A MULTI-INPUT DEPENDENCY PARSER

Salah Aït-Mokhtar **Jean-Pierre Chanod** **Claude Roux**
Xerox Research Centre Europe
6, Chemin de Maupertuis
38240 Meylan, France
{ait-mokhtar,chanod,roux}@xrce.xerox.com

Abstract

This paper describes a generic approach to compute dependencies from a variety of input ranging from raw text to syntactic trees. The dependency calculus is incremental and combines topological constraints on sub-tree patterns together with logical constraints defined as Boolean expressions over the set of dependencies. This formalism has been successfully implemented and tested with a broad coverage grammar for French, and leads to computationally efficient and linguistically deep parsing.

1 Introduction

This paper describes a general calculus to compute dependencies out of a variety of input streams, ranging from raw text to pre-analysed constituent structures. A dependency in this paper is an n-ary relation that connects words or constituents in the input text according to a specific relationship, such as standard syntactic dependencies, as in [7, 3], or, even broader relations including inter-sentential relations (e.g. coreference). The calculus relies on topological constraints selecting specific configurations within the input syntactic tree and on logical constraints applied to subsets of already computed dependency relations.

After introducing the overall system and formalism, the paper shows how the proposed calculus can handle various linguistic phenomena. The paper then presents an evaluation of a broad-coverage French grammar within the framework. The computational and linguistic performance shows that the proposed framework reconciles the need for deep syntactic representations as emphasised by linguistically motivated parsers, and the requirements for broad coverage and speed as achieved by more shallow approaches [5, 9, 4, 2].

2 Overview of the system

The system produces a set of dependency relations between linguistic objects. One of its main characteristics is its ability to take as input different kinds of linguistic objects: raw ascii texts, sequences of tokenised and morphologically analysed words, sequences of POS-disambiguated words or sequences of constituent structures, *e.g.* as produced by a chunker, provided that such input complies with the predefined XML DTD of the system. As partial or shallow parsers that produce chunked structures for raw text are now widespread, the ability for the system to process those structures is particularly suitable. It allows for a deeper syntactic analysis, featuring explicit functional relations between words, while preserving robustness. It can also be used for inter-sentential analysis.

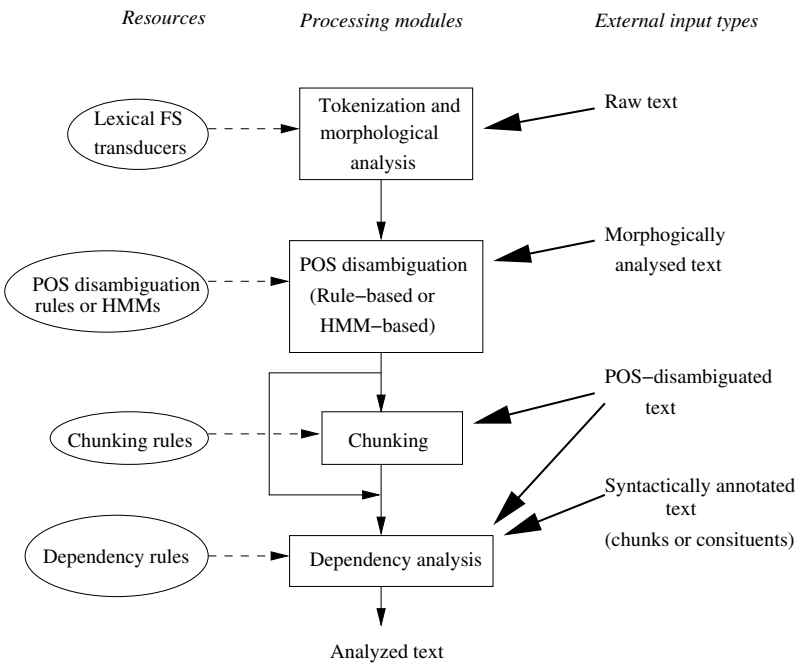

Figure 1: Architecture of the multi-input dependency parser

3 Dependency analysis

We use the term *dependency*, though the modelled relations may go beyond syntax, *e.g.* inter-sentential co-reference relations. The dependency rules state topological and logical constraints for the relation to hold. The constraints involve linear and structural properties of the constituent trees and/or the set of dependency relations that have been computed prior to the rule under scope. The structural constraints are not limited to local trees. The syntax of a dependency rule is the following:

`|<subtree_pattern>| if <conditions> <d_term>`

`<subtree_pattern>` is a tree matching expression that describes structural properties of a part of the input tree (possibly the whole tree). `<conditions>` is any Boolean expression built up from dependency terms, linear order statements and the operators `&` (conjunction), `|` (disjunction) and `~` (negation). `<d_term>` is a dependency term of the form `name<f_list>`$(a_1, a_2, ..., a_n)$, where name is the name of the dependency relation, `<f_list>` is a list of features, and $a_1, a_2, ..., a_n$ are the arguments.

A dependency rule states that a dependency term `d_term` is added to the set of dependency relations for the current input if `<conditions>` are satisfied within the current set of dependency relations and `<subtree_pattern>` matches successfully a part of the current input. All the arguments of `d_term` should be variables that are also expressed in the conditions and/or the pattern. Thus, the satisfaction of `<conditions>` and/or the successful match of `<subtree_pattern>`, as a side effect, instantiate the arguments of the new dependency term, which in turn is added to the current set of dependency relations. The following 3 examples illustrate how different types of linguistic relations are handled.

3.1 Example 1: surface relations and regular tree patterns

Dependency rules for surface syntactic relations usually feature a regular tree pattern for the constituent or chunk trees. Features associated with the nodes (or with a dependency) are within square

brackets. Sister nodes are separated with a comma. It is also possible to go down and describe the internal structure of a constituent at any level, using curly brackets. The following rule defines a verb-complement dependency relation, as between *enjoyed* and *wine* in the chunked sentence: *SC*{ *NP*{ *John*} *FV*{ *has always enjoyed*} } *NP*{ *good wine*}.} :

```
|SC{?*, FV[trans:+]{?*,#1[last:+]}},NP[time:~]{?*,#2[last:+]}|
                    vcomp[dir=+](#1,#2)
```

The rule defines a dependency of type `vcomp` between 2 words `#1` (*enjoyed*) and `#2` (*wine*) if `#1` is the head of a finite verb chunk (`FV`) that has a `trans` feature (*i.e.* accepts a direct complement), and the `FV` is within an `SC` (clause chunk) followed by an `NP` chunk with no `time` feature (*i.e.* not a time expression), and the head of which is `#2`. The `vcomp` dependency relation is assigned the feature `dir` (for direct verb complement). This sample rule has no condition on the current dependency relation set and would derive a relation noted `vcomp[dir](enjoyed,wine)` from the above example sentence.

3.2 Example 2: coordination and shared functions

Some dependency relations can be derived from logical constraints bearing on the current set of dependency relations. Handling coordinated verbs, the rule below infers a complement relation between `#1` and `#3`, on the condition that `#1` is a transitive verb (`#1[trans]`), with no direct complement (`~vcomp[dir](#1,?)`) and is coordinated with a verb `#2` that takes `#3` as a complement.

```
if (coord(#1[trans],#2) & vcomp[dir](#2,#3) & ~vcomp[dir](#1,?))
                    vcomp[dir=+](#1,#3)
```

3.3 Example 3: co-reference

The dependency rules can also define inter-sentential relations [8]. An example is given below. It shows conditions both on the current relation set and on the constituent structures. `S` is the top node of a sentence, `SC` is the top node of a subclause, `FV` is a finite verb chunk node, `subj` is the label of the subject function, and `within` is a structural relation that is true if the second argument is embedded within the first argument. The rule below selects the possible antecedent candidates for a pronoun.

```
                          |S#3,S{SC{?*,FV{?*,#1[last]}}}|
if  ( subj[imperso:~](#1,#2[pron,clit,p3,indef:~]) & within(#3,#4) )
                          coref(#2,#4)
```

It states that if a pronoun `#2` is the subject of a verb `#1` (and not an impersonal subject), and if there is a word `#4` occurring within a sentence `#3`, expressed by the relation `within(#3,#4)`, then there is a potential coreference relation between the pronoun `#2` and the word `#4`, given that the structural conditions are satisfied, *i.e.* sentence `S#3` precedes the sentence `S` in which `#1` is embedded as the finite verb of the main clause. This rule is only a first step towards co-reference resolution. Next steps apply other constraints (*e.g.* agreement control) in order to eliminate unlikely candidates.

4 Evaluation

The system has been implemented in ANSI C++ and ported to several platforms (PC/Windows, PC/Linux and Sparc/Solaris). The implementation of the engine and data structures is partly based on a previous parser [6]. Among others, this system has been used to build a broad coverage dependency

grammar for French. In its current version, the grammar defines 22 types of functional dependencies, implemented in 199 dependency rules. Lexical ressources include POS, morphological features and subcategorization information for verbs, nouns and adjectives. The French parser (from raw text to parsed text) runs at a speed of 1300 words/sec on a PC (Pentium II, 500 MHz, 128 MB of RAM). As for the evaluation of linguistic performance, we used a corpus of 50K words (articles from the newspaper *Le Monde*). We evaluated the subject dependency (including coordinated subjects, infinitive control and relative subjects) and direct complements of verbs. For subjects, precision and recall were respectively 93.45% and 89.36%, while the figures for verb complements were 90.62% and 86.56%.

5 Conclusion

This paper introduces an incremental parsing framework for deep dependency relations, at sentence and inter-sentential level. The parser accepts multiple input forms ranging from raw to chunked texts. It allows for linguistically deep robust parsing of unrestricted texts at a high speed. This system opens the way to new research development and practical applications. It is being used over a variety of corpora, for applications such as coreference resolution and knowledge extraction.

References

[1] Abney, S. P. (1991) Parsing by chunks. In R. Berwick, S. Abney, and C. Tenny (eds.), *Principled-Based Parsing*. Dordrecht: Kluwer Academic Publishers.

[2] Aït-Mokhtar, S. and Chanod, J.-P. (1997a) Incremental Finite-State Parsing. In *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP-97)*, Washington, DC, USA.

[3] Aït-Mokhtar, S. and Chanod, J.-P. (1997b) Subject and Object Dependency Extraction Using Finite-State Transducers. *ACL workshop on Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications*. Madrid.

[4] Grefenstette, G. (1996) Light Parsing as Finite-State Filtering. In *Proceedings ECAI'96 workshop on "Extended finite state models of language"*, August 11-12, 1996, Budapest.

[5] Jensen, K., Heidorn, G. E., and Richardson, S. D. (eds.) (1993) *Natural language processing: the PLNLP approach*, Kluwer Academic Publishers, Boston/Dordrecht/London.

[6] Roux, C. (1996) *Une méthode de parsage efficace pour les Grammaires Syntagmatiques Généralisées*. Ph.D, Université de Montréal.

[7] Tapanainen, P., Järvinen, T. (1997) A non-projective dependency parser. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, pp. 64–71, Washington, D.C.

[8] Trouilleux, F. (2001) (forthcoming) Thèse de doctorat en Sciences du langage, Université Blaise Pascal, Clermont-Ferrand, France.

[9] Voutilainen, A. and Heikkila J. (1994) An English constraint grammar (EngCG): a surface syntactic parser of English. In Fries, Tottie and Schneider (eds.), *Creating and using English language corpora*. Rodopi.