also as a testing tool for the research on prepositional phrases in Czech [13].

References

1. Jürgen Kunze: Die Auslassbarkeit von Saltzteilen bei koordinativen Verbindungen in Deutschen. Akademie-Verlag-Berlin, 1972
2. Ladislav Nebeský: A Projectivity Theorem. In: Prague Studies in Mathematical Linguistics (3), Academia, Praha, 1972, pp. 165-169
3. Alexej V. Gladkij: Formalnyje gramatiki i jaziki. Nauka, Moskva, 1973
4. Martin Kay: Functional Unification Grammar: A formalism for machine translation.. In: Proceedings of the 10th International Conference on Computational Linguistics, Stanford University, California, 1984, pp.75-78
5. Nicholas J. Haddock, Ewan Klein, Glyn Morrill: Categorial Grammar, Unification Grammar and Parsing. Edinburgh University Centre for Cognitive Science, Edinburgh 1987
6. Igor A. Meľčuk: Dependency Syntax: Theory and Practice. State University of New York Press, 1988
7. Carl J. Pollard, Ivan A. Sag: Head-driven Phrase Structure Grammar. University of Chicago Press, Chicago 1994
8. Tomáš Holan, Vladislav Kuboň, Martin Plátek: An Implementation of Syntactic Analysis of Czech. ÚFAL MFF UK, Praha, 1996, Language Technologies for Slavic Languages
9. Martin Plátek, Vladislav Kuboň, Tomáš Holan: Formal Tools for Separating Syntactically Correct and Incorrect Structures (extended abstract to a poster). In: Proceedings of the Conference IWPT'97, pp. 247-248, Massachusetts Institute of Technology, Boston, MA, USA, 1997
10. Libor Lisý: Pracovní prostředí překladatele na počítači PC. Diploma thesis, MFF UK, Praha, 1998
11. Vladislav Kuboň: A Robust Parser for Czech. Tech. report TR 1999-6, ÚFAL MFF UK 1999
12. Tomáš Holan, Vladislav Kuboň, Karel Oliva, Martin Plátek: On Complexity of Word Order. In: Les grammaires de dpendance - Traitement automatique des langues Vol 41, No 1 (2000) (special issue on dependency grammars of the journal Traitement automatique des langues, guest editor Sylvain Kahane), pp. 273-300.
13. Markéta Straňáková: Selected Types of Pg-Ambiguity: Processing Based on Analysis by Reduction. In: Proceedings of TSD'00, P.Sojka, I.Kopeček, K.Pala (Eds.), Springer Verlag, LNAI 1902, pp.139-144, 2000