APPLYING AND IMPROVING THE RESTRICTION GRAMMAR APPROACH FOR DUTCH PATIENT DISCHARGE SUMMARIES

PETER SPYNS [1,2] GEERT ADRIAENS [1,3]
Katholieke Universiteit Leuven
[1] Center for Computational Linguistics (ccl@ct.kuleuven.ac.be)
Maria-Theresiastraat 21, B-3000 Leuven, Belgium
[2] Division of Medical Informatics, University Hospital Gasthuisberg
Herestraat 49, B-3000 Leuven, Belgium
[3] Siemens-Nixdorf Software Center Liège
Rue Des Fories 2, 4020 Liège, Belgium

0. abstract

This paper starts by giving a short overview of one existing NLP project for the medical sublanguage (1). After having presented our objectives (2), we will describe the Restriction Grammar formalism (3), the datastructure we use for parsing (4) and our parser (5) enhanced with a special control structure (6). An attempt to build a bootstrap dictionary of medical terminology in a semi-automatic way will also be discussed (7). A brief evaluation (8) and a short outline of our future research (9) will conclude this article.

1. context and state of the art

In medecine, the use of natural language for compiling reports and patient documents is a widespread habit. The importance of the information embedded in a patient discharge summary (PDS) has led to the creation of various storage and retrieval programs (e.g. Dorda 1990) . Basically, these programs use pattern-matching (enhanced with boolean selection possibilities) to retrieve the required information. A more scientifically based method to extract the information from a patient discharge summary is a linguistically based analysis, which captures all the subtilities of the free text. This "intelligent" approach permits questions that imply deductive reasoning and is therefore preferred to simple pattern-matching based techniques (Zweigenbaum et al 1990).

A team from the New York University, directed by Sager, has developed an NLP system that analyzes PDSs, structures their information and stores the whole in a relational database. The data can thus be accessed by other programs in an organized way. This Medical Language Processor (MLP) is an extension of the Linguistic String Project (LSP) (Sager 1981), which aimed at analyzing technical and specialized language by means of *String Grammar* (Sager et al 1987). More recently, the LSP-MLP was successfully imported in Europe and it is currently functioning in the Hôpital Cantonal de Genève, where it handles French patient documents (Sager et al 1989).

2. objectives

Starting from an existing grammar for English, an attempt was made to develop a grammar for the Dutch language by making use of the experience gained during the LSP. This implies the creation of a set of grammar rules, the implementation of a interpreter/translator for these rules and the generation of a limited dictionary.

On the basis of a set of 6 PDSs, a limited grammar has been built. Every word constitutes a separate entry in the dictionary; no morphologic analysis takes places. Conjunctions are not yet handled, but the possibility exists (Hirschman 1986). Relative clauses also fall outside the scope of the current grammar. It is our intention to use the grammar to analyze free input (as it occurs in PDSs) with an eye to extracting the relevant medical information. From there on, the system will be used to help medical secretaries in the classification of the PDSs with respect to medical database systems.

3. restriction grammar

3.1. historical background

Restriction Grammar (RG) is the Prolog-version of String Grammar, which emanated during the 60s as the grammar formalism of the distributionalism school promoted by Harris (1962). One might wonder why this theory is being reused. An up-to-date theoretical formalism suffers to a large extent from the limited and experimental character of its applications. The LSP-MLP has proved to lead to large scale useful NLP systems. That is why we adopted the same theoretical background (distributionalism) to develop an analogous grammar for the Dutch language.

3.2. relation with other logic formalisms

The RG-formalism is related to Definite Clause Grammar (DCG). A grammar consists in both formalisms of a set of context-free production rules interspersed with Prolog predicates that function as restrictions. Advantages of RG are the absence of parameters in the rules and the separate treatment of context-

sensitive information, which is stored in a tree-structure that is gradually built up during the sentence-analysis. Flexibility in creating, adapting and checking the grammar is thus guaranteed (Hirschman 1986, Hirschman & Puder 1986).

3.3. some type definitions

The RG-grammar contains different types of structures.

A linguistic string is a single option definition which consists of a sequence of "required" elements in direct correspondence with an elementary word in the sentence, interspersed with elements of the adjunct set type. For instance, the definition for an affirmation is the following:

affirmation ::= np, sa_rec_opt, vp, sa_rec_opt.

This rule states that the affirmation consists of an *np* and *vp*-string (both required) as well as optional sentence adjunct strings.

An adjunct set definition has several options, which are the names of string definitions or sets of string definitions. The strings of the adjunct set are optional additions to the sentence. In opposition to Sager and Hirschman, the optionality and recursivity of some strings (*sa*, *rn*, *rv*) is embedded in our grammar and no special parser mechanisms are needed (see below).

sa ::= pdate; ptime; pn; nstgt;csstg; dstg.

The rule states that a sentence adjunct can consist of prepositional strings (indicating a *date* or moment in *time*), an expression of time (*nstgt*), a subordinate sentence-structure (*csstg*), or an adverbial string (*dstg*).

Structures of the type LXR consist of a word class "X", which is the core or head of the structure, occurring with optional left and right adjuncts (Sager 1981).

lnr ::= ln, nvar, rn_rec_opt .

Here, a nominal constituent is surrounded by its left and right adjunct strings.

Positional variants are auxiliary definitions which group together linguistically related classes of strings.

nvar ::= lnamer; {d_nvar}, *n; {d_inf},lo_vinf_ro;
{dn1}, *nulln; *pro, {w_pro2}.

The core of a nominal constituent can be formed by a proper name with its adjuncts, a noun, a nominalized verb string, a null noun and a pronoun. Note the existence of the zero noun (*nulln), which is only justified for reasons of the economy principle applied to the grammar rather than on purely linguistic grounds.

New optional and recursive structures are created and included in our grammar. This was useful to skip the extra machinery needed by Sager and Hirschman to cope with the "empty" and repeated adjunct strings (Sager 1972). These new strings allow the parser to adopt a uniform strategy for the complete grammar. All the strings of the adjunct set have a corresponding optional and recursive structure.

sa_rec_opt ::= sa_rec; []. (optional definition)
sa_rec ::= sa, sa_rec_opt. (recursive definition)

These constructs allow a transparent treatment of an optional recursive sentence-adjunct.

Sometimes, the theoretical frontiers between the different structures are slightly blurred in grammar rules, but this will be cleaned up during the further development of the grammar.

As opposed to non-terminal categories (the structures already mentioned), a terminal category constitutes a leaf of the parse tree. The variable *Word* (see below) of every leaf is instantiated. In the grammar, a terminal category is marked by an asterisk.

nvar ::= lnamer; {d_nvar}, *n; {d_inf},lo_vinf_ro;
{dn1}, *nulln; *pro, {w_pro2}.

Literals are words which are directly integrated in the grammar and function more or less as fixed expressions. The example shows the definition of a left-adjunct to a proper noun. The literals appear between square brackets.

lname ::= [Prof, '.', dr, '.']; [dr,'.'] .

3.4. restrictions

The application of the grammar rules to a sentence can result in various parse trees. These are not necessarily correct from a pragmatic-linguistic point of view. Thus, the need emerges to distinguish the acceptable analyses from the bad ones. The restrictions prune the combinatorial explosion of parses permitted by the context-free grammar rules. When a restriction fails, the next positional variant of the category which functions as the head of a grammar rule is considered. Restrictions appear between curly brackets.

nvar ::= lnamer; {d_nvar}, *n; {d_inf},lo_vinf_ro;
{dn1}, *nulln; *pro, {w_pro2}.

There exist two kinds of restrictions. The *well-formedness restrictions* check if the parse tree meets syntactic and semantic constraints imposed on the leaves or on the syntactic relations between various nodes of the parse tree (e.g. w_pro2). *Disqualify restrictions* consider the input stream (e.g. d_inf fails if no infinitive is present in the remaining input stream)

and enhance efficient parsing by blocking directions which lead to failure. The restrictions are implemented by means of special functions (locating routines) that allow navigating from one node in the tree to another.

Currently, the still limited grammar for the Dutch language consists of some 157 RG-clauses completed with 41 restrictions. As already mentioned, conjunctions are not yet covered; neither are relative clauses or interrogatives sentences. This grammar was the result of confronting a larger theoretically built grammar with the linguistic reality of 6 patient discharge summaries.

4. the data structure of the parse tree

The data structure used for the parse tree is the one proposed by Hirschman and Puder (1986). The parse tree together with its operators are defined as an abstract data type. The abstraction function can be defined as follows:

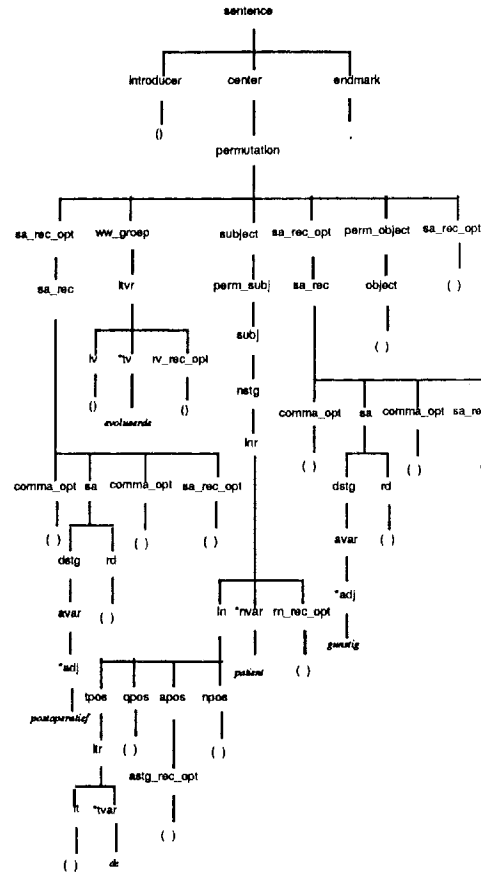
F: Node -> [TreeTerm,Path] :
TreeTerm = tt(Label,Child,Sibling,Word),
Word = [Item,Info] .

The representation invariant is:
V k E {Nodes} :
Item is a Dutch word or punctuation sign,
Info is one of the grammatical categories,
Item = [] <=> Label = *nulln,
Word is instantiated <=>
Label = (literal v terminal),
Label is always ground .

The first relation of the abstraction function states that a TreeTerm contains linguistic (Label,Item) as well as positional information (Child,Sibling). The second relation links a node to the path in the parse tree that must be traversed to reach the root starting from that node. Movements in the parse tree are executed by means of the **up/2, down/2, right/2** and **left/2** operators. As subtrees are successfully generated, the path gradually shrinks during the movement upwards in the tree.

5. the implementation of the parser

Basically, an RG translator-interpreter does not differ substantially from a DCG translator-interpreter. The main difference resides in the handling of the parameters needed for the construction of the parse tree (absent in RG-rules). The translator-interpreter takes care of the parameter bindings instead of the grammar rules as is the case in DCG. An example of the output after parsing a sentence can be found below.



6. the control structure

6.1. general outline

The interpreter/translator mechanism as it is described in Hirschman 1986 or Dowding & Hirschman 1988 works in a depth-first fashion that backtracks when necessary. We added a control structure that records the options that are unsuccessfully tried as well as the options that have led to a null-node.

The sentence to be analyzed is logically considered as a graph where the words of the sentence function as arcs between the vertices. The arcs will be labeled with grammatical categories. An arc can span various vertices. The "sentence-arc" e.g. spans all the vertices. An arc is uniquely identified by its label and coordinates in the tree (number of the starting vertex and tree-depth). The fourth characteristic of a arc, its mode, serves to distinguish a *void* from an *empty* arc. A void arc means that a grammatical category under a certain

vertex leads to no success, whereas an empty arc only leads to success if it is realized as an empty string. In opposition to a chart parser that remembers which grammatical structure spans which vertices, the only function of the control structure is to record which grammatical category leads to failure or realization as an empty string under a certain vertex (= a word).

6.2. the data structure of the control mechanism

The backbone consists of a kind of sparse matrix, of which each element is a stack of grammatical labels. The stack contains all the grammatical labels of the arcs that were (or still are) considered on the current depth in the parse tree under the current vertex. The Last In First Out ordering principle provides the advantage that arcs recently added below a vertex under construction can be modified or discarded without the need for time-consuming search-routines.

Here again the control structure is conceived as an abstract data type. There exist only five public predicates: **check_option/3**, **treat_most_recent-arc/4**, **empty_arc/3**, **initialize_control/1** and **remove_ control/0** .

6.3. the parsing algorithm

When the parser tests if an arc can span two vertices (**check-option/3**), the control structure is checked first to see if the same grammatical category is not already present under the starting vertex. If this is the case, a failure blocks this option and the parser will consider the next category in the grammar rule as the new candidate arc. In the other case, the arc is included as a void arc in the control structure. This prevents the parser from ending up in an infinite loop, by trying to satisfy a grammatical category already functioning as a candidate arc but on a higher level of the parse tree.

If a category is present as an empty arc (**empty_arc/3**) in the control structure, this means that all the other underlying grammatical categories have already been tried, but only the empty string can satisfy this category. When the parser backtracks and retries this category under the same vertex, useless search paths can be pruned as the previous parsing attempt only allowed the empty string.

On successful completion of a parsing attempt by a terminal element or a literal, the grammatical category is either removed from the control structure to allow the same category to be retried on backtracking, or marked as an empty arc (**treat_most_recent_arc/4**). Backtracking does not affect the control structure, as it is implemented by means of global Prolog "record keys".

7. semi-automatic dictionary buildup

As was noted by Wolff, the major lexical category of a word is determined by its final constituent part (Wolff 1984). In order to semi-automatically build up our dictionary (currently only containing word class information), we ordered a set of technical medical terms alphabetically starting from their ends. This allowed us to distinguish relevant suffixes and determine the associated grammatical category (e.g. the suffix *-iaca* is an adjectival suffix. The data about the link between suffix and word class can be entered into the system by means of a short interactive program. Some words belonging to a closed grammatical category (e.g. articles) are also integrated in the suffix file. A distinction is made between the real suffixes and the closed category words. The latter are stored under the form of a closed Prolog-list while the former are entered as open ended Prolog-lists.

The suffix entries are alphabetically classified, whereby the ending character functions as the sort key. The "ending groups" created by this process contain two subgroups: the real morphemes versus the real suffixes. The former have a higher ordering value than the latter. The latter are in addition ordered according their decreasing length. Concerning the suffixes, this means that the principle of the longest match is applied. The general strategy can be summarized as follows:

> (...) the table is looked at starting with the most specific entry, and ending with the least specific one (Adriaens & Lemmens 1990).

The output of the classifying program is a (primitive) dictionary file that must be completed, e.g. by adding semantic features. Some words of the dictionary file can be marked as "category unknown" or can be attributed the wrong category. To cope with this problem, an interactive program was added to allow corrections to be carried out. Despite the fact that the semi-automatic classification of words proved to be highly effective and time-saving (partly due to the high degree of regularity in medical terminology), it still functions on a too limited basis to be fully integrated in an NLP system.

8. evaluation and results

The most serious problem for our parsing approach appears to be structural ambiguity. A branch is attached to the first possible node in the parse tree instead of a subsequent node on the same or higher level. This is due to the depth-first mechanism of the parser. The generation of more than one parse tree does not provide a solution, because then the question on which basis a final parse tree will be selected remains unanswered. The LSP-MLP did also encounter this problem, but allows the adjuncts to be integrated freely (Sager 1981). Subsequently, the parse tree is passed to a "sublanguage selection module" and re- arranged on the basis of lists of allowed syntagmatic combinations

of semantic classes of the medical language as they appear in the distribution pattern of a word.

9. further research

Further research will be oriented towards the integration of conjunction handling, which implies the use of a meta-grammar (Hirschman 1986). The existing grammar will be continously extended in two phases. After a survey of various grammar books, theoretically sound grammar rules will be developed. These rules will subsequently be checked against a corpus of PDSs, to see how well the paper grammars fare when confronted with random input.
A more elaborated grammar requires more and refined restrictions, which need to eliminate more accurately dead ends during the parsing process. A sublanguage module must be added to account for the syntactic specificities of the medical language as well as to rearrange some branches of the parse tree. To transcend the prototype status, a large scale dictionary should be developed, including a morphological analyzer.

Furthermore, as the ultimate goal is to represent the meaning of the sentence, the syntactic analysis needs to be completed by a semantic counterpart. A distinction should be made between general purpose and medical sublanguage concepts. The already mentioned rearrangement of the parse tree by the sublanguage component uses semantic classes, but these are created on the basis of a syntactic analysis of the distribution of the various lexical terms. This does not include any modeling of the medical domain nor any deductive reasoning as is pointed out by Zweigenbaum (1990). In the long run, these two aspects need to be included in an intelligent information retrieval system.

note: For reasons af space limitations, only a restricted set of grammar rules was shown. The complete grammar as well as the full Prolog code can be found in (Spyns 1991).

References

Adriaens G., & M. Lemmens (1990): The self-extending lexicon: off-line and on-line defaulting of lexical information in the METAL MT System. In Proceedings of the 13th COLING Conference, Vol 3, 305-307.

Dorda W.G. (1990): *Data-screening and Retrieval of Medical Data by the System WAREL*, in Methods of Information in Medecine, 29, 1: 3 - 11.

Dowding J. & L. Hirschman (1988): *A Dynamic Translator for Rule Pruning in Restriction Grammar*, in Natural Language Understanding and Logic Programming Ibl, Dahl V. and Saint-Dizier P. (eds.), Elsevier Science Publishers, North-Holland, pp. 79 - 92.

Grishman R., N.Sager, C.Raze, & B.Bookchin (1973): *The Linguistic String Parser*, in AFIPS 427 - 434.

Harris Z. (1962): String Analysis of Sentence Structure, The Hague.

Hirschman L. (1986): *Conjunction in meta-restriction grammar*, in Journal of Logic Programming 4: 299 - 328.

Hirschman L. & K. Puder (1986): *Restriction Grammar: A Prolog Implementation*, in Logic Programming and its Applications M. van Caneghem and D.H.D. Warren (eds.), Ablex Publishing Corporation Norwood, New Jersey, pp. 244 - 261.

Sager N. (1972): *A Two-Stage BNF Specification of Natural Language*, in Journal of Cybernetics 2,3:39-50.

Sager N. (1981): Natural Language Information Processing, a Computer Grammar of English and its Applications, Addison-Wesley Publishing Company, New York.

Sager N., C. Friedman & M. Lyman (1987): Medical Language Processing: Computer Management of Narrative Data, Addison-Wesley Publishing Company, New York.

Sager N., M. Lyman, L.J. Tick, F. Borst, N.T. Nhan, C. Reveillard, Y. Su, & J. R. Scherrer (1989): *Adapting a Medical Language Processor from English to French*, in MEDINFO 89: 795 - 799.

Spyns P. (1991): A prototype of a semi-automated encoder for medical discharge summaries. University of Leuven Master of Computer Science Thesis [in Dutch].

Wolff S. (1984): The use of morphosemantic regularities in the medical vocabulary for automatic lexical coding. In Methods of Information in Medecine, 23: 195 - 203.

Zweigenbaum P. et al (1990): Natural language processing of patient discharge summaries (NLPAD) - extraction prototype, in Noothoven van Goor J. (ed.), AIM Reference Book. IOS, Amsterdam, 1991