A Dependency-based Algorithm for Grammar Conversion

Alessandro Bahgat Shehata and Fabio Massimo Zanzotto

DISCo
University of Milano-Bicocca
Milano - Italy
zanzotto@disco.unimib.it

Abstract

In this paper we present a model to transfor a grammatical formalism in another. The model is applicable only on restrictive conditions. However, it is fairly useful for many purposes: parsing evaluation, researching methods for truly combining different parsing outputs to reach better parsing performances, and building larger syntactically annotated corpora for data-driven approaches. The model has been tested over a case study: the translation of the Turin Tree Bank Grammar to the Shallow Grammar of the CHAOS Italian parser.

1. Introduction

Different syntactically annotated corpora as well as different syntactic parsers generally realize different grammatical theories. This fact intrinsically limits some very important activities such as parsing evaluation (as noted for example in (Carroll et al., 1998)), researching methods for truly combining different parsing outputs to reach better parsing performances, and building larger syntactically annotated corpora for data-driven approaches (e.g. (Collins, December 2003)). In languages other than English where de-facto standards (like the Penn Treebank (Marcus et al., 1993)) are still not available, this problem is even more important. Relevant efforts are addressed in building possibly diverging linguistic resources and tools. In Italian, that is the language we are interested in this study, there are at least three different syntactically annotated corpora: the Turin Treebank (Bosco et al., 2000), the Venice Italian Treebank (Delmonte, forthcoming), and the ISST (Barsotti et al., 2001). None of them is comparable in size with the Penn Treebank (Marcus et al., 1993). Nevertheless, all follow different annotation schemes and different grammars. Moreover also some syntactic parsers for Italian exist (e.g. CHAOS (Basili and Zanzotto, 2002)), but again they do follow peculiar grammar theories.

In order to efficiently exploit all these resources for building better syntactic parsers, we are exploring the possibility to define standard methods to convert a grammar formalism in another. This can only be done in the strict conditions that the target syntactic theory produces annotations less informative than the source. Even with this limits, we believe that this can provide better resources for evaluating existing syntactic parsers such as CHAOS (Basili and Zanzotto, 2002) or larger training sets where to experiment state-of-the-art statistical parsers as done in (Corazza et al., 2004).

In this paper we then present a dependency-based grammar conversion algorithm that has been defined in line with what suggested in (Lin, 1995; Basili et al., 1998). The method has been currently applied for the conversion if an existing Treebank, the TUT[1] (Bosco et al., 2000) to the grammar of an existing parser, CHAOS[2] (Basili and Zanzotto, 2002).

The rest of the paper is organised as follows. Sec. 2. describes the extended dependency graph (XDG), the meta-grammar formalism that we use to encode the grammars. Sec. 3. explains our transformation algorithm. Finally, Sec. 4. describe our case study: the transformation of the TUT grammar in the CHAOS grammar.

2. Syntactic Graph Formalism: Extended Dependency Graph

We rely on the extended dependency graph (XDG) (Basili and Zanzotto, 2002) as syntactic representation. An XDG is basically a dependency graph whose nodes C are *constituents* and whose edges D are the *grammatical relations* among the constituents, i.e.:

$$\mathcal{XDG} = (C, D)$$

This representation is fairly useful when translating a grammar formalism G in another G'. This has the possibility of representig both a fully constituent-based tree and a fully dependency-based tree or graph. It includes also the feature structure formalism.

Moreover, from the point of view of a modular processor the XDG has relevant two relevant properites: it hides unnecessary ambiguity in eventually underspecified constituents and it may represent alternative interpretations in a single graph.

Constituents, i.e. the elements of C, are classical syntactic trees with explicit *syntactic heads* and *potential semantic governors*. Constituents can be represented as feature structures, having as relevant features:

- the $head$ and the gov, having as domain $\mathcal{C}$ (the set of trees and subtrees derived from C), and representing respectively *syntactic heads* and *potential semantic governors*;

[1]The TUT has been downloaded at http://www.di.unito.it/∼tutreeb

[2]The CHAOS parser has been downloaded at http://ai-nlp.info.uniroma2.it/external/chaosproject

- the $type$ representing the syntactic label of the constituent and having as domain Λ.

Moreover, a constituent can be either *complex* or *simple*. A *complex constituent* is a tree containing other constituents as children (which are expressed by the feature $subConstituents$). A *simple constituent* represent a leaf node, i.e., a token span in the input sentence, that carries information about lexical items through the following features:

- $surface$, representing the actual form found in the token span,
- $lemma$, taking values in the lexicon $\mathcal{L}$ and representing the canonical form of the target surface,
- $morphology$, representing the morphological features of the inflected form.

On the other hand, dependencies in D represent typed and ambiguous relations among a constituent, the $head$, and one of its $modifiers$. The ambiguity is represented using $plausibility$, a real value ranging between 0 and 1, where 1 stands for unambiguous. Then, D is defined as a subset of $C \times C \times \Gamma \times (0, 1]$, where the sets represent respectively the domains of the features $head$, $modifier$, $type$, and $plausibility$.
Given a constituent or a dependency e and a feature $\mathcal{F}$ we will use $\mathcal{F}(e)$ to refer to the actual value of the feature $\mathcal{F}$ in c (e.g., $gov(the\_red\_cat) = cat$).

3. The translation model

Our main objective is to design an algorithm that translates annotations made in a grammatical model G to annotations made in another grammatical model G'. This algorithm should give the possibility to minimize the information loss and the distortion of the meaning of the data as it proceeds with the translation.
Given a sentence in the corpus, the translation model implements a function that has the form:

$$\mathcal{XDG}' = Translate(\mathcal{XDG}, \rho) \quad (1)$$

where $\mathcal{XDG}'$ and $\mathcal{XDG}$ are the source and the target annotation for the analysed sentence. These are respectively written according to G and to G'. The set ρ is the set of the translation rules. As we will see, this function will be realised as a cascade of more simple steps

$$\mathcal{XDG}_{i+1} = T_i(\mathcal{XDG}_i, \rho_i) \quad (2)$$

where $\mathcal{XDG}_i$ are intermediate transformations and ρ_i is the set of rule of the transformation T_i. Considering ρ_i a parameter and indicating $T_i(\mathcal{XDG}_i, \rho_i)$ as $T_i(\mathcal{XDG}_i)$, the overall transformation is seen as:

$$\mathcal{XDG}' = Translate(\mathcal{XDG}, \rho) = T_n \circ \ldots \circ T_1(\mathcal{XDG}) \quad (3)$$

We want this algorithm to be clearly customizable to possibly arbitrary source and target annotation grammar schemes. Using this translation model requires then some necessary activities:

1. the comparative analysis the two formalisms: source G and target G'
2. the assesment of the restrictive hypotheses
3. if the restrictive hypotheses are met, the writing of the translation rules

The translation model should then offer a clear language to express translation rules and a clear definition of how these rules are applied in the cascade of transformations. The step 3 requires the writing of the translation rules for each processor and the definition of the correct cascade.
In the following we will first of all define the restrictive hypotheses (Sec. 3.1.). Then we will give a sketch of the overall model (Sec. 3.3.) and on the admitted transformations and finally (Sec. 3.3.) we will describe the possible transformation processors that can be adopted: feature transformer (Sec. 3.3.1.), dependency transformater (Sec. 3.3.2.), and, finally, the constituent aggregator (Sec. 3.3.3.). A preliminary step, hereafter called *format transformation*, is clearly needed to import the (constituency or dependency-based) graphs in the XDG formalism (Sec. 3.2.).

3.1. Restrictive Hypotheses

In the construction of this kind of algorithm we have to make two assumptions:

- the grammar G has more expressive power than the grammar G';
- the translation is possibly a function, i.e. there is no grammatical information in G that could have multiple translations in G'.

These become the *restrictive hypotheses* of the applicability of the proposed model.
In particular, assuming that G has more expressive power than G' has an important counterpart. It is always possible to map data expressed using the first grammar to an equivalent form using the second one. The second assumption limits the possibility of ambiguous translations: if a specific input graph could be translated in multiple output graphs, then a choice would be required to discriminate between all the possible output meanings. If those requirements are met, it is possible to model the translation between different grammars as a set of deterministic rule-based process.

3.2. Format transformation

The *format transformation* deals with the transformation of an input graph Gr, represented according to a model G, in preliminary extended dependency graph, $\mathcal{XDG}_1$, that represents the source graph in a format closer to the one used by the XDG model. The resulting graph $\mathcal{XDG}_1$ has roughly the same nodes and arcs of Gr. The process mapped constituents in constituents and relation between constituents in depencencies.
The two extremes are treated as follows. A completely dependency-based graph is mapped to an extended dependency graphs where source nodes are represented by *simple constituents* and the arcs to *dependencies* between these

simple constituents. For a completely constituency based graph, as a first step, the structure is replicated in the XDG. This constituent is then flattened to a dependency based graph in this way. Simple constituents will be the nodes of the XDG. Each complex constituent c will give $n-1$ depencencies where n is the number of direct sons. The dependencies derived for c will be drawn from the simple constituent that is the potential governor of c and the potential governors of the direct sons of c.

3.3. Model transformation

The *model transformation* phase is the most important one, and it is here that the actual translation takes place. After the format transformation, the graph $\mathcal{XDG}_1$ has a structure that is consistent with the XDG format, but its elements (nodes and arcs) might have feature structures expressed using a format and lexicon specific of the source grammar. Furthermore, the original grammar G and the grammatical model G' used to represent the final XDGs (called from now on $\mathcal{XDG}'$) could associate different interpretations (represented as sub-graphs) to the same grammatical phenomena, and therefore, the graph $\mathcal{XDG}_1$ could have, according to G', a meaning that is slightly different than the one of the original Gr in G.

Because of the previous considerations, the preliminary $\mathcal{XDG}_1$ is modified by a series of transformations with the purpose of making it consistent to the expectations of the destination grammar. This phase is composed by two main kinds of transformations: the first one deals with the feature structures associated with particular elements of the graph (nodes or arcs).

3.3.1. Translation of the feature structures

These transformations are mostly rules that specify how to translate a feature structure of G into another one expressing the same (or, eventually, a more generic) meaning when translated to the format used by the resulting XDG. In particular, these rules specify how to map a feature names and values of a source grammar to the correspondent ones in the destination one. For each feature, it is possible to define a table that specifies how to translate a source value, expressing a concept in the original grammar, to another value, expressing the same meaning in the destination grammar. These translation tables can usually be created just by observation and analysis of the two grammars involved in the translations.

3.3.2. Dependency transformations

The second kind of transformations is more important because it allows to modify particular structures in the connections between nodes of the graph $\mathcal{XDG}'$. This is useful to deal with particular grammatical phenomena that generate sub-graphs with different connection structure when analyzed according to G or G'. Many of these cases have been described in previous works (Lin, 1995), and we will make just an example: coordination.

As can be seen in Figure 1 coordination can be represented in, a least, two different manners. After the previous transformations, the intermediate $\mathcal{XDG}_1$ will have a set of constituents and dependencies, expressed in the correct form,

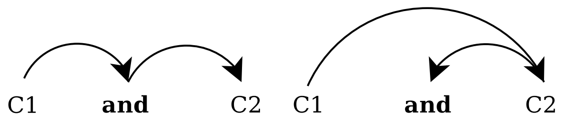


Figure 1: Different representations of coordination between elements.

but the structure of the dependencies, will reflect the assumptions made by the source grammar. Our algorithm, therefore, must be able to detect coordinations (and other patterns of dependencies) in the intermediate graph, and replace them with an equivalent set of dependencies expressing coordinations in the format used by the destination grammar.

Therefore, the rules that define these transformations specify how to transform particular patterns of connections in other patterns, modifying the structure of the graph to reflect the differences between the two grammars.

Usually, these transformations can be designed by a priori observation of the two grammars, resulting in an initial set of rules, but it is necessary to compare the interpretation of the same sentences in the two grammars to grasp the more subtle differences between them. Therefore, it appears to be a good idea to develop a preliminary set of dependency modification rules, and evaluate it using a set of sentences as test set.

If the rules being tested fail to preserve the meaning of dependency structures after the translation algorithm, they can be used to analyze the differences between the grammatical models, and develop and improved rule set. This process can be iterated until a satisfying set is found.

These rules can be realized specifying constraints on the type of dependencies and features of the nodes being connected. If a set of dependencies S that satisfies all the constraints is found in $\mathcal{XDG}_1$, it is transformed according in another set S'. The transformation is usually performed by changing the source and target nodes of the dependencies in S, resulting in redirection the connections between the nodes.

This last step completes the translation process, and produce the final graph (in this case an XDG) representing the original data in the new format we chose to adopt.

3.3.3. Constituent aggregation

Some grammatical models involve aggregation of simple units of the original sentence in complex ones (e.g. the one adopted by the Chaos parser, described above). To handle correctly this kind of grammars, it is necessary to define how to handle aggregation in the translation process.

If simple units of the original sentence (single words, for example) are aggregated in more complex constituents, it can be necessary to update the set of dependencies to reflect the new structure of the nodes. Some links may connect simple constituents that have been merged into the same complex constituent: in this case the information carried by the dependency is captured by the aggregation itself, since the two nodes have been included in the same higher-level constituent.

The most important links, however, are the ones that, after

aggregation, connect different constituents: in this case it is useful to analyze in more detail the dependency patterns.
The particular policies that determine how to handle dependency links after the aggregation depend on the structure of the complex constituents. If there are any simple nodes (constituents) that have a special role in the higher-level unit that includes them, the output graph should contain only links that connect only this kind of nodes. Therefore, dependencies that do not satisy this criterion should be examined in more detail, as they can provide some useful insight about the similarities and differencies of the two models involved in the process.

4. Case study

We applied this model to transform for the Turin University Treebank (Bosco et al., 2000) in the grammatical formalism of the Chaos parser, based the extended dependency graph (XDG). The TUT uses a particular dependency based model, that associates feature structures to both dependencies and constituents. It represents sentences as trees, having single words as nodes and each link corresponds to a functional connection between nodes.

4.1. Transforming the TUT Grammar in the Chaos Grammar

To deal with some specific details of the two formats involved in the translation, we had to include some intermediate transformations to the process described previously.

4.1.1. Format conversion

The first step of the algorithm is identical to the one described above: each node of the dependency tree is transformed in a corresponding simple constituent of an XDG, translating its original type to the equivalent one in the grammar adopted by Chaos, but copying the original feature structure without any modification. A similar process is applied to the dependencies.

4.1.2. Translating node and arc feature structures

In the following phase, all the feature structures are translated according to the translation tables described in 3.3.1., Table 1 shows, for example, the translation table associated with the verbal mood feature.

TUT	Chaos
CONDIZ	cond
CONG	cong
GERUND	geru
IMPER	imper
IND	ind
INFINITE	inf
PARTICIPLE	part

Table 1: Translations for values of the *mood* feature

Those tables were obtained by comparing the two grammars involved in the translation, and by comparing them. Each table handles a specific feature, and each entry describes how to map the values used in the TUT to the corresponding ones used in the Chaos grammar.

So far, the algorithm we used is exactly the same as the one described in the previous section, but, after the translation of the feature structures, we have to perform a series of minor transformations that deal with specific differenes between the source and destination formalisms.

4.2. Aggregating constituents using chunking

Even if most of these transformations deal with less important aspects of the translation process, it may be worthwhile to describe on specific transformation: chunking.
Therefore, the nodes of the intermediate XDG, produced after the first phase of the process, are processed by a particular module, called *chunker*, that groups them to form complex constituents, selecting grammatical heads and potential semantic governors.
The translation process should select only links that connect grammatical heads and potential governors of different constituents, discarding dependencies between consitituents belonging to different complex ones if they have none of these two roles.
If there are no links to discard using this policy, it means that there are no conflicts between the original dependencies and aggregation in complex constituents performed by the chunker module.

4.2.1. Dependency transformation rules

The last, and most important, step in the translation process, is the one that deals with dependencies, described in 3.3.2.. In this case, in particular, importing TUT dependency trees as Chaos XDGs without any elaboration of the arcs of the graph would result in the alteration of the meaning associated to the original graph.
It was necessary, therefore, to design a set of rules to handle all the grammatical phenomena that are represented in different ways by the grammars used in the TUT and Chaos. The rule below, for example, is used to handle the redirection of dependencies associated with coordinations. The grammars adopted by the Chaos parser and the TUT associate different graphs to this specific phenomenon (as anticipated in Figure 1), therefore it is necessary to detect all dependencies that represent coordinations in the original model and process them to represent coordination according to the destination model.

Input

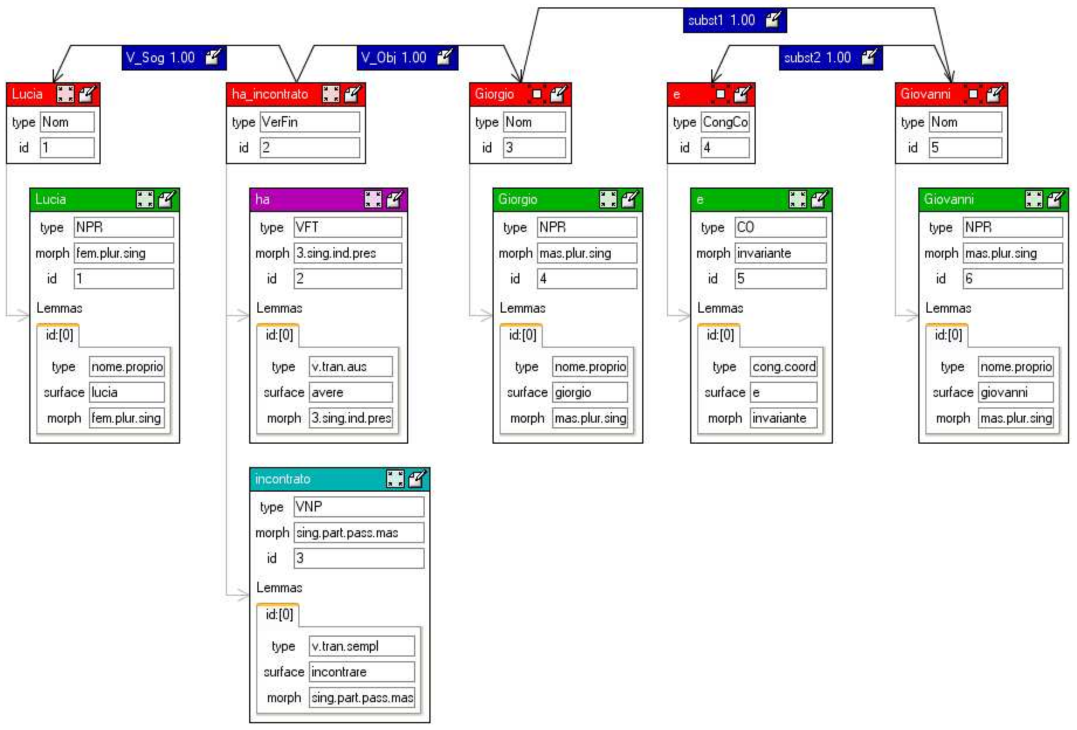
head	modifier	type	plausibility score
$f1	$t1f2	“COORD.*”	$pl1
$t1f2	$t2	“COORD2ND.*”	$pl2

Output

head	modifier	type	plausibility score
$f1	$t2	“coord1”	$pl1
$t2	$t1f2	“coord2”	$pl2

Table 2: The rule used to transform coordinations

The rule described by Table 2 can be intuitively described as composed by a set of premises (listed under Input) and a set of consequences (Output). In addition, a rule can have also a set of constraints, that needs to be satisfied by the premises of the rules.



subst1 1.00

V_Sog 1.00

V_Obj 1.00

subst2 1.00

Lucia: type Nom, id 1

ha_incontrato: type VerFin, id 2

Giorgio: type Nom, id 3

e: type CongCo, id 4

Giovanni: type Nom, id 5

Lucia: type NPR, morph fem.plur.sing, id 1; Lemmas id:[0]: type nome.proprio, surface lucia, morph fem.plur.sing

ha: type VFT, morph 3.sing.ind.pres, id 2; Lemmas id:[0]: type v.tran.aus, surface avere, morph 3.sing.ind.pres

Giorgio: type NPR, morph mas.plur.sing, id 4; Lemmas id:[0]: type nome.proprio, surface giorgio, morph mas.plur.sing

e: type CO, morph invariante, id 5; Lemmas id:[0]: type cong.coord, surface e, morph invariante

Giovanni: type NPR, morph mas.plur.sing, id 6; Lemmas id:[0]: type nome.proprio, surface giovanni, morph mas.plur.sing

incontrato: type VNP, morph sing.part.pass.mas, id 3; Lemmas id:[0]: type v.tran.sempl, surface incontrare, morph sing.part.pass.mas

Figure 2: Transformation of the example in the CHAOS grammar

The rules used in the translation process were determined at first just by comparing the source and destination grammar, and by trying to develop a rule set that could be most accurate possible, obtaining a preliminary rule set ρ. Since these rules are based just on the observations of the two grammars, it is possible that they fail to capture the most subtle differences between the two grammars. For this reason, we used a set of 20 sentences from the TUT (we will call this the development set) to test the accuracy of ρ and analyze is performance and consistency in translation. By comparing the output of the translation with the expected result, we iteratively optimized the ρ rule set obtaining a new rule set called ρ'. We evaluated the performance of ρ' in translating randomly chosen sentences from the whole TUT.

4.2.2. A walk-through example

We will now describe the application of the rules described above to a simple sentence, represented in the TUT format as shown in Figure 3:

Lucia ha incontrato Giorgio e Giovanni.

The first step of the algorithm is really simple: the feature structures associated to the words in the input graph are translated in their Chaos equivalent.

The next relevant transformation deals with chunking (4.2.): the simple constituents in the node graph are grouped in complex ones, when applicable. In this case, *ha* and *incontrato* are aggregated in a single higher level entity, and other words are left as single nodes. It can be noted that the AUX+TENSE dependency between *ha* and *incontrato* is dropped since both words are grouped together.

The last step deals with dependencies: in this case, VERB-SUBJ and VERB-OBJ are translated to their equivalents, V_Sog and V_obj and, more important, coordination dependencies are redirected and relabeled according to the rule described in Table 2.

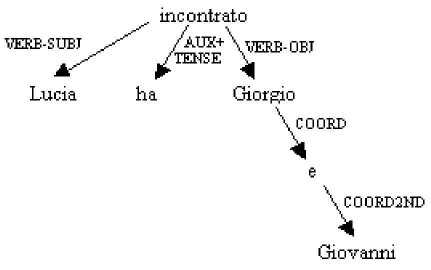


Figure 3: Representation of the example sentence as a TUT tree.

This process produces the output XDG (Fig. 2). Each word has been included in its own chunk. The two words, *ha* and *incontrato*, have been aggregated in one chunk. Finally, the conjunction has been treated.

5. Conclusions

In this paper we present a model to transfor a grammatical formalism in another. The model is applicable only on restrictive conditions. However, it is fairly useful for many purposes: parsing evaluation, researching methods for truly combining different parsing outputs to reach better parsing performances, and building larger syntactically annotated corpora for data-driven approaches. The model has been tested over a case study: the translation of the Turin Tree Bank Grammar to the Shallow Grammar of the CHAOS Italian parser. The translation model is available in the Chaos distribution. The Chaos Parser as well as the translation model is downloadable at http://ai-nlp.info.uniroma2.it/external/chaosproject .

6. References

F. Barsotti, R. Basili., M. Battista, N. Calzolari, O. Corazzari, R. Del Monte, F. Fanciulli, N. Mana, M. Massetani, S. Montemagni, M.T. Pazienza, F. Pianesi, R. Raffaelli, D. Saracino, A. Zampolli, and F.M. Zanzotto, 2001. *The Italian Syntactic-Semantic Treebank: Architecture, Annotation, Tools and Evaluation*. Kluwer, Dordrecht, Germany.

Roberto Basili and Fabio Massimo Zanzotto. 2002. Parsing engineering and empirical robustness. *Natural Language Engineering*, 8/2-3.

Roberto Basili, Maria Teresa Pazienza, and Fabio Massimo Zanzotto. 1998. Evaluating a robust parser for italian language. In *Proc. of the Workshop on Evaluation of Parsing Systems, held jointly with 1st LREC*, Granada, Spain.

C. Bosco, V. Lombardo, D. Vassallo, and L. Lesmo. 2000. Building a treebank for italian: a data-driven annotation schema. In *Proc. 2nd International Conference on Language Resources and Evaluation LREC 2000*, pages 1420–1425, Athens, Greece.

John Carroll, Ted Briscoe, and Antonio Sanfilippo. 1998. Parser evaluation: a survey and a new proposal. In *Proc. of 1st International Conference on Language Resources and Evaluation*, Granada, Spain.

Michael Collins. December 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4).

Anna Corazza, Alberto Lavelli, Giorgio Satta, and Roberto Zanoli. 2004. Analyzing an italian treebank with state-of-the-art statistical parsers. In *Proceedings of the 3rd Workshop on Treebanks and Linguistic Theories (TLT-2004)*, Tbingen, Germany.

Rodolfo Delmonte, forthcoming. *Strutture sintattiche dallanalisi computazionale di corpora di italiano*. Franco Angeli, Milano, Italy.

D. Lin. 1995. A dependency-based method for evaluating broad-coverage parsers. In *Proc. of the 14th IJCAI*, pages 1420–1425, Montreal, Canada.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19:313–330.