A Robust and Flexible Platform for Dependency Extraction

Hagège Caroline

Roux Claude

Xerox Research Centre Europe
6, Chemin de Maupertuis
38240 Meylan
(caroline.hagege,claude.roux)@grenoble.xrce.xerox.com

Abstract

This paper describes a linguistic platform, Xerox Incremental Parser (XIP hereafter), to develop robust grammars. Most robust parsers usually impose one specific strategy (constraint-based or incremental) in the grammar writing, whereas XIP allows mixing both types of analysis. The first part introduces XIP and its main functionalities. The second part illustrates how a linguist can benefit from merging different strategies in grammar writing. Finally, a first evaluation of different grammars is given.

1. Xerox Incremental Parser

XIP is the successor of IFSP, from which it borrows most of its linguistic strategies. IFSP, based on the Finite-State technology, had some drawbacks, which this new platform tries to correct. For instance, XIP provides a rich feature system and a formalism that aims at simplifying the development and maintenance of a grammar, while keeping the same parsing strategies. Furthermore, the XIP engine has been implemented in C++ to offer a platform that would be more linguistics-driven than transducers, thus improving the performance of the whole system, in terms of speed and memory footprint.

1.1. Three Level of Analysis

XIP processes a linguistic unit (phrase, sentence or even paragraph) in an incremental way. Rules are applied one after the other, whether a rule succeeds or fails. Since the system never backtracks on any rules, XIP cannot propel itself into a combinatorial explosion.

The parsing is done in three different stages:

1) The chunking and part-of-speech disambiguation.
2) The extraction of dependencies between words on the basis of regular expressions over the chunk sequence.
3) The combination of those dependencies with Boolean operators to generate new dependencies, or to modify or delete existing dependencies.

XIP starts with a translation of the linguistic unit into a sequence of part of speech. In a first pass, this sequence is disambiguated and chunked. In a second pass, the previous result is transmitted to regular expressions that extract basic dependencies between words, according to their configuration in the chunk tree. In a last pass, deduction rules mesh together those dependencies in order to trigger the creation of new dependencies. Those deduction rules can also modify or delete existing dependencies.

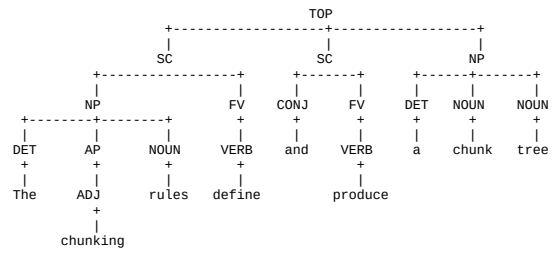
1.2. The Different Steps of Analysis

Here is an example of a sentence treated by XIP with some of the dependencies that are extracted from different part of chunk tree.

The chunking rules define and produce a chunk tree.

1.2.1. Stage 1

In a first stage, chunking rules are applied and the following chunk tree is generated for a sentence.



1.2.2. Stage 2

The next step consists in extracting some basic dependencies on that tree. Those dependencies are extracted with some very basic rules that only connect nodes that occur in a specific sub-tree configuration.

SUBJ(define,rule)
VCOORD(define,produce)

SUBJ is a subject relation and VCOORD is a coordination between two verbs.

A typical rule to extract the *subject 1* is:

| NP{?*, noun#1}, FV{verb#2}| SUBJ(#2,#1)

Where #1 and #2 are two variables that are associated with the lexical sub-nodes of a noun phrase (NP) and a finite verb (FV) that are next to each other. The "Sx{...}" denotes an exploration of the sub-nodes under the node Sx.

1.2.3. Stage 3

In the last stage, a simple Boolean expression is used to generate those new dependencies on the basis of the dependencies that have been extracted so far.

For instance, XIP generates the following dependency:

SUBJ(produce,rule)

With the following rule:

```
If (SUBJ(#2_VERB,#1_NOUN)
&VCOORD(#2_VERB,#3_VERB))
 SUBJ(#3,#1)
```

This rule reads as follow: if a subject has been extracted for a verb (#2) and a noun (#1), and a verb coordination has been found between this verb (#2) and some other verb (#3), then #3 shares the same subject as #2.

1.3. Extraction Rules

In stage 2, rules describe sub-configuration of nodes in the chunk tree, while in stage 3, rules describe some very abstract combination of dependencies. In reality, those rules comply with a unique formalism, where an extraction rule is a combination of a node configuration together with a test on previous dependencies.

For instance, here is how a subject rule could be written, that would verify whether a subject relation has already been extracted for a given verb:

```
| NP{?*, noun#1}, FV{verb#2}|
if (~SUBJ(#2,?))
SUBJ(#2,#1)
```

If no subject has been extracted for verb#2 then a subject relation is computed.

Since, the tree regular expression or the test can be omitted, a rule may be written as a stage 2 rule or as a stage 3 rule.

1.3.1. Modification of a Dependency

The last important point about XIP, is the possibility to modify the existing set of dependencies.

When an extraction rule has over-generated a certain sort of dependencies, whether the constraints are too feeble or the context is too scarce, it might be important to modify or even to delete some of those dependencies. The "^" operator is used to mark which dependency should be tacked by the rule.

N.B. A rule is a deletion rule, if the result of that rule is "~".

For instance, the rule below deletes a subject dependency that is post-positioned, if a regular subject has also been extracted.

```
if (^SUBJ[post](#1,#2) & SUBJ[pre](#1,#3))  ~.
```

We could also decide to modify that dependency into an object dependency:

```
if (^SUBJ[post](#1,#2) & SUBJ[pre](#1,#3))
OBJECT(#1,#2).
```

In the above rule, the post-positioned dependency is renamed as an object dependency.

2. Strategies

In robust dependency-based parsing, grammars or heuristics generally follow one of the following strategies:

- Incrementality (see Ait-Mokhtar and Chanod 1997) - Constraints (see Tapanainen and Jarvinen 1999)

In incrementality, the main idea is to sequentially apply rules to enrich step by step the linguistic structure that is under construction. The order in which rules are applied is very important. At each stage the structure is enriched with new information.

In constraint-based approaches, all possible relations are computed *a priori*. Then the constraints are used to destroy relations that do not satisfy them. In this kind of approach, the order of constraints is not relevant any more.

Both of these approaches have drawbacks and advantages.

The first one favors precision in the resulting analysis, while the second gives priority to recall. Furthermore, an incremental strategy may prove more adequate to describe certain linguistic phenomena, while constraint-based strategy provide a better approach to other phenomena.

For example, in English, it seems reasonable to describe a SUBJECT relation between a nominal head and a verb in an incremental way (e.g. the relation has to be established between a verb and the head of a NP that starts the sentence. If no NP is available on the left and if the sentence is an interrogative, the subject is the noun that is on the right of the DO-auxilary, etc.).

For PP attachment however, a constraint-based approach can be more appropriate. In a first step, all possible PPs' attachments are considered. In later steps, rules filter those PPs, utilizing subcategorization information, preposition form, distance or whatever information the linguist may think relevant.

Practically, with respect to the three levels of analysis presented above, the establishment of all hypothetical dependencies is done with rules from the second stage (extraction rules) on the basis of the syntactic natures of constituents, while constraints are implemented with rules from the third stage.

For instance, the following second stage rule:

```
Rule 1
| NP{?*,#1[last]};PP{?*,#1[last]},
  ?*[verb:~],
  PP{?*,NP{?*,#2[last]}}
|
MODIF(#1,#2)
```

attaches the NP head within a PP to all NP heads on left of the PP when there is no verbal form (?*[verb:~]) in between. This attachment is expressed with a MODIF dependency between those two head nodes.

While the second stage rule:

```
Rule 2
| ?[verb]{?*,#1[last]},
?*[verb:~],
 PP{?*, NP{?*,#2[last]}}
|
MODIF(#1,#2)
```

attaches, in a similar way, a PP to the first verb on its left.

Consequently, for the sentence:
He will deal in the future with other countries

We will have the following dependencies[1]:

Rule 1: MODIF(future,countries)
Rule 2: MODIF(deal,future)
Rule 2: MODIF(deal,countries)

In a later stage, the following rule of type 3 can be seen as a constraint, which expresses that:

- if a PP depends both on a verb and on nouns,
- and if this dependent PP corresponds to a subcategorized argument of the verb and does not correspond to any subcategorized argument of the nouns,
- then the dependencies between this PP and the nouns are deleted.

This rule can be formalized in the following way:

```
if (
MODIF(#1[verb],#2) &
^MODIF(#3[verb:~],#2) &
PREPD(#2,#4) &
#4[souscat]:#1[souscat] &
#4[souscat]~:#3[souscat]
 )
      ~
```

The "~" is the deletion operator, which removes from the set of dependencies, the ones that match the rule element that is preceded by a "^". In our rule this element is: ^MODIF(#3[verb:~],#2).

The two last lines deal with the comparison of the subcategorization features between two nodes. The first test in our rule checks that the subcategorization features between the verb governor and the PP are compliant, while the second test ensures that those features for the PP and the noun governor are different.

3. Evaluation

A French grammar and an English grammar have been developed so far with XIP. For the moment, only the French grammar has undergone some evaluations for 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%.

A first evaluation for the under-development English grammar has also be made.

For subject dependency (including coordinated subjects, infinitive controls, sentencial subjects and relative subjects), precision is 78.2% and recall is 85.7%.

For object dependencies (including sentential objects), precision is 80.1% and recall 74%.

Finally, in order to check the quality of attachment, we create a general dependency (MODIF) for other types of complements (sentential or not) for each kind of categories (verbs, nouns, adjectives). We obtain a precision of 72.7% and a recall of 78.7%.

4. Conclusion

The XIP formalism does not impose any pre-defined strategy, in grammar development. Hence, the possibility to choose the most suitable strategy according to some specific task.

Since the current version cannot easily tackle some peculiar syntactico-semantic problems (such as word sense disambiguation), we are currently working on new features to enlarge the coverage of XIP rules.

5. References

Abney Steve, (1991). Parsing by chunks. In *Principled-Based Parsing*, R. Berwick, S. Abney, and C. Tenny, editors. Kluwer Academic Publishers, Dordrecht, 1991.

Aït-Mokhtar Salah, Chanod Jean-Pierre, (1997). Incremental finite-state parsing. In *Proceedings of Applied Natural Language Processing 1997*, Washington, DC. April 97.

Aït-Mokhtar Salah, Chanod Jean-Pierre, (1997). Subject and Object Dependency Extraction Using Finite-State Transducers. In *ACL workshop on Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications.* 1997, Madrid.

[1] The first and the second dependencies are generated by the second rule when the third is generated by the first rule.

Aït-Mokhtar Salah, Chanod Jean-Pierre, Roux Claude (2001). A Multi-Input Dual-Entry Point Dependency Parser. In *IWPT 2001*, Beijing.

Chanod, J.P., Tapanainen, P., (1995). Tagging French - comparing a statistical and a constraint based method. In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics*. Dublin.

Roux C., (1999). Phrase-Driven Parser. In *VEXTAL 99.* Venice.

Tapanainen P., Järvinen T.(1999) A non-projective dependency parser. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, (pp.64-71) Washington, D.C.