Towards a flexible syntax/semantics interface

Luísa Coheur[1], Fernando Batista[2], and Nuno J. Mamede[3]
Spoken Language Systems Lab
Rua Alves Redol 9, 1000-029 Lisboa, Portugal

[1] L^{2}F – INESC-ID Lisboa/IST/GRIL,
Luisa.Coheur@inesc-id.pt,
[2] L^{2}F – INESC-ID Lisboa/ISCTE,
Fernando.Batista@inesc-id.pt
[3] L^{2}F – INESC-ID Lisboa/IST,
Nuno.Mamede@inesc-id.pt

Abstract. We present a syntax/semantics interface that was developed having in mind a set of problems identified in system Edite, which wasbased on a traditional syntax/semantics interface. In our syntax/semantics interface, syntactic and semantic rules are independent, semantic rules are hierarchically organized, and partial analysis can be produced.
Resumo. Apresentamos uma interface sintaxe/semântica que foi desenvolvida tendo em mente um conjunto de problemas identificados no sistema Edite, uma interface sintaxe/semântica tradicional. Na nossa interface sintaxe/semântica as regras sintácticas e semânticas são independentes, as regras semânticas estão hierarquicamente organizadas, e podem ser produzidos resultados parciais.
Keywords. Syntax/semantics interface, partial results, semantic rules hierarchically organized.

1 Introduction

Edite [13, 16] was a conventional natural language interface, developed in the late 90s, in order to be integrated in a multimedia kiosk and answer questions about tourist resources. In order to obtain logical forms, Edite used a syntax/semantics interface, like the ones presented in [4] and [15]. Typically, in these largely spread syntax/semantics interfaces:

– each semantic rule is associated with a syntactic rule;
– formulas are recursively obtained: the semantics of a constituent is obtained by combining the semantics of its sub-parts;
– the semantic analysis depends on a complete syntactic analysis;
– the production of formulas depends on a totally successful semantic analysis.

Edite had a fast implementation – in 6 months it was aswering simple questions – but it soon became difficult to extend, mainly due to problems regarding the inefficient syntax/semantics interface. The strong dependency between syntactic and semantic rules, the recursive organization of rules, and the need of complete analyses were the source of many problems, including but not limited to the following:

– a new syntactic rule could force semantic rules to run with unexpected semantic values;
– the desired semantic values could be difficult to obtain, as significant syntactic parts could be spread across different syntactic rules;
– a new rule (either syntactic or semantic) could cause dramatic over-generation;
– it was difficult to identify rules causing over-generation or errors;
– if the syntactic or (semantic analysis) was incomplete, no partial result could be returned.

In this paper we propose an alternative syntax/semantic interface, that has been developed having the above problems in mind, and we describe the tools and data structures it uses. The paper is organized as follows: section 2 describes the interface; section 3 presents a few case studies; finally, section 4 presents conclusions and future work.

2 The system description

The architecture of the syntax/semantic interface is presented on Figure 1 and described in the following sections.

The first processing step consists of performing a syntactic surface analysis over a morpho-syntactically tagged text. The chunks in the set obtained by this process are then connected, as well as the elements inside them. The resulting structure is then transformed into graphs, where words are connected by dependencies. From these graphs, logical forms can be generated and converted into SQL queries, allowing for database access.

2.1 The surface analysis

Susana [5] is a syntactic analyser that, using a surface grammar, splits sentences into chunks [2]. The linguistic information, required for the analysis, consists of a set of rules, named blocks [14], for defining the behavior of a given linguistic structure inside another structure. Besides blocks, preferences may also be defined in order to refine the results.

The analysis process is performed into different stages: the algorithm finds all possible sequences during the first stage, in order to cover unusual linguistic constructions; then selects the most promising ones, either according to linguistic preferences or by applying psycholinguistic principles, such as longest model principle [1, 14].

Linguistic preferences are restriction mechanisms, for choosing alternatives from a set of possibilities, based on some confidence level. Each preference establishes a confidence value for selecting between two different models, based on several restrictions, and may be applied whenever two solutions exist, having the same conditions. A confidence level associated with each preference, allows the introduction of probabilistic elements in the grammar.

Besides preferences, SuSAna makes use of psycholinguistic principles [15, 4], for choosing between different interpretations that the parser might be able to

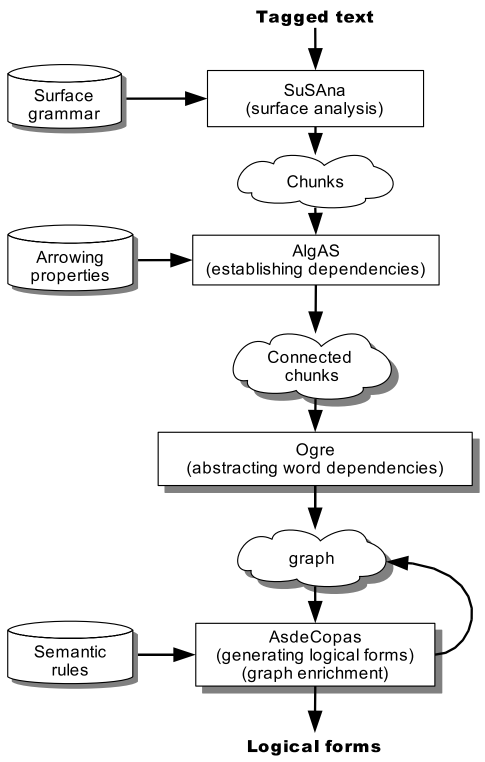
Tagged text
Surface grammar
SuSAna (surface analysis)
Chunks
Arrowing properties
AlgAS (establishing dependencies)
Connected chunks
Ogre (abstracting word dependencies)
graph
Semantic rules
AsdeCopas (generating logical forms) (graph enrichment)
Logical forms

Fig. 1. The overall architecture of the syntax/semantic interface.

find. Currently, the module uses the longest model principle, which states that, all other things being equal, new constituents tend to be interpreted as being part of the constituent under construction (rather than part of some constituent higher in the parse tree) [4]. In the future, other psycholinguistic principles, such as minimal attachment and right association, may be applied. The use of restriction mechanisms by SuSAna is optional for each analysis.

Hierarchical relations can be defined between linguistic structures, allowing the introduction of generic and specific rules. For example, a general rule can be written for a nc (common noun), and a more specific one for nc_s (singular common nouns). The generic rule is used when no specific rule can be applied.

2.2 Establishing dependencies

After the surface analysis, dependencies between chunks and the elements inside them are established, by using the arrow properties from the 5P Paradigm [6, 7, 14] and a tool called Algas.

Arrow properties: The arrow properties from 5P are used in our interface in order to declare dependencies between elements. This formalism can be applied to establish dependencies between words, chunks, or phrases. Arrows can be seen as dependencies but, contrary to the main dependency theories: (i) an arrow is not labeled; (ii) arrows go from dependents to the head [14].

For example, the following arrow property says that within an interrogative sentence (Sint), an interrogative chunk (IntC) with an interrogative pronoun inside (pint) arrows a nominal chunk (NC) on its right ($i \prec k$), as long as there is no other nominal chunk between them ($i \prec j \prec k$).

$\mathsf{IntC}_i(\{\mathsf{pint}\}/) \rightarrow_{Sint} \mathsf{NC}_k$
$\text{-}\{\mathsf{NC}_j\}$

This formalism can be extended with many other restrictions that constraint the relations to be established [9].

Algas: Algas is the tool responsible for connecting chunks and the elements inside them, taking as input arrow properties and information that limits the search space. For example, the user may declare the chunks for which all the elements arrow the last one (the head). As this stands for the majority of the chunks, no calculus is needed in order to compute dependencies inside these chunks, as all the elements arrow the last one. This feature is computationally very useful as it avoids unnecessary computation. Moreover, we can limit the distance of a dependency. This is also very useful, especially when we have to deal with problems like PP-attachment.

Although Algas can skip unanalysable parts of a sentence, some constraints are made to its output, e.g., only one element arrowing itself (the head) is allowed, either inside each chunk or at the sentence level. Also, for the moment, no arrow crossing is allowed (projectivity). We intend to transform this requirement into a user option.

2.3 Abstracting word dependencies

The motivation for this step is to converge different syntactic structures with the same semantics into the same graph. For example, *O Ritz é onde?*, *É onde o Ritz?* and *Onde é o Ritz?*. Although they all have the same meaning, the arrow structures can be different. This step allows to converge these structures into a single representation.

Ogre: Ogre is the tool that generates a graph from the structures produced by Algas. Ogre applies the following rule to Algas output structure: if a chunk arrows another chunk, the head of the first chunk arrows the head of the second chunk. As a results, a graph is obtained, where:

- Each node keeps: a) a word; b) the word category; c) the word's position in the text;
- Each arc (dependency) relates two nodes;

2.4 Generating logical forms

This final step may result in different outputs, according to the task to be performed. For example, in this step, we can enrich the graph; we can perform semantic disambiguation; we can calculate semantic representations. This step uses hierarchically organized semantic rules and is performed by AsdeCopas.

Semantic rules: Each semantic rule specifies how an element or set of elements will contribute to the output, according to its context. The context of the elements is given by dependencies. A semantic rule has the following syntax, which is similar to the rules used by the IFSP [3] in the syntactic analysis:

[R$_i$] `elements_to_translate` : `arrowing_ context` $\mapsto$ `translation`
where

- `elements_to_translate` is a possibly empty set of elements.
  Each element has the form `elem(word, cat, X)`
- `word` is the word to translate (it can be undefined) and `cat` is its category.
- `arrowing_context` is a possible empty set of arrowing relations.
  Each arrow is a triple `arrow(word1, cat1, X1, word2, cat2, X2, d, l)` where
  - `word1` is the word associated with the source element and `word2` the word associated with the target (both can be undefined);
  - `cat1` is the category of the source element and `cat2` of the target (or categories subsuming them);
  - `dir` $\in$ {L, R} $\cup$ {_} is the direction of the arrow (from right to left (L), from left to right (R) (this field may be undefined when the direction is irrelevant);

   - `X1` is the variable that identifies the source and `X2` the target;
   - `l` is a possibly undefined label;
- `X` is a variable, that identifies the element.
- `translation` is the set of translation functions that will lead to the final logical representation.

Informally, each semantic rule has the following semantics: every time the input text unifies with `elements_to_translate` and verifies `arrowing_context` it is transformed according to `translation`

Additionally, these rules are organized in a subsumption hierarchy. As a result, if a set of semantic rules can be applied to a particular part of a graph, only rules that do not subsume other rules (that is, the most specific rules) are applied. This subsumption relation is detailed in [8], [11] and [10]. Note that this characteristic allows us to define default rules that can be triggered when no other rules apply.

As an example, consider proper nouns (npr). If a npr arrows a verb (which happens in situations such as *O Ritz tem discoteca?*[4]), it will be responsible for generating an entity. If it arrows a noun (*O hotel Ritz tem discoteca?*[5]) it is the noun that generates the entity. In both cases, the npr adds information to the formula: it names an entity. We may represent this information in two rules. The first will be a default rule, that can be applied whenever we have a npr. The second is a more specific rule, that can be applied when an npr arrows a common noun (nc) . Assuming that we have the translation functions sem(X) that returns the predicate associated with the element identified by X, and var(X) a variable associated with the element identified by X, we have the following semantic rules:

[R$_1$]{elem(_, npr, X)} : $\emptyset \mapsto$ {`NAME`(var(X), sem(X)}.

[R$_2$]{elem(_, npr, X1)} : {arrow(_, npr, X1, _, nc, X2, _)} $\mapsto$ {`NAME`(var(X2), sem(X1)}

Rule R$_2$ is more specific that rule R$_1$ because its arrowing context is subsumed by the arrowing context of rule R$_1$ (which in this particular case is empty). Thus, when we have an npr arrowing an nc, only the second rule is applied.

AsdeCopas: AsdeCopas is based on a very simple algorithm. It goes through each graph node and does the following:

- identifies rules that could be applied to that node (this process is a simple unification);
- chooses the most specific rules (depending on the rules hierarchy);
- triggers the rules.

AsdeCopas controls the generation of variables. Although apparently naif, this is an important feature of AsdeCopas. Instead of randomly generating variables, each variable index is the position that the related word occupies in the

[4] Does Ritz has a disco?
[5] Does the hotel Ritz has a disco?

text. As a result, different semantic processes can run at different times and the results merged at the end.

3 Applications

This syntax/semantics interface was used to generate logical forms. In this task, quantification was ignored. As an example of a typical output, the question *Quais os hotéis com piscina de Lisboa?*, leads to the following representation.

```
R10:    ?x11,
R1:     hotéis(x11),
R7:     com(x11, x13)
R1:     piscina(x13),
R7:     de(x11, x15),
R2:     NAME(x15, Lisboa)
```

Details about this task can be found in [11] and [10].

The syntax/semantic interface was also applied in a semantic disambiguation task. The goal was to disambiguate a set of quantifiers, given the syntactic context. Semantic rules, as defined in 2.4, were especially useful in that task.

4 Conclusions and Future work

Regarding the problems identified in system Edite, we can conclude the following:

- in what concerns the system extention, the process of adding a new syntactic rule to Edite resulted in uncontrolled over-generation of information. In our interface, SuSAna controls a possible over-generation recurring to preferences, and as AsdeCopas is based on hierarchically organized self-contained rules (only the most specific rules are triggered) over-generation is limited;
- concerning the capacity of dealing with partial results, in the case of Edite, if a sentence was not syntactically or semantically covered by the grammar, no answer was given. In our interface, SuSAna performs the analysis, as described in section 2.1, first by applying a relaxed set of restrictions, and then filtering the results. This strategy allows unusual constructions, while avoiding an explosion of results. Even if no results can be achieved, a partial parsing is still returned. In what concerns semantic analysis, AsdeCopas returns partial analyses and, when the expected dependencies are not computed, default rules are applied in most of the cases, granting minimal results;
- finally, in what respects diagnosis, in our interface, it is easy to identify the property or rule that is causing problems, as they are not defined recursively.

Although yet to be proved, we think that AsdeCopas and semantic rules can be applied to any dependency structure, as no restriction is made to the graph taken as input.

Our interface is being applied to question interpretation, quantifier disambiguation and to a more formal framework where scope is simulated by using handles and restrictions over the handles as proposed in MRS [12].

References

1. S. P. Abney. Part-of-speech tagging and partial parsing. In Ken Church, Steve Young, and Gerrit Bloothooft, editors, *Corpus-Based Methods in Language and Speech*, chapter Dordrecht. Kluwer Academic Publishers, 1996.
2. Steven Abney. Chunks and dependendencies: Bringing processing evidence to bear on syntax. In J. Cole, G. Green, and J. Morgan, editors, *Linguistics and Computation*, pages 145–164, Standford, CA, 1995. CSLI Publications.
3. Salah Aït-Mokhtar and Jean-Pierre Chanod. Incremental finite-state parsing. In *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP-97)*, Washington DC, 1997.
4. James Allen. *Natural Language Understanding.* The Benjamin Cummings Publishing Company, 2 edition, 1995.
5. Fernando Batista. Análise sintáctica de superfície. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisboa, Portugal, 2003.
6. Gabriel G. Bès. La phrase verbal noyau en français. In *Recherches sur le français parlé*, volume 15, pages 273–358. Université de Provence, França, 1999.
7. Gabriel G. Bès and Caroline Hagège. Properties in 5P. Technical report, GRIL, Université Blaise-Pascal, Clermont-Ferrand, França, 2001.
8. Luísa Coheur, Nuno Mamede, and Gabriel G. Bés. ASdeCopas: a syntactic-semantic interface. In Fernando Moura Pires and Salvador Abreu, editors, *Progress in Artificial Intelligence: 11th Portuguese Conference on Artificial Intelligence, EPIA 2003*, volume 2902 / 2003 of *Lecture Notes in Artificial Inteligence*, Beja, Portugal, Dezembro 2003. Springer-Verlag.
9. Luísa Coheur, Nuno Mamede, and Gabriel G. Bès. From a surface analysis to a dependency structure. In *Workshop on Recent Advances in Dependency Grammar (Coling 2004)*, Genebra, Suiça, 2004.
10. Luísa Coheur, Nuno Mamede, and Gabriel G. Bès. A multi-use incremental syntax-semantic interface. In *Estal – España for natural Language Processing*, Alicante, Espanha, Outubro 2004. Springer-Verlag.
11. Luísa Coheur, Nuno Mamede, and Gabriel G. Bès. A step towards incremental generation of logical forms. In *Romand 2004: 3rd workshop on RObust Methods in Analysis of Natural Language Data (Coling 2004)*, Genebra, Suiça, 2004.
12. Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A. Sag. Minimal recursion semantics: An introduction. *L&C*, 1(3):1–47, 2001.
13. Luísa Marques da Silva. Edite, um sistema de acesso a base de dados em linguagem natural, análise morfológica, sintáctica e semântica. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, Portugal, 1997.
14. Caroline Hagège. *Analyse Syntatic Automatique du Portugais.* PhD thesis, Université Blaise Pascal, Clermont-Ferrand, França, 2000.
15. Daniel Jurafsky and James Martin. *Speech and Language Processing*, chapter 15. Prentice Hall, 2000.
16. Paulo Reis, J. Matias, and Nuno Mamede. Edite - a natural language interface to databases: a new dimension for an old approach. In *Proceedings of the Fourth International Conference on Information and Communication Technology in Tourism (ENTER'97)*, pages 317–326, Edinburgh, Escócia, 1997. Springer-Verlag, Berlin, Germany.