JDII: Parsing Italian with a Robust Constraint Grammar

ANDREA BOLIOLI LUCA DINI GIOVANNI MALNATI
Dima Logic
C.so Turati 11/c, 10128 Torino Italy
fax + +39-11-501618

Abstract

Italian is a language presenting a lot of syntactical problems, such as a rather unrestricted word order, unbounded agreement controls, long distance structure checkings and so on. Things get worse and worse if we pass from "sentences of linguists" to real texts. In this paper we will present a system able to retrieve and signal syntactic errors in real italian texts.

1 Introduction

We are going to present a system by which syntactical errors of Italian can be recognized and signalled. This system is called JDII (James Dean the second) and has been developed in Turin by DIMA LOGIC.

JDII can accept wrong Italian sentences, even long and complex ones, and it returns a comment on the type of the detected mistake(s), if any. A corpus of 400 grammatical sentences (from a minimum of 4 words to a maximum of 40 words) and possible ungrammatical variants has been used to test the system. The sentences contain the grammatical phenomena treated by the grammar (see below). The system is implemented in PROLOG and C. It runs on UNIX and DOS environments.

The modularity of the system is guaranteed by the division of linguistic and software knowledge into two indipendent modules. The linguistic module is roughly made of a morphological component and a syntactic one, while the computational framework (DIMACheck, cf. chapter 3) is mainly based on a parser and on a theorem prover. Software resources are shared with another syntax checker, aiming at an analogous system for the English.

2 Processing errors

2.1 Error interpretation

An error is a violation of some constraint posed by linguistic rules on the language. The violation of these constraints causes, according to standard classification, spelling, morphological, syntactic and semantic errors. This classification, which is still useful in defining the nature of linguistic violations from an informal point of view, poses some problems in an authomatic treatment of errors. The fact is that an error can be properly classified only on the basis of writer's intentions. For example, in a sentence like 1) * Ho scritto una lettera a un ponte [1].

I wrote a letter to a bridge

We could assume:

- a semantic error, since people generally do not write letters to bridges;
- a syntactic error, since locative complements with words like *ponte* are realized by different prepositions (*sotto, su*);
- a spelling error, if the writer had not wanted to write *ponte*, but *conte* ("earl").

The fact is that people correcting texts usually have a pragmatic context which allows mistakes disambiguation. For "context" we mean all the background information concerning the writer, the external conditions when the text was written and , above all, the kind of facts and things which the text deals with. (Consider, e.g.,the different interpretations of a sentence like 1 if it were found in the answer of a fool to a psychological test, in a novel about ancient chivalry, or in some essay titled "Which is the best place for writing a letter?").

Since we are not able to handle contextual knowledge for error disambiguation, we decided, for our purposes, to drop the classification , and to make use of the notion of **error interpretation**: "Given a wrong sentence, an error interpretation is any hypothesis of substitution in order to make the sentence correct".

In correction performed by humans, the number of error interpretations is constrained by pragmatical and contextual data. In authomatic detection this number is linked to the capacity of ammitting constraint violations on the rules. In the worst case (i.e. when all the constraints, including lexical constraints, are allowed to be violated) the set of error interpretations is infinite.

2.2 Limits of the system

To get out from this empasse we chose to constrain the possible violations allowed by our grammar just to syntactic ones. We could also deal with spelling corrections by including a small set of incorrect variants for some words. This would reveal meaningful only by tuning the system on the specific linguistic background of the user (in fact different mispellings are made by people speaking different languages and with different degrees of instruction). As for semantic errors we are not able to deal with contexts, so they are simply ignored and

[1] English translations will be word by word. In the last chapter we don't provide any translation, since ill formed phrases and sentences exemplifying the coverage are language specific.

sentences like 1 are considered correct (and in fact they are, see e.g. a context like *Giovanni e' impazzito: dice di aver scritto una lettera a un ponte*). .

To sum up, in our system the error checking works as follows :

1) If a word is found the root of which is not present in the vocabolary or which cannot be properly inflected, the message "unkown word' is reported.

2) If a word is found which is included only in the set of the uncorrect variants, the morphology will return to the grammar the lexical unit properly inflected, but containing a violated constraint .

3)If all the words in a sentence have been analized by the morphological module, the syntactic processing starts.

4) As we will see, the syntactic parsing will produce either a comment on the grammaticality of the sentence or a general refuse of it (i.e. a generic error such as "unknown grammatical structure").

Ha visto un crane -> Unknown word (*crane*)

A visto un cane -> Spelling error (*a* missing *h*)

Ha visto una cane -> Syntactic error (agreement)

Ha visto cane uno -> Unknown structure

2.3 Principles of error diagnosis

In the previous paragraph we stated that our attention will be drawn only to syntactic violations. However, this limitation does not solve the problem of multiple error interpretations at all, since, even from a purely syntactic point of view, a sentence can be wrong in different ways. Let us consider a sentence like:

*2) * Il ragazzo e' stata affettuoso.*

the(masc) boy(masc) has been(fem) lovely(masc)

We have at least two hypotheses of correction:

2.a) Il ragazzo e' statO affettuoso.

the(masc) boy(masc) has been(masc) lovely(masc)

2.b) LA ragazzA e' stata affettuosA.

the(fem) girl(fem) has been(fem) lovely(fem)

If we take into account psychological plausibility, we should signal only the error on the word *stata*. This and other data support a principle in error correction which states that " given a set of possible error interpretations, the right error interpretation is the one with the smallest number of violated constraints ". This principle has been implemented as a built-in preference mechanism over the set of possible final interpretations, while the set itself is restricted by the power of the grammar. The restriction is obtained by implementing peculiar linguistic statements that

i) impose linguistically plausible criteria rather than statistical ones;

ii) prevent that the explosion of all the possible error interpretations makes the system completely inefficient.

An application of the above criteria is provided by the sentence:

*3) * Il ragazzo che e' stata picchiata dai fascisti sta male.*

the(masc) boy(masc) who has been(fem) hit(fem) by fascists is suffering

where we can hypothesize two agreement violations either in the subject NP or in the VP of the relative clause. In this case our system allows us to state that agreement features of the head will win on the ones of the modifiers, so that a gender agreement error is signalled in the relative clause.

3 DIMACheck framework

DIMACheck is a general-purpose unification-based natural language parser that, while retaining computational effectiveness and linguistic expression power, stresses the concepts of monotonicy, declarativity and robustness. These goals are achieved, on the one hand, introducing several linguistic devices, like weak constraints, user-defined operators and functions, and on the other hand enforcing strict data-type checking and implementing a time-independent evaluation function (the interpreter of the rules) that guarantees a high expressive power in a totally declarative environment.

In order to mantain readability and ease of use of grammars, only two kinds of rules have been introduced, namely structure building rules and lexical rules.

3.1 User Defined Operators

We think that a re-write system based only on equality constraints is inadequate to express linguistic knowledge, and the introduction of inequality constraints does not always solve the problem. In order to augment the linguistic expressive power without incurring in redundancy and computation-ineffectiveness we introduced the following tool. Formally a User Defined Operator (UDO) is function of the form

Boolean <- DataType1 * DataType2

i.e. a UDO is a function mapping pairs of values belonging respectively to DataType1 and DataType2 onto boolean values. The composition rule (the rule that associates the relevant boolean value to each pair) is given explicitly, by listing all the value pairs that map onto true (all the other ones are mapped automatically onto false).

3.2 User Defined Functions

User Defined Functions (UDFs) have been introduced to stress the locality of computation. The basic idea is that each value inside a constraint (be it an equality constraint or not) may, in principle, be replaced by a function that computes it on the basis of some given parameters. So, whenever one must compute the value of an attribute which is known to depend on and only on a finite set of other features,

instead of writing lots of rules which embed (and hide) this piece of knowledge into a larger description, it is possible to declare a UDF that manages the computation, thus reducing the number of rules from many to one.

Formally, a UDF is a function that maps values from N data types into values of a given data type. in symbols:

TargetDataType <- DataType1*...*DataTypeN

UDFs are declared explicitely, more or less like UDOs: for each n-tuple of relevant values the result value is stated. UDFs need not to be deterministic: a given pair of input parameters may map into more than one target value.

3.3 Constraint and constraint bundles

As stated above parsing is, in our view, applying cation a finite set of constraints over an input list of words. We may therefore distinguish between structural constraints (the ones that deal with the order, the occurrences, etc. of parse trees) and feature constraints, that put restrictions on the value of a given variable (the value of an attribute inside the parse tree). The former kind is described in paragraph 3.4, while the latter, is described here.

3.3.1 Definition of Feature Constraint

A feature constraint, or simply a constraint, is a triple of the form:

< Operator, AttributeName, ValueExpression >

Operator is the name of either a system defined operator ('=' and '~=') or the name of a user-defined one (e.g. 'is_a').

AttributeName is a legal name for an attribute, the type of which matches the type foreseen by the operator for its left- hand side.

ValueExpression may be

- an atomic value
- a single variable
- a disjunction of atomic values
- a user-defined function

In our formalism a constraint is stated in an infix form (e.g. 'tense = pres' or 'tense agreem_tense' T or 'tense = compute_tense(M,T)').

When a constraint is applied to an object it may evaluate either to true or to false: we can therefore say that a constraint is a boolean function. The way in which the result of the application of the constraint is handled by the system leads to the distinction between strong and weak constraints.

3.3.2 Strong and weak constraints

A strong constraint is a constraint that, if it fails, causes a strong failure, i.e. the object to which it applied is rejected. When a strong constraint succeeds nothing happens, apart from some possible variable binding. Strong constraints are used mainly to prevent useless overgeneration over irrelevant paths. (Usually, but not necessarily, constraints that involve the major syntactic categories are strong). They are also used to propagate values from lower nodes to upper ones.

A weak constraint is a constraint that behaves like a strong one if it evaluates to true , but which otherwise produces a soft failure. A soft failure simply consists in recording in the object the information that a weak constraint has failed, without rejecting it. In order to mantain trace of the failed constraints, they are annotated by the user with a number which is used at the end of parsing to generate a proper error message indicating which constraint failed and where. Apart from annotation, the syntax of weak constraints is the same of the strong ones, and the same restriction applies.

A constraint bundle (CB) is a list of conjuncted constraints (both weak and strong). Notationally, a CB is delimited by braces, single constraints are separated by commas; a slash ('/') splits the list in two parts: the strong one and the weak one. If the weak part is empty, the slash is omitted. Here are some examples of legal CB's:

{ cat = np / (1) nb = N, (2) gd = G }

{ cat = v, aux, vtype = V / (81) cat = v }

{ cat = pp }

3.3.3 Constraints solution

During parsing, the parse trees which are built are labelled with a list of pending constraints - i.e. of constraints that have not yet proved to be true or false- and a score - i.e. an indication of how many weak constraints associated to the tree have already proved to be false. Intuitively, the lower is the score, the better is the object. The constraint solver applies to the list of pending constraints of each final tree, trying to minimize the number of failed weak constraints. The constraint solver selects, as final result, the tree with the smallest associated error. It's worth noting that this is a global strategy, not a local one. All parse trees, independently of their score are carried on up to the end of parsing, and only then the selection is made. There are two reasons for this choice. The first one is theoretical: it is not possible to assume that a locally well formed subtree will lead to a better global tree than that produced by a locally ill-formed one. The second reason is pragmatical: since constraints are solved only when the variables they involve get instantiated, partial trees tend to contain few or no failed weak constraints but long lists of constraints still to be evaluated. Applying the constraint solver in the middle of parsing would be a waste of time, and making the choice disregarding the pending constraints is definitely wrong.

3.4 Grammatical rules

The system operates on the input data driven by rules. Rules mix together structure and feature constraints in order to produce a quasi-well-formed (sub)tree (the 'quasi' is there because the subtree may contain failed weak constraints: it would not be proper

to call it a well-formed one). Rules are handled by a parser in order to produce all possible results. Currently the system uses a bottom-up, left to right algorithm. However the result is totally independent of the parsing strategy.

Structure building rules (sb-rules) are augmented rewrite rules used to describe the structure of quasi well-formed subtrees.A sb-rule has the following form:

```
< RuleName > = < TopConstraintBundle > = >
        < SubTreeExpression1 >,
          ... . .
        < SubTreeExpressionN > .
```

where:

< RuleName > is a legal unique identifier,

< TopConstraintBundle > is a legal CB,

< SubTreeExpression1..N > are one of the following:

- a CB,
- a SubTree description (of any depth)
- a regular expression over CBs like:

'*'Exp : 0 or more occurrences of Exp
'+'Exp : 1 or more occurrences of Exp
'^'Exp : 0 or 1 occurrences of Exp
'!'Exp : 0 or 1 occurrences of Exp
(if 0 signal weak error)
'?'Exp : 0 or 1 occurrences of Exp
(if 1 signal weak error)
Exp1,Exp2 : Exp1 followed by Exp2
Exp1;Exp2 : Exp1 or Exp2

The error associated to the newly built tree is the sum of the errors contained in all its subtrees plus the errors originated by the application of all the constraints of the current rule, both in feature and in structure. Here is an example of sb-rule:

```
cNP_interr = {cat = npp,qtype = QT,wh = yes.wh_nb = N.
wh_gender = G,nb = N,gender = Gquant = no,ntype = inter,}
        = >
    {cat = detp, ntype = interr/(2)gender = G(1)nb = N},
  ?(81) + {cat = detp},
  {cat = np,wh = no,nb = N,gender = G,qtype = QT/
            (67)ntype ~ = proper_not_art:proper}
```

Lexical rules are the interface between the external representations of words (i.e. strings) and the internal ones (i.e. CBs). A lexical rules has the general form:

```
< RuleName > = < ConstraintBundle > .
```

where < RuleName > is a legal unique identifier and < ConstraintBundle > is a legal CB.

3.5 Morphological Rules

In the morphological rules, each root is associated to a morphological class and to a lexical rule. Before the syntactical parsing starts every word in a sentence has to be processed by the morphological module. The resulting CB is the union of the CB associated to the ending of the word and the CB defined in the lexical rule associated to the root.

4 Linguistics

JDII does not make strict use of any linguistic theory, even if the guidelines of the implementation are, in a large number of cases, taken from theoretically well founded works (such as Burzio (1986) for the verbal system, Gazdar (1981) for comparative structures, Cinque (1988) for relative clauses and so on). On this respect we fully agree with Dietmar Roesner when he says that "Theorist tend to restrict their approaches to the very techniques available within their theories. In practical NLP systems it may be fruitful to freely combine elements from distinct "linguistic schools"."

Structurally a binary recursive X-BAR schema is followed. The reason why a binary grammar is used is that we lack of a dedicated kind of rule for performing structural checking. As a consequence the computation of constraints depending either on the occurrence or on the linear precedence of optional constituents would reveal very difficult. Indeed, since UDOs and UDFs are not allowed to contain optional parameters, a non binary grammar could handle strings like

```
X[agreem oper(V1 V2)] - > Y *B ^ A[s_a = V1] *B ^ A[s_a = V2]
```

(where 'agreem' depends on the values of 'V1' and 'V2' and on the presence of 'A') only by exploding all the possible cases. This huge and inefficient explosion is avoided in a system of rules where 'X' is right recursive and the 'agreem' value is updated by an UDF when every new projection is built:

```
X[agreem = default] - > Y
X[agreem = VAR0] - > X[agreem = VAR0] B
X[agreem = funct(VAR1,VAR2)]- > X[agreem = VAR1]
                                A[agreem = VAR2]
```

As for ambiguities, we are not interested in producing all possible interpretations of a sentence. Indeed the production of all semantically plausible parses would be out of the scope of a syntax checker, which is supposed to handle only ambiguities relevant for error retrivial.

4.1 The coverage

The coverage of JDII includes:

- main sentences, both affirmative or negative
- argumental clauses playing the role of either subject or object
- hypothetical clauses
- comparative and consecutive clauses
- prepositional clauses
- relative clauses
- participial clauses
- gerundive clauses

As for the other constituents, we have a complete treatment for each possible phrasal projection (AP's,

NP's, VP's ...)

Particular attention has been drawn to the following phenomena:

- quantification (e.g. * *tutte ragazze*, * *molte di ragazze*, * *nessuno delle ragazze*, ...)
- determination (e.g. * *la Maria*, * i *cromi*, * *della ragazza verra'*, ...)
- coordination (e.g. * *la ragazza bella e sensuali*, **la ragazza e la sua amica e' venuta*, ...)
- movements

wh-movement (e.g. * *la ragazza che Andrea ama Maria*, * *la ragazza che dicono che e' stato amato*, * *la ragazza che dicono che dovrebbe essere stato amato*, ...)

clitic climbing (e.g. * *li ha amato*, * *li deve aver amato*, * *deve averli amato*, ...)

- dislocations

topicalization in coordinate structure (*e.g.* **e' venuta Maria ma Moana*, * *Maria e' venuta ma Moana* vs. *non e' venuta Maria ma Moana*, *non Maria e' venuta ma Moana*, ...)

comparative structure (e.g. * *ho dato tanti baci ieri a Maria che a Moana* vs. *ho dato piu' baci ieri a Maria che a Moana*, ...)

In particular the last four phenomena worked as a test bench in order to check the power and the efficency of the formalism w.r.t. hard tasks, such as unbounded distance structure checking, long distance agreement, discontinuous patterns and so on.On the contrary the formalism proved to be inadequate to tackle context sensitive phenomena such as ellipsis in coordination and comparison, when more than one constituent is bound by the deleted element. In these cases a principle does apply which imposes a context sensitive corrispondence (X Y Z W... X Y Z W....) between the constituents in the second conjunct / comparative clause and the ones in the main clause:

4)Da' piu' baci Maria a Ugo che schiaffi Eva a Luca
NP NP PP ... NP NP PP

4.2 The errors

In the following we give a description of the main kinds of errors that our system is able to diagnose.

4.2.1 Spelling errors

If a word is found whose root is not present in the set of roots or whose inflection is not in the proper inflectional class, the message "unknow word' is given. Unfortunately, if a word is mispelt in such a way that the morphology will recognize it as a word of another category (e.g. *ha visto un corro*, where *corro* is mispelt for *carro*, but it is however present in the morphology as the first singular person of the verb *correre*) the system is likely to produce a generic "Unknown grammatical structure" message (indeed there are no grammatical rules able to deal with completely wrong structures, such as "Verb-det-Verb"). Just in a few cases we have specific morphological rules which guess a wrong interpretation of the input word (e.g. *e* is interpreted also as *e'* and *a* as *ha*).

4.2.2 Inflectional violations.

* agreement on number, gender, person: e.g. subj < - > verb, determiner < - > nbar, clitic < - > past participle, past participle < - > displaced quantifiers
* case: it is controlled just when personal pronouns are involved
* tense and mood agreement: main verb < - > suburdinate verb in hypothetical clauses, consecutio temporum, adverbials < - > main verbs.

4.2.3 Structural violations

* missing elements: determiners in particular constructions (**tutti documenti*, **molti di lettori*), negation with particular adverbs (**ho visto nessuno*)
* exceeding elements: repetition of determiners, wrong number of NP's with certain kinds of verbs (**Veronica collabora Veronica*), repetition of certain adverbs (**collabora neppure solo*)....
* wrong word order: position of the AP w.r.t. the noun (**il chirurgico intervento*), position of the quantifier w.r.t. the verb (**molto collabora*), position of the adverb w.r.t. to the verb (**Veronica neppure lavora*)
* wrong head-argument selection: selection of the complementizer in objective clauses (**la voglia che amare*), selection of the preposition in head-shifting constructions...

References

- Carbonell,J.,Hayes,P.,"Recovery Strategies for parsing Extragrammatical Language", in AJCL,1983.
- Dietmar Roesner "Why implementors of practical NLP systems can not wait for linguistic theories", in Coling 88, 1988.
- Burzio,L., "Italian Syntax", 1986.
- Cinque,G., "La frase relativa", in Grande Grammatica Italiana di Consultazione,1988.
- Gazdar,G.,"A Phrase Structure Syntax for Comparative Clauses", in Hoekstra, T., van der Hulst,H., Moortgat,M.,ed: Lexical Grammar,1981.
- Jensen,K. et al., "Parse fitting and Prose Fixing: Getting a Hold on Ill-formedness", in AJCL, July-December 1983.
- Menzel,W., "Error diagnosis and selection in a training system for second language learning", in Coling 88, 1988.
- Shwind , C.,"Sensitive parsing:error analysis and explanation in an intelligent language tutoring system", in Coling 88, 1988.
- Thurmair,G.,"Parsing for Grammar and Style Checking", in Coling 90, 1990.