Natural Language Engineering **8** (2/3): 145–165. © 2002 Cambridge University Press
DOI: 10.1017/S1351324902002917 Printed in the United Kingdom

Full parsing approximation for information extraction via finite-state cascades

FABIO CIRAVEGNA
Department of Computer Science, University of Sheffield,
Regent Court, 211 Portobello Street, Sheffield S1 4DP, UK

ALBERTO LAVELLI
ITC-irst Centro per la Ricerca Scientifica e Tecnologica,
via Sommarive 18, 38050 Povo (TN), Italy

(*Received 12 July 2001*)

Abstract

This paper proposes a robust approach to parsing suitable for Information Extraction (IE) from texts using finite-state cascades. The approach is characterized by the construction of an approximation of the full parse tree that captures all the information relevant for IE purposes, leaving the other relations underspecified. Sequences of cascades of finite-state rules deterministically analyze the text, building unambiguous structures. Initially basic chunks are analyzed; then clauses are recognized and nested; finally modifier attachment is performed and the global parse tree is built. The parsing approach allows robust, effective and efficient analysis of real world texts. The grammar organization simplifies changes, insertion of new rules and integration of domain-oriented rules. The approach has been tested for Italian, English, and Russian. A parser based on such an approach has been implemented as part of *Pinocchio*, an environment for developing and running IE applications.

1 Introduction

Most successful approaches in IE (Appelt, Hobbs, Bear, Israel and Mabry 1993; Grishman 1995; Aone, Halverson, Hampton and Ramos-Santacruz 1998) make a very limited use of syntactic information. They are generally based on finite-state techniques that only partially exploit syntactic information. They perform a first step of shallow parsing for the analysis of (non recursive) NPs and Verbal Groups (VGs). Then domain-dependent regular patterns are applied to trigger primitive actions that fill templates; meta-rules precompile patterns to cope with different syntactic clausal forms (e.g. passive forms). According to Aone *et al.* (1998), one of the main limitations of such IE technology is the inability to extract (and represent) syntactic relations among elements in the sentence; however, complex IE tasks need the correct treatment of syntactic relations at both sentence and text level. Discourse relations in particular need the correct modeling of syntactic relations. As Kameyama (1997) noted, the lack of syntactic relations may decrease accuracy in discourse processing.

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

Full parsing methodologies have been adopted for IE in the past (Hobbs, Appelt and Mabry 1992), but they tend to be slow (because of huge search spaces) and brittle (because they generally use a closed-world assumption with respect to the grammar and hence they can be fooled by gaps in coverage). The use of big grammars partially solves the problem of gaps but worsens the problem of huge search spaces and makes grammar modifications difficult (Grishman 1997).

In general, grammar modifications are an interesting point for evaluating parsing approaches for application-oriented systems such as IE systems. When coping with real world applications, modifications are always to be taken into account, as many domain-specific texts present idiosyncratic phenomena that require non-standard rules. Often such phenomena are limited to specific cases only (e.g. some types of coordinations are applied to people only and not to organizations). Note that introducing generic rules for domain-specific constructions may introduce (useless) extra complexity into the parser search space and – when applied indiscriminately (e.g. on classes other than people) – can worsen the system results. Finite-state approaches to IE are specifically designed for defining domain-specific grammars. Their grammars are usually limited in size, hence grammar modifications are easily implemented. Generic full parsing methodologies generally adopt large (or even very large) generic grammars. Inserting new rules in big grammars is a difficult task (Grishman 1997). Moreover it is not clear how semantic restrictions can be introduced into generic grammars.

In this paper, we propose a parsing approach for IE based on finite-state cascades. The parser aims at building a syntactic representation equivalent (from an IE point of view) to the ones built by systems based on full parsing. Starting from Abney's work on finite-state cascades for parsing (Abney 1996) we have developed:

- a robust method for efficiently and effectively performing full parsing on texts for IE purposes; and
- a methodology of organizing generic grammars that simplifies changes, insertion of new rules and especially integration of domain-oriented rules.

The proposed approach has been tested mainly for Italian, but demonstrators based on such an approach have also been successfully developed for English and Russian.

Moreover, the approaches to both analyzing via sequences of cascades of finite-state rules and organizing grammars have been extended to all the modules in the architecture of *Pinocchio*, a tool for IE applications. Also, lexical (lexical normalization and preparsing), semantic (default reasoning and template filling) and discourse modules are based on the same approach. They actually share with the parser the same development, debugging and runtime environment, i.e. they use the same rule editor, compiler, interpreter and tracer. *Pinocchio* (Ciravegna, Lavelli and Satta 2000a), was initially developed as part of FACILE (Ciravegna, Lavelli, Mana and others 1999; Ciravegna, Gilardoni, Lavelli and others 2000b), a successfully completed project funded by the European Union. FACILE deals with text classification and information extraction from text in the financial domain. Applications

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

and demonstrators have been built by using *Pinocchio* in a number of different domains for different languages.

This paper is organized as follows. In section 2 the adopted formalism is introduced. Then in section 3 we describe how parsing is accomplished, and in section 4 we underline the relevance of grammar organization. In section 5 the proposed approach is compared with the state of the art, and in section 6 some experimental results are presented. Finally, a brief presentation of the *Pinocchio* environment is provided in section 7, and section 8 presents some final remarks.

2 Representation and rules

Every lexical element a in the input sentence w is abstractly represented by means of elementary objects, called *tokens*. A token T is associated with three structures:

- $[T]_{dep}$ is a dependency tree for a, i.e. a tree representing syntactic dependencies between a and other lexical elements (its dependants) in w.
- $[T]_{feat}$ is a feature structure representing syntactic and semantic information needed to combine a with other elements in the input.
- $[T]_{lf}$ is a Quasi Logical Form (QLF) providing a semantic interpretation for the combination of a with its dependants.

Rules operate on tokens, therefore they can access all the three structures above. Rules incrementally build and update such structures. Lexical, syntactic and semantic constraints can then be used in rules and processed uniformly.

The general form of a *rule* is a triple $\langle \gamma\alpha\delta, \Gamma_T, \Gamma_A \rangle$ where:

- $\gamma\alpha\delta$: is a non-empty string of tokens, called the rule *pattern*; α (the rule **core**) is non-empty, while γ, δ (the rule *context*) may be empty;
- Γ_T: is a set of boolean predicates (the rule *test*), defined over tokens in the rule pattern;
- Γ_A: is a set of elementary operations (the rule *action*), defined only over tokens in the rule core.

The postfix, unary operators '$*$' (Kleene star) and '?' (optionality operator) can be used in the rule patterns.

A basic data structure, called a *token chart*, is processed and dynamically maintained. This is a directed graph whose vertices are tokens and whose arcs represent binary relations from some (finite) basic set. Initially, the token chart is a chain-like graph with tokens ordered as the corresponding lexical elements in w, i.e. arcs initially represent lexical adjacency between tokens. During the processing, arcs may be rewritten so that the token chart becomes a more general kind of graph.

For a rule to apply, a path σ must be found in the token chart that, when viewed as a string of tokens, satisfies the two following conditions: (i) σ is matched by $\gamma\alpha\delta$; and (ii) all the boolean predicates in Γ_T hold when evaluated on σ.

When a rule applies, the elementary operations in Γ_A are executed on the tokens of σ matching the core of the rule. The effect of action execution is that $[T]_{dep}$, $[T]_{feat}$ and $[T]_{lf}$ are updated for the appropriate matching tokens. As explained in

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

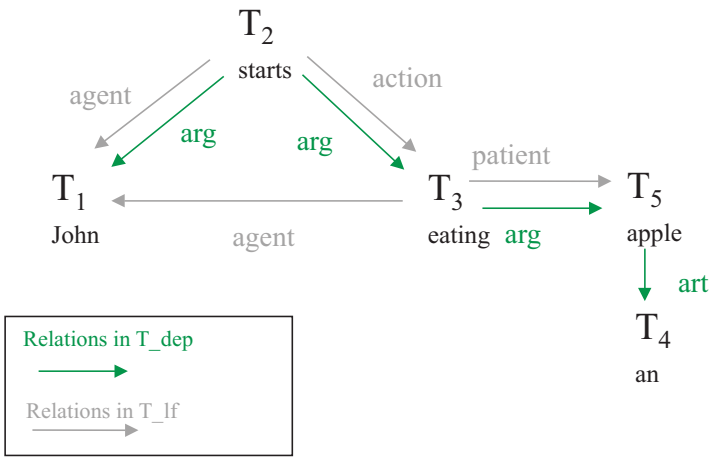


Fig. 1. Relations in $[T_n]_{dep}$ and $[T_n]_{lf}$ for the sentence "John starts eating an apple".

detail in Section 4 rules are grouped into cascades that are finite ordered sequences of rules.

In Figure 1 the token chart for "John starts eating an apple" is shown (only relations in $[T]_{dep}$ and $[T]_{lf}$ are displayed). A rule could easily test whether the "patient" of eating is a definite or an indefinite NP by starting from T_3, traversing the arc labeled "patient" (which is part of $[T_3]_{lf}$), reaching T_5 and then moving to T_4 by traversing the arc labeled "art" (in $[T_5]_{dep}$). $[T_4]_{feat}$ will then provide the information about the definiteness of the article (not shown).

3 Parsing

The parsing model is strongly influenced by IE needs. In fact, when a parser is inserted into a larger architecture, there is a tension between producing results useful with respect to application needs and results correct from a linguistic point of view. This is in line with what is claimed by Abney (1996): "in measuring parser accuracy, there is a tension between evaluation with respect to some application and evaluation with respect to a linguistic definition of 'phrase' ". Therefore, considering that for IE purposes a fully correct and specified parse tree for each sentence is not always necessary, we define the parser aim as that of building the *Sufficient IE Approximation* (SIEA) of the correct parse tree. A SIEA is a complete parse tree where all the relations relevant for template filling are represented, while other relations are underspecified or even left implicit. In other words, a SIEA is the minimal approximation of the complete parse tree providing all the relations useful for IE. For evaluation, the parser results are to be compared with such approximation of the parse tree. We assume that only one possible SIEA for each sentence exists.

Syntactic and semantic processing are interleaved. As soon as a syntactically motivated constituent is built, its semantic consistency is tested; if it succeeds, the actions associated with the rule are executed and $[T]_{dep}$, $[T]_{feat}$ and $[T]_{lf}$ (i.e. dependency tree, feature structure and Quasi Logical Form) are updated for the affected tokens. The final result of the parsing process is that the final parse tree

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

is available in $[T_{sent}]_{dep}$, together with information in $[T_{sent}]_{feat}$ and $[T_{sent}]_{lf}$ (where T_{sent} represents the semantic head of the sentence). The final parse tree obtained is very close to the SIEA mentioned above.

Parsing is based on the application of a fixed sequence of cascades of rules. It is performed in three steps using different grammars:

- chunking (analysis of NPs, VGs and PPs);
- clause recognition and nesting;
- modifier attachment.

The first two steps are syntax-driven and based on generic grammars. Most rules in such grammars are general syntactic rules, even if they crucially rely on the lexical and semantic information provided by a foreground lexicon (see section 3.1). During modifier attachment, mainly semantic patterns are used. At the end of these three steps the SIEA is available.

We use deterministic dependency parsing[1] operating on a specific linear path within the token chart (*the parsing path*); at the beginning the parsing path is equal to the initial token chart. When a rule is successfully applied, the parsing path is modified so that only the head token is visible for the application of the following rules; this means that the other involved elements are no longer available to the parser.

In the following we do not describe chunking because it is accomplished in a standard way and immediately move to the successive steps of processing. We just mention the fact that in our approach chunks are linguistically motivated (i.e. they are non recursive constituents such as NPs, PPs, Verbal Groups, etc.). For example in the sentence "the cat is chasing the grey mouse in the barn", the result of chunking will be: `[the cat]`$_{np}$ `[is chasing]`$_{vg}$ `[the grey mouse]`$_{np}$ `[in the barn]`$_{pp}$.

3.1 Clause recognition

The main task during clause recognition is what we call A-structure analysis. A-structure analysis is concerned with the recognition of argumental dependencies between both chunks and clauses. All kinds of modifier dependencies (e.g. PP attachments) are disregarded during this step.

More precisely: let w be the input sentence. A *dependency tree* for w is a tree representing all predicate-argument and predicate-modifier syntactic relations between the lexical elements in w. We define the *A-structure* for w as a tree forest equal to w's dependency tree but with all the nodes representing modifiers unattached from the element they modify. An A-structure is associated with the token that represents the semantic head of w (T_{sent} in the following). A-structure analysis finally associates to T_{sent}:

- $[T_{sent}]_{dep}$: the A-structure spanning the whole sentence;
- $[T_{sent}]_{feat}$: its feature structure;
- $[T_{sent}]_{lf}$: its Quasi Logical Form.

[1] Nevertheless, in the examples given in this paper we use the traditional constituency based categories (NP, PP, etc.).

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

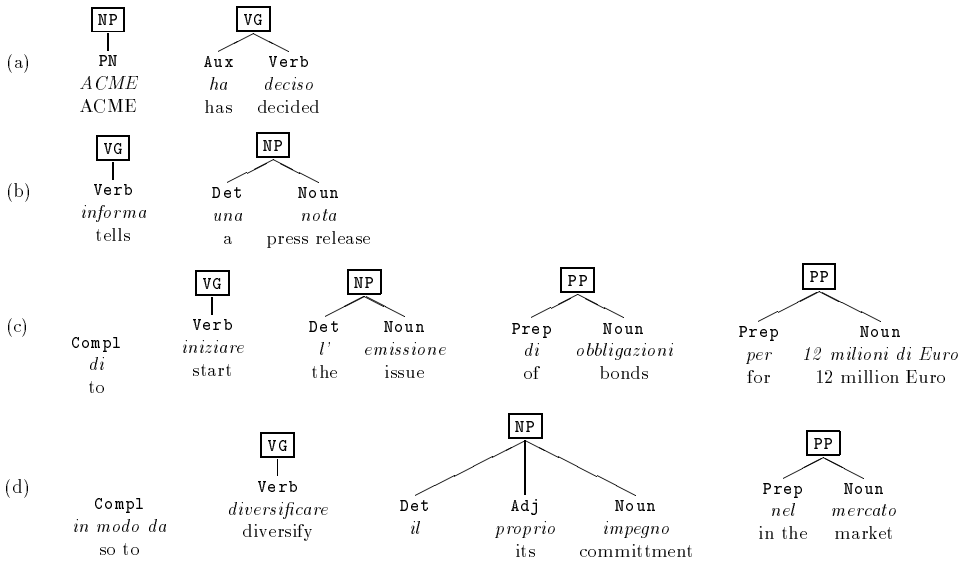


Fig. 2. Analysis of the sentence after chunking and clause boundary identification. Each block (e.g. that in line (a)) is a basic clause whose boundaries were recognized by rule sequence (i). Each tree in a block is a chunk.

A-structure analysis is performed without recursion by applying four sequences of rule cascades:[2]

(i) clause boundary recognition: delimiters such as complementizers, punctuation marks, etc. are used to identify clause boundaries;
(ii) analysis of basic (i.e. non-recursive) clauses. Subcategorization frames of heads of available chunks are used to connect chunks together. Three cascades of rules are involved: one for the subcategorization frames of heads of NPs and PPs, one for those of VGs, and one for combining complementizers with their clausal arguments.
(iii) analysis of dependencies between basic clauses. This sequence processes all sentential arguments and all incidentals, without any need for recursion (actually – as we will see – we apply a controlled form of recursion by iterating cascade applications). This sequence is applied three times, i.e. it recognizes structures with a maximum of three nested clauses.
(iv) recovery analysis. All tree fragments not yet connected together are merged.

Tokens not recognized as arguments at the end of A-structure analysis are marked as modifiers, and left unattached in the resulting A-structure. Note that in the figures illustrating the example below, unattached constituents are indicated by the presence of a $^{\mathtt{u}}$ superscript. These tokens will be attached in the parse tree during modifier attachment (see section 3.2).

[2] The order of the cascades in the sequences depends on the intrasentential structure of the specific language coped with.

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

Pattern	Test	Action	Matched Input
$\mathbf{T}_1$	$[T_1]_{feat}$.cat=NP	Dependant($[T_1]_{dep}$,$[T_3]_{dep}$) $[T_1]_{feat}$.subcat.int-arg=$[T_3]_{feat}$ $[T_1]_{lf}$.patient=$[T_3]_{lf}$.head	**"the issue"**
T_2*	$[T_2]_{feat}$.cat=Adjunct		
T_3	$[T_3]_{feat}$.cat=PP $[T_3]_{feat}$.marked=''of'' $[T_3]_{feat}$=$[T_1]_{feat}$.subcat.int-arg		**"of bonds"**

Fig. 3. The rule that recognizes [the issue of bonds]$_{np}$. It matches two tokens (T_1 and T_3), possibly interleaved by some optional modifiers (T_2). As action it inserts T_3 as (i) dependant of T_1 in the parse tree; (ii) internal argument of T_1 in the feature structure; (iii) "patient" of T_1 in the Quasi Logical Form.

In the following, we shall go through the analysis of the sentence below to show how the algorithm works:[3]

ACME ha deciso, informa una nota, di iniziare l'emissione di obbligazione per 12 milioni di Euro allo scopo di diversificare il proprio impegno sul mercato

(literal translation: ACME has decided, tells a press release, to start the issue of bonds for 12 million Euro so to diversify its commitment in the market)

In Figure 2 the analysis of the sentence is shown immediately after the application of the rules in sequence (i) above: blocks (a), (b), (c), and (d) represent the four clauses whose boundaries have been identified by the rules in such sequence. Trees in each line represent chunks.

We adopt a highly lexicalized approach. In a pure IE perspective the information for A-structure analysis is provided by a foreground lexicon (Kilgarriff 1997). Foreground lexica provide detailed information about words relevant for the domain (e.g. subcategorization frame, relation with the ontology). For words outside the foreground lexicon a large background lexicon provides generic information. The term subcategorization frame is used here in a restricted sense: it generally includes subject, object and – possibly – one indirect complement. The information in the foreground lexicon allows the classification of known tokens as arguments of other known tokens with high reliability.

Let's go back to our example. Basic clauses are analyzed by the rules in sequence (ii). The constituents are built in the following order:

- [the issue of bonds]$_{np}$: the rule in Figure 3 recognizes [of bonds]$_{pp}$ as internal argument of [the issue]$_{np}$; the rule uses the information associated with issue in the foreground lexicon, i.e. that it is the nominalization of "to issue". The subcategorization frame of such a verb specifies its arguments and

[3] In this paper, we mainly use literal English translations of Italian examples.

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

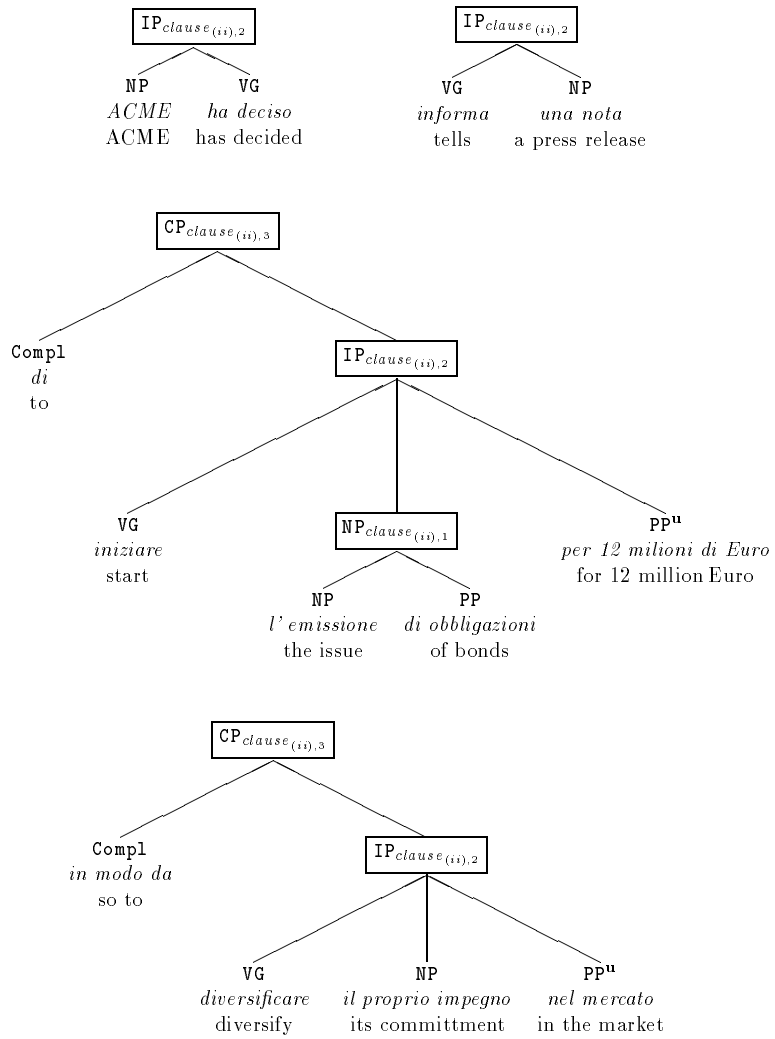


Fig. 4. Analysis of the sentence after basic clause recognition (rule sequence (ii)). Constituents marked with `u` are unattached and positioned in a default position (mainly constrained by other surrounding constituents) and will be attached during the modifier attachment step, as described in section 3.2.

their semantic restrictions. The syntactic rule adds the condition that – being a nominalization – the internal argument is realized as a PP marked by `of`;

- `[ACME has decided]`$_{ip}$: `[ACME]`$_{np}$ is the external argument of `[has decided]`$_{vg}$;
- `[tells a press release]`$_{ip}$: `[a press release]`$_{np}$ is the external argument of `[tells]`$_{vg}$;
- `[start the issue of bonds for 12 million Euro]`$_{ip}$: `[issue]`$_{np}$ is the internal argument of `[start]`$_{vg}$; `[for 12 million Euro]`$_{pp}$ is a modifier (as it is not subcategorized by other tokens) and is unattached in the A-structure;
- `[diversify its obligation in the market]`$_{ip}$: `[its obligation]`$_{np}$ is the internal argument of `[diversify]`$_{vg}$; `[in the market]`$_{pp}$ is a modifier;

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

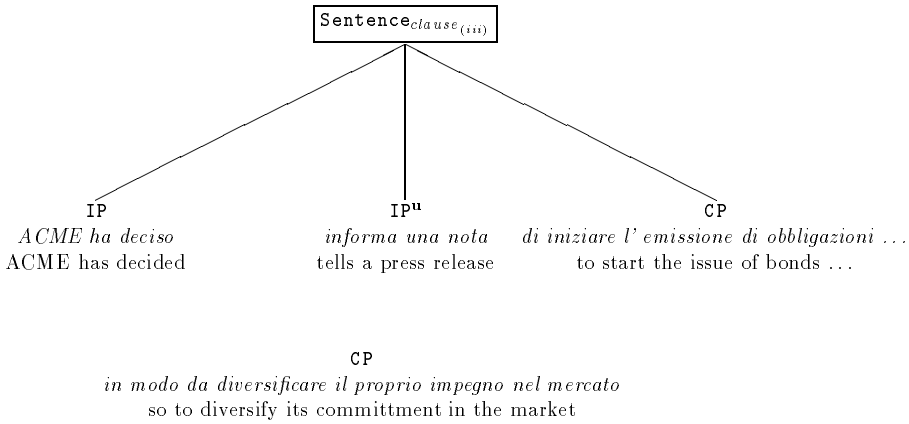


Fig. 5. Analysis of the sentence after the analysis of dependencies between basic chunks (rule sequence (iii)).

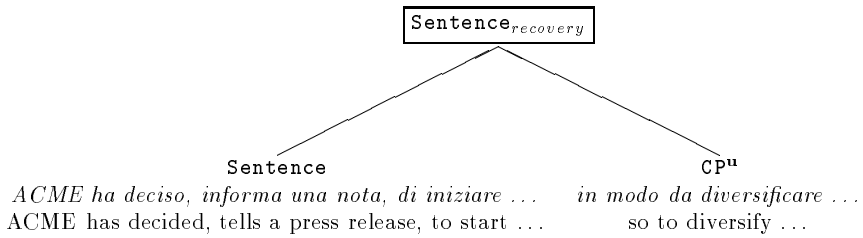


Fig. 6. Analysis of the sentence at the end of A-structure recognition.

- `[to start the issue of bonds for 12 million Euro]`$_{cp}$: `[to]`$_{compl}$ gets its argument `[start ...]`$_{ip}$;
- `[so to diversify its obligation in the market]`$_{cp}$: `[so to]`$_{compl}$ gets its argument `[diversify ...]`$_{ip}$.

The result of the application of basic clause recognition at the $[T]_{dep}$ level is shown in Figure 4.

Rule sequence (iii) recognizes dependencies among clauses, e.g. that `[ACME has decided]`$_{ip}$ still needs a sentential argument, i.e. a subordinate clause introduced by `to` (such information comes from the foreground lexicon). The argument is found after the incidental `[tells a press release]`: it is the `CP` headed by `[to start]`. The result of this phase is shown in Figure 5.

Finally, the recovery sequence (iv) collapses the two constituents above into a single sentence structure; the `CP` is considered a clausal modifier (as it is not subcategorized by any element) and – being a modifier – is left unattached in the A-structure. The result is shown in Figure 6.

At the end of the clause recognition step T_{sent} (i.e. the token associated with the semantic head of the sentence) is the token associated with the head of `[has decided]`.

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

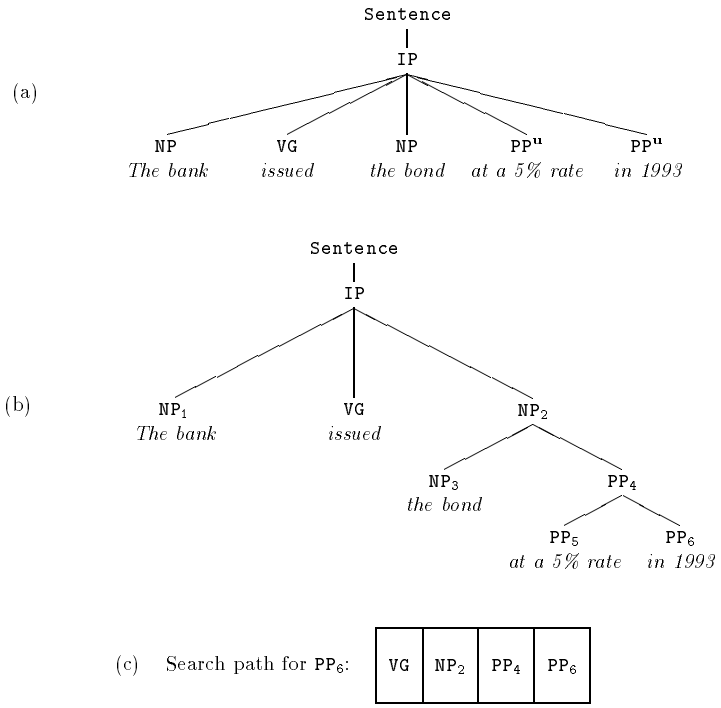


Fig. 7. (a) Result produced by clause recognition. (b) Tree with the unattached modifiers in default positions. (c) Search path for PP_6. Note that the parser produces dependency trees (that are shown as constituency trees for explanatory purposes); so, for example, attaching a modifier to NP_2 actually means attaching it to NP_3.

3.2 Modifier attachment

The result of the clause recognition process is a parse tree with the modifiers not yet attached in $[T_{sent}]_{dep}$ together with the corresponding feature structure in $[T_{sent}]_{feat}$ and the Quasi Logical Form in $[T_{sent}]_{lf}$. The aim of the step described in this section is to find the correct position for attaching relevant modifiers in the parse tree and to add the proper semantic relations between each modifier and its modifiee in $[T_{sent}]_{lf}$.

$[T_{sent}]_{dep}$ and $[T_{sent}]_{lf}$ are used to determine the correct attachments and are also modified during this step. Modifier attachment is performed in two steps: first all the possible attachments are computed for each modifier (its *search path*, SP). An example of a search path is shown in Figure 7. During this step mainly generic syntactic rules are used. Then the correct attachment in the search path is determined for each modifier, applying domain-specific rules. Rules always modify both $[T_{sent}]_{dep}$ and $[T_{sent}]_{lf}$. Modifiers relevant for the IE task are attached in the proper position in $[T_{sent}]_{dep}$ and a specific semantic relation is established in $[T_{sent}]_{lf}$ with the modified token. Other modifiers are attached in a default position in $[T_{sent}]_{dep}$ and a default generic relation is established in $[T_{sent}]_{lf}$.

Initially, modifiers are attached in the lowest position in $[T_{sent}]_{dep}$ and a generic

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

Pattern	Test	Action	Matched Input
$\mathbf{T}_1$	$[T_1]_{lf}$.head=TO-INCREASE	Dependant($[T_1]_{dep}$,$[T_5]_{dep}$) $[T_1]_{lf}$.increased-by=$[T_5]_{lf}$.head	**"an increase"**
T_2*	$[T_2]_{feat}$.cat=Adjunct		
T_3	$[T_3]_{lf}$.head=PROFIT $[T_3]_{feat}$.cat=PP $[T_3]_{feat}$.marked=''of''		**"of profits"**
T_4*	$[T_4]_{feat}$.cat=Adjunct		
T_5	$[T_5]_{lf}$.head=PERCENTAGE $[T_5]_{feat}$.cat=PP $[T_5]_{feat}$.marked=''of/by''		**"of/by 20%"**

Fig. 8. An example of modifier attachment rule.

semantic relation is hypothesized between each modifier and the rest of the sentence in $[T_{sent}]_{lf}$. Given:

- T_n: a modifier token derived from chunk n,
- T_{n-1}: the token, derived from chunk $n-1$, immediately preceding n in the sentence,

in right branching languages (such as Italian) the lowest possible attachment for T_n is in the position of modifier of T_{n-1}.[4]

Afterwards, the possible SP for each modifier is computed. The SP for a modifier T_n is a linear path in the token chart connecting T_n with other elements in $[T_{sent}]_{dep}$ that – from a syntactic point of view – can be modified by T_n (see Figure 7). The initial SP for T_n is given by the path in $[T_{sent}]_{dep}$ connecting T_{n-1} with T_{sent}. Then rules are applied to filter out elements in SPs according to syntactic constraints (e.g. NPs or PPs can be modified by a relative clause, but VGs cannot).

After SPs have been computed, modifiers are attached using a sequence of cascades of rules.

A first cascade, mainly composed of generic syntactic rules, attaches subordinates (e.g. relative clauses). Many of these rules are somehow similar to A-structure recognition rules. They are truly syntactic rules recognizing part of the subcategorization frame of subordinated verbs, using semantic information provided by the foreground lexicon. Note however that they are applied to SPs and not on the parsing path (as A-structure rules are).

Other cascades are used to attach different types of modifiers, such as PPs. Such

[4] The only exception to this rule is that if T_n is the first token in a clause it modifies the head of the clause; e.g. in "in 1993 ACME decided to give up" "in 1993" modifies "decided". For the sake of explanation we ignore such a case.

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

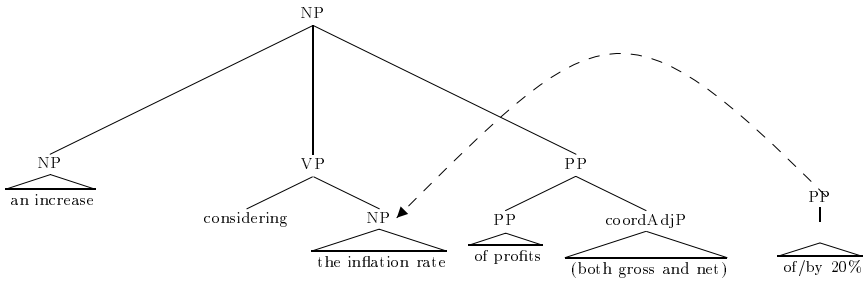


Fig. 9. A modifier attachment that violates syntactic constraints.

rules mainly involve semantic constraints. For example, the rule shown in Figure 8 can recognize `una crescita dei profitti del 20%` (lit. an increase of profits of/by 20%).[5]

The SP-based rule application simplifies rule writing. Patterns are usually developed having in mind a simple case, i.e. a sequence of contiguous chunks in the sentence that can be interleaved by other non relevant chunks. For example, the rule in Figure 8 has been developed with `[an increase] [of profits] [of/by 20%]` in mind, but it also applies to:

- `an increase in 1997 of profits of/by 20%`
- `an increase, newspapers report, of profits of/by 20%`
- `an increase, considering the inflation rate, of profits (both gross and net) of/by 20%`

Conceptually this step is very similar to the one used by shallow parsing approaches (e.g. Grishman 1997). Note, however, that rules are not applied on a list of contiguous chunks, but on the search path (where the parse tree and related syntactic relations are available). Parse-tree based modifier attachment is less error prone than attachment performed on flat chunk structures (as used in shallow parsing). For example, it is possible to avoid cases of attachments violating syntactic constraints, as it would be the case in attaching (in the third example above) `profit` to `increase` and `20%` to `inflation rate` (see Figure 9).

3.3 At the end of parsing

At the end of the modifier attachment step the final parse tree is available in $[T_{sent}]_{dep}$, together with information in $[T_{sent}]_{feat}$ and $[T_{sent}]_{lf}$. The obtained final parse tree is very close to the SIEA mentioned at the beginning of section 3. In this tree all the A-structures are built and all the modifiers are attached.[6] Modifiers relevant for the

[5] Generally rules involve only two elements (i.e. the modifier and the modifiee), taking into account the presence of non relevant tokens (e.g. some adjuncts) intervening in the SP. The rule in Figure 8, instead, involves three tokens with intervening adjuncts, because it copes with a recurring ambiguity in some financial domains: T_5 could modify both T_1 and T_3 from a syntactic and semantic point of view. This rule attaches T_5 to T_1.

[6] For an evaluation of the accuracy of A-structures recognition, see section 6.

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

IE application are attached in the correct position in the tree and a valid semantic relation is established with the modifiee in $[T_{sent}]_{lf}$. Irrelevant modifiers are attached in a default position in the tree (the lowest possible attachment) and a null semantic relation is established with the modifiee. The only difference between the produced tree and the SIEA is in the A-structure, where all the argumental relations (e.g. verb arguments) are captured (and not only those relevant for the domain). Modeling also the argumental structures of irrelevant constituents can be useful to correctly assign salience at discourse level. For example, when interesting relations involve elements that are dependants of irrelevant verbs (see section 5).

4 Grammars and rule organization

At all levels of analysis (chunking, clause recognition, modifier attachment, etc.) rules are grouped into *cascades* that are finite, ordered sequences of rules. Cascades represent elementary logical units, in the sense that all the rules in a cascade deal with some specific construction (e.g. subcategorization of verbs). From a functional point of view a cascade is composed of three segments:

- **s1** contains rules that deal with idiosyncratic cases for the construction at hand;
- **s2** contains rules dealing with the regular cases;
- **s3** contains default rules that fire only when no other rule can be successfully applied.

The initial generic grammars for chunking and clause recognition are designed to cover the most frequent phenomena in a restrictive sense. Additional rules can be added to the grammar (when necessary) for coping with the uncovered phenomena, especially domain-specific idiosyncratic forms. The limited size of the grammar makes modifications simple (e.g. our clause recognition grammar for Italian is composed of 66 rules).

The deterministic approach combined with the use of cascades and segments makes grammar modifications simple, as changes in a cascade (e.g. rule addition/modification) influence only the following part of the cascade or the following cascades. This makes the writing and debugging of grammars easier than in approaches based on recursive (e.g., context-free) grammars, where changes to a rule can in principle influence the application of any rule in the grammar.

The grammar organization in cascades and segments allows a clean definition of the grammar parts. The grammar is organized in a very modular way. Each cascade copes with a specific phenomenon. All the rules for the specific phenomenon are grouped together and are easy to check.

The segment structure of cascades is suitable for coping with the idiosyncratic phenomena of restricted corpora. As a matter of fact domain-oriented corpora can differ from the standard use of language (such as that found in generic corpora) in two ways:

- in the frequency of the constructions for a specific phenomenon;
- in presenting different (idiosyncratic) constructions.

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

Coping with different frequency distributions is conceptually easy by using deterministic parsing and cascades of rules, as it is just necessary to change the rule order within the cascade coping with the specific phenomenon, so that more frequently applied rules are first in the cascade. Coping with idiosyncratic constructions requires the addition of new rules. Adding new rules in highly modularized small grammars is not complex.

Finally, from the point of view of grammar organization, defining segments brings some added value to ordered cascades. Generic rules (in s2) are separated from domain specific ones (in s1); rules covering standard situations (in s2) are separated from recovery rules (in s3). In s2, rules are generic and deal with unmarked cases. In principle s2 and s3 are units portable across the applications without changes. Domain-dependent rules are grouped together in s1 and are the resources the application developer works on for adapting the grammar to the specific corpus needs (e.g. coping with idiosyncratic cases). Such rules generally use contexts and/or introduce domain-dependent (semantic) constraints in order to limit their application to well defined cases. S1 rules are applied before the standard rules and so idiosyncratic constructions have precedence with respect to standard forms.

Segments also help in parsing robustly. S3 deals with unexpected situations, i.e. cases that could prevent the parser from continuing. For example, the presence of unknown words is coped with after chunking by a cascade trying to guess the word's lexical class. If every strategy fails, a recovery rule includes the unknown word in the immediately preceding chunk so to let the parser continue. Recovery rules are applied only when rules in s1 and s2 do not fire.

5 Discussion and related research

The proposed approach has two reference areas: parsing and IE. Concerning parsing, recently there has been an increasing interest in finite-state techniques for NLP (Ejerhed, Jelinek, Karttunen and Kornai 1996; Roche and Schabes 1997; Kornai 1999). Deterministic parsers based on finite-state cascades have been proposed as suitable for analyzing unrestricted text: such parsers are efficient and often as accurate as those based on more complex approaches. A reference work in this area is that by Abney (1996), where parsing of free texts proceeds deterministically by growing islands of certainty into larger and larger phrases. Abney identifies two main levels:

- chunk identification and recognition, and
- clause identification and recognition.

Chunks never contain same-level or higher-level chunks. Modifier attachment is performed only if non ambiguous (but the possible attachment positions are limited by the containing clause). Embedded clauses are treated as siblings, i.e. no attachment is performed and there is no commitment on which is the main clause. Recognized phrases are not rejected because of unification failures: a chunk composed of a masculine determiner and a neutral noun is accepted. Robustness is obtained by favoring local decisions (e.g. attachments) with respect to global optimization, i.e. no closed-world assumption is adopted with respect to the grammar. Precision is obtained by

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

keeping pattern precision high, so that it is possible to parse deterministically with acceptable error rates.

The approach presented in this paper derives from Abney's work the multi-level approach and the idea of organizing parsing as a 'growing islands of certainty' process. Our approach is more standard in the sense that we adopt a full unification grammar style where non consistent chunks are rejected and our islands have a strong constituency-based nature. We perform relevant modifier attachments by exploiting lexical and semantic information (in $[T_{sent}]_{feat}$ and $[T_{sent}]_{lf}$). Such information is made available by the foreground lexicon of the IE system within which the parser is used. The search space for modifiers is explicitly computed, not just constrained by clause boundaries as in Abney's work. Chunks may contain same-level chunks, as iteration of cascades is performed in order to obtain bounded recursion (e.g. clause embedding is computed). Finally, we have considered the issue of grammar organization and modifications.

As for Information Extraction, full parsing is not a very popular approach. As a consequence of the poor performance obtained by systems based on full parsing at Message Understanding Conference evaluations (see, for example, MUC7 1998)), most of the current systems are based on shallow approaches. As a matter of fact standard full parsing approaches tend to be slow (because of huge search spaces) and brittle (because of gaps in the grammar). In general, standard full parsing approaches are not suitable for Information Extraction because they are not able to adapt the depth of analysis to the parts of the input that are of interest for the IE task at hand. For example most of processing time of a chart parser is due to the ambiguity in attaching modifiers (e.g. PP attachments). Note that the attachment point of many of these modifiers is irrelevant from an IE point of view when they carry information irrelevant for the IE process. Moreover, full parsing approaches generally adopt a closed world assumption about the grammar, i.e. they tend to hypothesize a grammar completeness with respect to the language. For this reason, they often produce spurious results in the attempt of pursuing a global analysis for sentences where gaps in coverage are present (Grishman 1997).

Our approach aims to provide the same output a full parsing approach provides, i.e. a complete parse tree for each sentence. The difference between the SIEA and a complete parse tree is that a SIEA is a parse tree where only the relations relevant for template filling are represented, while the other relations are underspecified or even left implicit. Our approach avoids the brittleness of the full parsing approaches because it does not adopt a closed world assumption as for the grammar (i.e. it can hypothesize the presence of gaps and deal with them) and therefore it applies recovery strategies for unexpected situations that allow gaps to be effectively overcome. Many full parsing approaches tend to provide partial solutions when confronted with gaps in the grammars in the form of a number of unrelated parse tree fragments, i.e. a gap in the grammar stops the analysis (Humphreys, Gaizauskas, Azzam, Huyck and others 1998). In our approach a gap in the grammar causes a recovery action to connect the fragments in a default way (e.g. by attaching the second fragment to the first one in the lowest possible position) and the analysis continues. In this way parts of sentences that are around the portion involved in the gap can still be correlated.

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

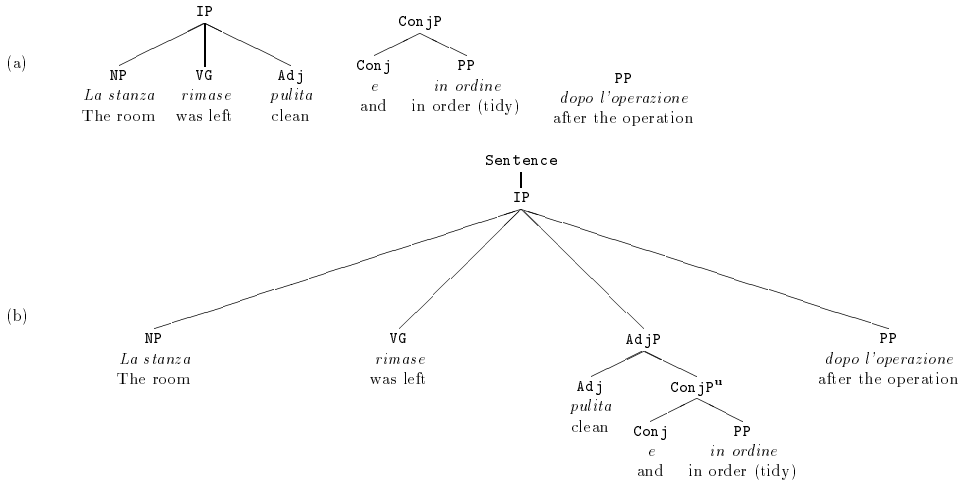


Fig. 10. (a) Fragmented analysis produced by a full parsing approach when a gap is present (the grammar is not able to cope with a coordination between an adjective and a PP). (b) Analysis produced by our approach.

In Figure 10(a) the fragmented result produced by a full parsing approach is shown; in Figure 10(b) what is produced by our approach is shown.

In our approach the parser spends little time performing irrelevant attachments, as modifier attachment is semantics-driven. Only attachments relevant for the domain at hand are performed. The only step performed for irrelevant modifiers is the computation of the search path, a rapid operation of graph traversing. Concerning gaps in the grammar, our system adopts a conservative approach, building constituents only when there is a strong evidence also from the domain point of view. A mixing of bottom-up (data-driven, mainly in chunking and in some parts of A-structure recognition) and top-down strategy (expectation-driven, mainly in A-structure and modifier attachment) allows parsing to be general, effective and adaptable to the domain needs. As mentioned, grammar changes (e.g. insertion of domain-specific rules) are simplified by the grammar organization. More importantly the SIEA produced by our parser is equivalent from an IE point of view to the complete parse tree (hopefully) produced by more complex approaches. So in principle the final system's effectiveness in extracting information from text does not change even if adopting a simpler approach (see the discussion about effectiveness in section 3).

Approaches to IE that make a limited use of syntactic information (Appelt *et al.* 1993; Grishman 1997; Aone *et al.* 1998) tend to be imprecise because of the lack of an explicit representation of syntactic relations among elements in the sentence (i.e. grammatical functions) and this fact has consequences also on the correct assignment of thematic roles. Meta-rules are able to represent syntactic variance, but they are applied to known patterns only. For this reason, syntactic structures (and generalizations) (even partially) involving irrelevant elements are lost. The lack of an explicit representation of relations very often implies the use of default strategies based on the linear order of chunks at many levels in the IE process. From

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

one side applying modifier attachment rules to the flat chunk sequence may lead either to violation of crossing constraints, or to wrong attachments if non-recognized nested structures (e.g. subordinated clauses) are present. From the other side, chunk linear order is used to compute precedence identification for coreference resolution at discourse level. This is a simplistic strategy that provides a sufficient approximation of the correct behavior for languages with a relatively rigid phrase order (e.g. English) (Kameyama 1997), but may be not adequate for other languages, such as Italian. In our approach generic grammars are used to cope with generic linguistic phenomena at sentence level. Syntactic relations (i.e. grammatical functions) in the sentence are represented and a better treatment of linguistic phenomena is possible. Nested structures are recognized at clause level and crossing constraint violations are avoided both at clause level and during modifier attachment due to the presence of an explicit global parse tree. The syntactic information in $[T_{sent}]_{dep}$ and $[T_{sent}]_{feat}$ is also useful in the steps following parsing because, for example, the availability of syntactic relations increases the accuracy in determining discourse relations. As a matter of fact at discourse level it is possible to adopt strategies to compute salience for coreference resolution that take into account both the syntactic relations among constituents and the argumental structure of the sentence (Not *et al.* 1999).

6 Evaluation

The proposed approach has been tested mainly for Italian, but demonstrators based on such an approach have been successfully developed also for English and Russian.

Given that the approach proposed in this paper is strongly influenced by IE needs, it is difficult to evaluate it by means of the standard tools used in the parsing community. Moreover, the language on which most effort was expended is Italian and for Italian there are no suitably annotated corpora for evaluation tasks. Approximate indications can be provided by the effectiveness in recognizing A-structures and by the results on the overall IE tasks.

Effectiveness in recognizing A-structures was experimentally verified for Italian on a corpus of 95 manually tagged texts in the domain of bond issue: 33 texts were used for training, 62 for test (10,472 words in all). In the experiments we tested the correctness of each established predicate-argument relation. For example in the sentence "the company announced a new offer" we tested if "the company" was assigned the role of subject of "announced". Separately we tested whether "offer" was assigned the role of object of the same verb. Results were: P=97, R=83 on the training corpus, P=95, R=71 on the test corpus. In our opinion, the high precision obtained in the tests demonstrates the applicability of the method. The lower recall shows difficulties of building complete foreground lexica by using a limited corpus, a well known fact in IE.

Effectiveness in modifier attachment was not directly tested. It is possible to derive indirectly its reliability by measuring the effectiveness of the whole IE process. Applications and demonstrators in different languages have been developed using *Pinocchio*, mainly in the financial domain (Ciravegna *et al.* 1999; Ciravegna *et al.* 2000b). For the Italian language, one application about bond issues has been fully

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

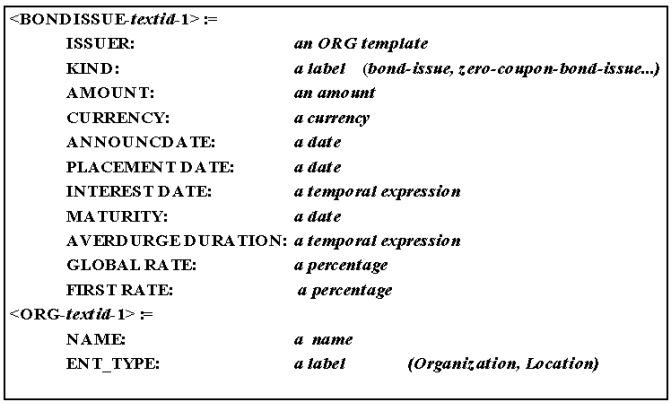


Fig. 11. The template to be filled for bond issues.

developed and two others have reached the level of demonstration (management succession and company financial results). Demonstrators were developed for English (economic indicators) and partially for Russian (bond issues). Concerning the Italian application for bond issues, the system reached P=80, R=72, F(1)=76 on 95 texts used for development (33 ANSA agency news, 20 "Il Sole 24 ore" newspaper articles, 42 Radiocor agency news; 10,472 words in all (Ciravegna *et al.* 1999; Ciravegna *et al.* 2000b)). Figure 11 shows the adopted template. Effectiveness was automatically calculated by comparing the system results against user-provided results via the MUC scorer. The development cycle of the template application was organized as follows: resources were developed by inspecting the first 33 texts of the corpus. Then the system was compared against the whole corpus (95 texts), reaching R=51, P=74, F(1)=60. Finally resources were tuned on the whole corpus mainly by focusing on the texts that did not reach sufficient results in terms of R&P. The system analyzed 1125 word/minute on a Sparc Ultra 5, 128M RAM. The Italian Named Entity Recognizer was also tested as a stand alone module on generic corpora in the financial domain. It reached P=86 R=92 in a blind test on a 5700 word corpus after training on a 9,200 word corpus (corpora manually tagged by us).

7 *Pinocchio*

The approach proposed in this paper for parsing has been extended to the whole architecture of the IE system *Pinocchio*. Also, lexical (lexical normalization and preparsing), semantic (default reasoning and template filling) and discourse modules are based on the same approach. *Pinocchio* (Ciravegna *et al.* 2000a) is an environment that provides a uniform and powerful set of tools for IE applications. Its design aims to supply a single environment taking into account both the current trends in IE (evaluation methodology, reduced porting time) and other aspects relevant for industrial applications (reduced cost, usability, integrability and multilinguality).

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

New applications can be developed and new languages can be covered by only modifying declarative resources. The declarative resources used by all the modules in the system employ the same formalism as the one presented in section 2: all the modules operate on the same input structure and use the same type of rules (processed by the same basic machinery). The environment for developing and debugging resources is uniform across the whole architecture. This uniformity simplifies the application development process and reduces the need of manpower and the required skills, i.e. the cost of the application. A single person with a generic knowledge in Computational Linguistics can eventually do the entire work.

Pinocchio is available in two configurations: the development environment and the delivery configuration. The development environment provides a set of tools that support application development (e.g. editors, compilers, debugger, tracers). At the end of the development process, a delivery configuration is produced that can be inserted as a black-box into the user environment through an API. *Pinocchio* provides an open architecture. In case of applications with specific requirements, it is possible either to define and integrate new modules (using the internal processors) or to include pre-existing modules (e.g. commercial technologies).

Domain-dependent and language-dependent resources are sharply separated. For this reason, when building new applications in the same language, most of the language-dependent resources can be reused. Conversely, when porting the same application from one language to another, most domain-dependent resources may be reused.

The system output is compatible with the current IE evaluation methodologies, in particular with the MUC standards and tools (Douthat 1998).

8 Final remarks

In this paper, we have presented an approach to full parsing approximation for IE purposes. The proposed methodology allows a parse tree to be produced for each sentence, a parse tree that – from an IE point of view – is equivalent to a complete parse tree. We have also discussed how a grammar for such parser is to be organized. The approach to parsing is reasonably simple in principle. Experience has shown that it is also easy to implement. A grammar organized in the proposed way is easier to modify and maintain than a large generic grammar.

The approach has been developed as part of LE-FACILE, a successfully completed EU project for multilingual text classification and IE (Ciravegna *et al.* 1999; Ciravegna *et al.* 2000b). The approach is currently used in *Pinocchio*, an environment for developing and running IE applications (Ciravegna *et al.* 2000a). The approach to rule writing and organization proposed in this paper (i.e. the type of rules, cascades, segments, and available primitives and rule interpreter) has been applied to all the modules of an IE system: also lexical (lexical normalization and preparsing), semantic (default reasoning and template filling) and discourse modules are organized in the same way (Ciravegna *et al.* 2000a).

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

Acknowledgements

The work presented in this paper was carried out while the first author was at ITC-irst. Giorgio Satta has contributed to the whole work on parsing for IE via cascades of rules. The authors would like to thank him for his constant help and the fruitful discussions. Thanks to Mark Hepple (University of Sheffield) for revising this paper. Errors, if any, are ours, of course. The FACILE project (LE 2440) was partially funded by the European Union in the framework of the Language Engineering Sector. The English demonstrator was developed by Bill Black, Fabio Rinaldi and David Mowatt (Umist, Manchester) as part of the FACILE project. The Russian demonstrator was developed by Nikolai Grigoriev (Russian Academy of Sciences, Moscow).

References

Abney, S. (1996) Partial parsing via finite-state cascades. *Proceedings ESSLI '96 Robust Parsing Workshop*. (Also available at `http://www.sfs.nphil.uni-tuebingen.de/~abney/Papers.html`.)

Aone, C., Halverson, L., Hampton, T. and Ramos-Santacruz, M. (1998) SRA: description of the IE^2 system used for MUC-7. *Proceedings 7th Message Understanding Conference (MUC-7)*.

Appelt, D. E., Hobbs, J. R., Bear, J., Israel, D. and Tyson, M. (1993) FASTUS: A finite-state processor for information extraction from real-world text. *Proceedings 13th International Joint Conference on Artificial Intelligence*.

Ciravegna, F., Lavelli, A., Mana, N., Gilardoni, L., Mazza, S., Ferraro, M., Matiasek, J., Black, W. J., Rinaldi, F. and Mowatt, D. (1999) FACILE: Classifying texts integrating pattern matching and information extraction. *Proceedings 16th International Joint Conference on Artificial Intelligence*.

Ciravegna, F., Lavelli, A. and Satta, G. (2000a) Bringing information extraction out of the labs: the *Pinocchio* environment. *Proceedings 14th European Conference on Artificial Intelligence*.

Ciravegna, F., Gilardoni, L., Lavelli, A., Mazza, S., Black, W. J., Ferraro, M., Mana, N., Matiasek, J. and Rinaldi, F. (2000b) Flexible text classification for financial applications. The FACILE system. *Proceedings 14th European Conference on Artificial Intelligence*.

Douthat, A. (1998) The Message Understanding Conference scoring software user's manual. *Proceedings 7th Message Understanding Conference (MUC-7)*.

Ejerhed, E., Jelinek, F., Karttunen, L. and Kornai, A. (eds.) (1996) *ECAI '96 Workshop on Extended Finite State Models of Language*.

Grishman, R. (1995) The NYU system for MUC-6 or where's syntax? *Proceedings 6th Message Understanding Conference MUC-6*. San Francisco, California: Morgan Kaufmann.

Grishman, R. (1997) Information extraction: techniques and challenges. In: Pazienza, M. T., editors, *Information Extraction: A multidisciplinary approach to an emerging technology*. Heidelberg, Germany: Springer-Verlag.

Hobbs, J. R., Appelt, D. E. and Tyson, M. (1992) Robust processing of real-world natural-language texts. *Proceedings 2nd Conference on Applied Natural Language Processing*.

Humphreys, K., Gaizauskas, R., Azzam, S., Huyck, C., Mitchell, B., Cunningham, H. and Wilks, Y. (1998) University of Sheffield: Description of the LaSIE-II System as Used for MUC-7. *Proceedings 7th Message Understanding Conference (MUC-7)*.

Kameyama, M. (1997) Recognizing referential links: an information extraction perspective. In: Mitkov and Boguraev, editors, *Proceedings ACL/EACL Workshop on Operational Factors in Practical, Robust Anaphora Resolution for Unrestricted Texts*.

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press

Kilgarriff, A. (1997) Foreground and background lexicons and word sense disambiguation for information extraction. *Proceedings Workshop on Lexically Driven Information Extraction.*

Kornai, A. (ed.) (1999) *Extended Finite State Models of Language*. Cambridge: Cambridge University Press.

MUC7 (1998) *Proceedings 7th Message Understanding Conference (MUC-7)*. `http://www.itl.nist.gov/iaui/894.02/related_projects/muc/index.html`: SAIC.

Not, E., Tovena, L. and Zancanaro, M. (1999) Positing and resolving anaphora in deverbal NPs. *Proceedings ACL Workshop on "The Relationship Between Discourse/Dialogue Structure and Reference"*.

Roche, E. and Schabes, Y. (eds.) (1997) *Finite-State Language Processing*. Cambridge, MA: MIT Press.

https://doi.org/10.1017/S1351324902002917 Published online by Cambridge University Press