Automatic Transformational Analysis and Generation

Max Silberztein

Automatic Transformational Analysis and Generation

Max Silberztein
LASELDI, Université de Franche-Comté
max.silberztein@univ-fcomte.fr

Abstract

I present a new automatic transformational engine for NooJ, capable of producing all the paraphrases of any given sentence (elementary or complex). The new engine does not require the implementation of a new level of linguistic description, as it uses slightly enhanced traditional NooJ syntactic grammars.

Keywords: NooJ. Syntactic Analysis. Transformational Analysis.

Introduction

NooJ is a linguistic development environment: it allows linguists to formalize several levels of linguistic phenomena: orthography and spelling, lexicons for simple words, multiword units and frozen expressions, inflectional and derivational morphology, local, structural and transformational syntax. For each of these levels, NooJ provides linguists with one or more formal tools specifically designed to facilitate the description of each phenomenon, as well as parsing tools designed to be as computationally efficient as possible.[1]

At the transformational syntactic level of analysis[2], NooJ allows linguists to associate a given sentence with a paraphrase via an elementary transformation, e.g.:

[Pron-0] *John ate an apple = He ate an apple*
[Passive] *John ate an apple = An apple was eaten by John*

as well as to associate more than one sentence with a complex paraphrase via a complex transformation, e.g.:

[Coord-1] *John ate an apple; John ate a pear = John ate an apple and a pear*

In NooJ, these transformations can be programmed with grammars such as the following one:

---

[1] This approach is the opposite of that of most computational linguistic tools such as XFST, LFG, HPSG, etc. which provide a single formalism supposed to be used to describe everything.

[2] I am using the term *transformation* as in (Harris 1968) and (Gross 1975) : an operator that links semantically equivalent paraphrases, as opposed to (Chomsky 1957) whose transformations link deep and surface structures.

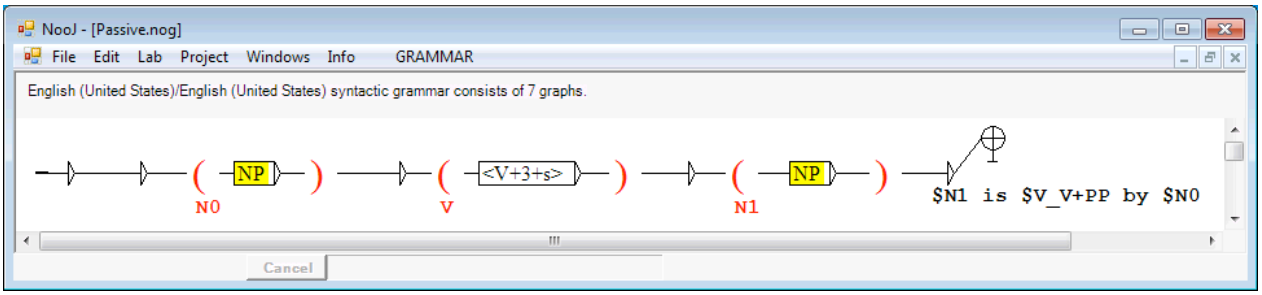

Figure 1. The [Passive] elementary transformation

In this graph, the transformation's arguments are stored in variables ($NO, $V and $N1); for instance, when parsing the sentence *John loves Eva*, the variable $N0 would store the linguistic unit *John*, $V would store *loves* and $N1 would store *Eva*. The grammar's output "$N1 is $V_V+PP by $N0" would then produce the string *Eva is loved by John*. Morphological operations, such as "$V_V+PP", operate on linguistic units rather than plain strings: NooJ knows that the linguistic unit *loves* is a form of the verb *to love*, and it has access to all the linguistic information associated with this linguistic unit: conjugation paradigm, derived forms, syntactic and semantic properties. NooJ is therefore capable of processing the variable $V and applying to it the operator "_V+PP" which stands for: inflect the verb in its past participle form.

This system has been used quite successfully in bilingual Machine-Translation systems, see for instance (Ben Hamadou et alii, 2010) and (Fehri et alii, 2010) for Arabic-French translation, (Barreiro 2008) for Portuguese-English translation and (Wu 2010) for French-Chinese translation.

Any serious attempt at describing a significant part of a natural language will involve the creation of a large number of elementary transformations:

[Pron-0]	*John eats an apple = He eats an apple*
[Pron-1]	*John eats an apple = John eats it*
[Pron-2]	*John gives an apple to Marie = John gives her an apple*
[Progr]	*John eats an apple = John is eating an apple*
[Preterit]	*John eats an apple = John ate an apple*
[Impfct]	*John eats an apple = John has eaten an apple*
[Futur]	*John eats an apple = John will eat an apple*
[Cond]	*John eats an apple = John should eat an apple*
[Passive]	*John eats an apple = An apple is eaten by John*
[Negation]	*John eats an apple = John does not eat an apple*
[Cleft-0]	*John eats an apple = It is John who eats an apple*
[Cleft-1]	*John eats an apple = It is an apple that John eats*
[Question-0]	*John eats an apple = Who eats an apple?*
[Question-1]	*John eats an apple = What does John eat?*
[Question-V]	*John eats an apple = What does John do?*
[Nom-0]	*John loves apples = John is an apple lover*
[Nom-V]	*John loves apples = John's love for apples*
[Nom-1]	*John gave the card to Mary = The card is John's gift to Mary*
…	

Figure 2. Basic transformations

In NooJ v2.x, each of these transformations would have to be described by one graph… actually *two* graphs, because each pair of paraphrases involve two different (reverse) constructions: one wants to be able to produce not only *An apple is eaten by John* from the sentence *John eats an apple*, but also the sentence *John eats an apple* from the sentence *An apple is eaten by John*.

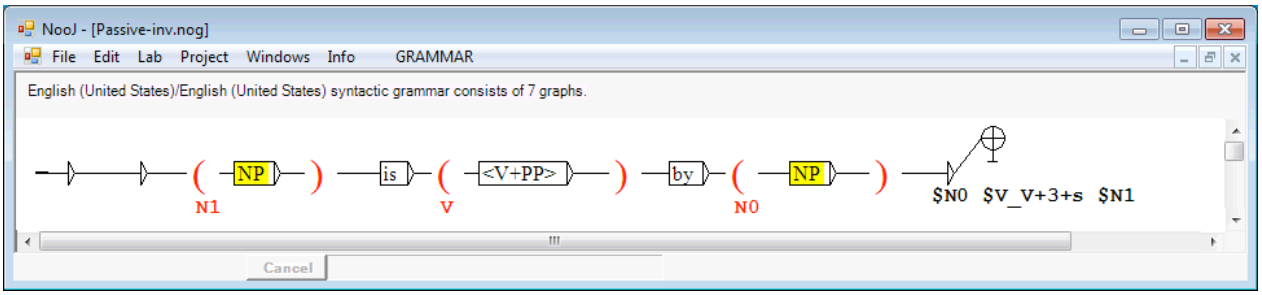

Figure 3. The [Passive-inv] elementary transformation

Unfortunately, even two graphs per transformation would not be enough to produce all the paraphrases of a given sentence, because most of the transformations presented above can be combined with other ones. For instance, from the sentence *John eats an apple*, one can construct the question *What did John not eat?* by combining three transformations: [Question-1], [Preterit] and [Neg]. These combinations of transformations cause the number of pairs of potential paraphrases to explode: for 100 elementary transformations, one would need to construct up to 10,000 pairs of graphs to represent all possible combinations of two transformations, one million pairs of graphs for three transformations, one hundred million pairs of graphs for four transformations, etc.

Clearly, this solution is not adequate.

Chains of transformations

Rather than describing combinations of transformations, linguists have traditionally worked on chains of transformations. For instance, here are two chains of transformations:

John sees a dog **[Cleft-0]**→ *It is John who sees a dog* **[Neg]**→ *It is not John who sees a dog*
John sees a dog **[Neg]**→ *John does not see a dog* **[Cleft-0]**→ *It is John who does not see a dog*

Notice that the order in which transformations are applied is important. As a matter of fact, ordered chains of transformations are often presented as *the* result of a transformational analysis of a sentence. In other words, the very meaning of the sentence *It is John who does not see a dog* is represented by the following formula:

John sees a dog **[Neg] [Cleft-0]**

In principle, chains of transformations would allow an automatic system to produce any potential paraphrase from a given sentence with a large economy of scale because the system would only store elementary transformation graphs: no need to construct a separate graph for each of the numerous possible combinations of transformations.

Unfortunately, this traditional approach creates at least three problems:

1. Complex elementary transformations

It is not possible to design an elementary transformation graph that could be applied to a sentence independently from the sentence's structure. For instance, even such a simple transformation as **[Neg]** is not applied the same way on an elementary sentence and on a complex (i.e. transformed) sentence:

John sees a dog **[Neg]** → *John* does not *see a dog*

John does shopping **[Neg]** → *John* does not *do shopping*
Why does John see a dog ? **[Neg]** → *Why* does not **do John see a dog ?*

John is buying a car **[Neg]** → *John is* not *buying a car*
John is rich **[Neg]** → *John is* not *rich*

John has bought a car **[Neg]** → *John has* not *bought a car*
John has a car **[Neg]** → *John* does not *have a car* | *John has* not *a car*

To produce the correct negation for the sentence *John is buying a car* for instance, one must insert only the adverb *not* but not the verbal form *does*. To produce the correct negation for the question *Why does not John see a dog ?*, one must delete the extra occurrence of the verb *to do*. Sentences with *have* (as well as *need* and *must*) accept two negative paraphrases. Sentences in the futur, in the conditional or in preterit would each require different transformation graphs, etc.[3]

In conclusion: in order to construct a system capable of performing elementary transformations in cascade, one would need to design transformation graphs that could produce paraphrases correctly, not only for elementary sentences, but also for complex (i.e. already transformed) sentences. Such graphs would necessarily include the structure of all their possible inputs, *i.e.* the initial elementary sentence as well as all its possible transformed paraphrases. In effect, building such graphs would not be significantly less complex than building a pair of graphs for each combination of transformations. Moreover, the resulting library of graphs would be unbearably redundant, because every single paraphrase would need to be described in every single transformation graph's input.

2. Ambiguities

The chain of transformations that must be applied to a given elementary sentence in order to produce a given paraphrase constitutes the semantic analysis of the paraphrase. As I showed above, the meaning of the sentence *It is John who does not see a dog* is exactly represented by the formula:

John sees a dog **[Neg] [Cleft-0]**

However, what happens where there is more than one way to produce a complex sentence from an elementary one? The sentence: *It was not the apple that was eaten by him?* seems to have two possible transformational analyses:

John ate an apple **[Pron-0] [Passive] [Cleft-0] [Neg]**
John ate an apple **[Passive] [Cleft-0] [Neg] [Pron-0]**

---

[3] And we have not even discussed complex negations such as in *John wants not to come* or sentences that do not accept negations such as *John never comes* or *John does see the dog*.

Are these two chains of transformation valid? If so, is this sentence really semantically ambiguous, or must these two analyses be unified? If so, what linguistic property can explain that some chains of transformations are to be ordered, whereas others might accept ordering variants? As transformation chains gets longer, the number of transformational paths that can be followed to analyze a single sentence will grow exponentially, and without any theoretical mean to distinguish between equivalent and different chains of transformations, the concept of transformational analysis itself becomes useless.[4]

3. Theoretical Sentences

The self-imposed absolute need find a chain of transformation to link a given complex sentence to its initial elementary sentence is also responsible for another major drawback of transformational grammars: the so-called "theoretical sentences". Theoretical sentences are syntactically invalid sentences that have to be considered as valid in order to activate the transformational analysis of other (correct) sentences.

For instance, consider the French sentence: *La pomme est mangeable* [The apple is edible]. This sentence is traditionally linked to the elementary sentence *On mange la pomme* [One can eat the apple] via the following chain of transformations:

On mange la pomme **[pouvoir]**→ *On peut manger la pomme* **[Passif]**→ *La pomme peut être mangée* **[Able]**→ *La pomme est mangeable.*

Now, how can one analyze the sentence: *Luc est risible* [Luc is laughable] ? The same chain of transformations would be:

On rit de Luc **[pouvoir]**→ *On peut rire de Luc* **[Passif]**→ **Luc peut être rit* **[Able]**→ *Luc est risible.*

However, notice that the sentence **Luc peut être rit* [Luc can be laughed at] is incorrect. How then can one analyze the sentence *Luc est risible*? Either one allows users to add an invalid sentence to the grammar, so that an automatic system can process the complete chain of transformations from *On rit de Luc* to *Luc est risible*; or one forces users to design another chain of transformations that would bypass the invalid sentence, something like:

(a) *On rit de Luc* **[pouvoir]**→ *On peut rire de Luc* **[Passif-Able]**→ *Luc est risible.*

Or alternatively:

(b) *On rit de Luc* **[pouvoir-Passif-Able]**→ *Luc est risible.*

The first solution — which involves including a large number of so-called "theoretical sentences" in the grammar — has usually been chosen by most linguists who don't want to be bothered by mundane details such as the a-grammatical intermediary sentence[5]… The idea that the grammar of a language should contain a large number of a-grammatical sentences is difficult to defend.

---

[4] If a given sentence may have a large number of different analyses, what is the point of the analysis?

[5] Indeed (Gross 1975) is a reaction to the general lack of interest for these "exceptions" and the LADL's project of constructing a systematic description of elementary sentences and their transformations.

The second solution (construct compound transformations to bypass invalid sentences) poses a theoretical problem: how do one decide which transformations will be compounded? How do one choose between the two transformational analyses (a) and (b)? And does it really matter? It makes no sense to give to a higher status to one or the other chain of transformations.

Moreover, this solution leads in practice to building a large number of compound transformations; in essence this solution is not different from the solution of constructing a pair of graphs for each pair of paraphrases; in effect it destroys the very idea of using elementary transformations.

The Automatic Generation of variants

1. Automatic generation of morphological variants

Generating a paraphrase from a given sentence is not very different from generating all the variants recognized by a given grammar. NooJ already can explore and generate all the paths of a morphological grammar. Consider the following morphological grammar **France** in Figure 4.

This grammar recognizes the word form *France* and associates it with the linguistic information "N+NomDePays" (it is a Noun and a name of a country). It also recognizes the word form *refranciser* and associates it with the information "V+FLX=AIDER+Re": it is a verb, conjugates according to the conjugation paradigm AIDER and it contains the repetition prefix.

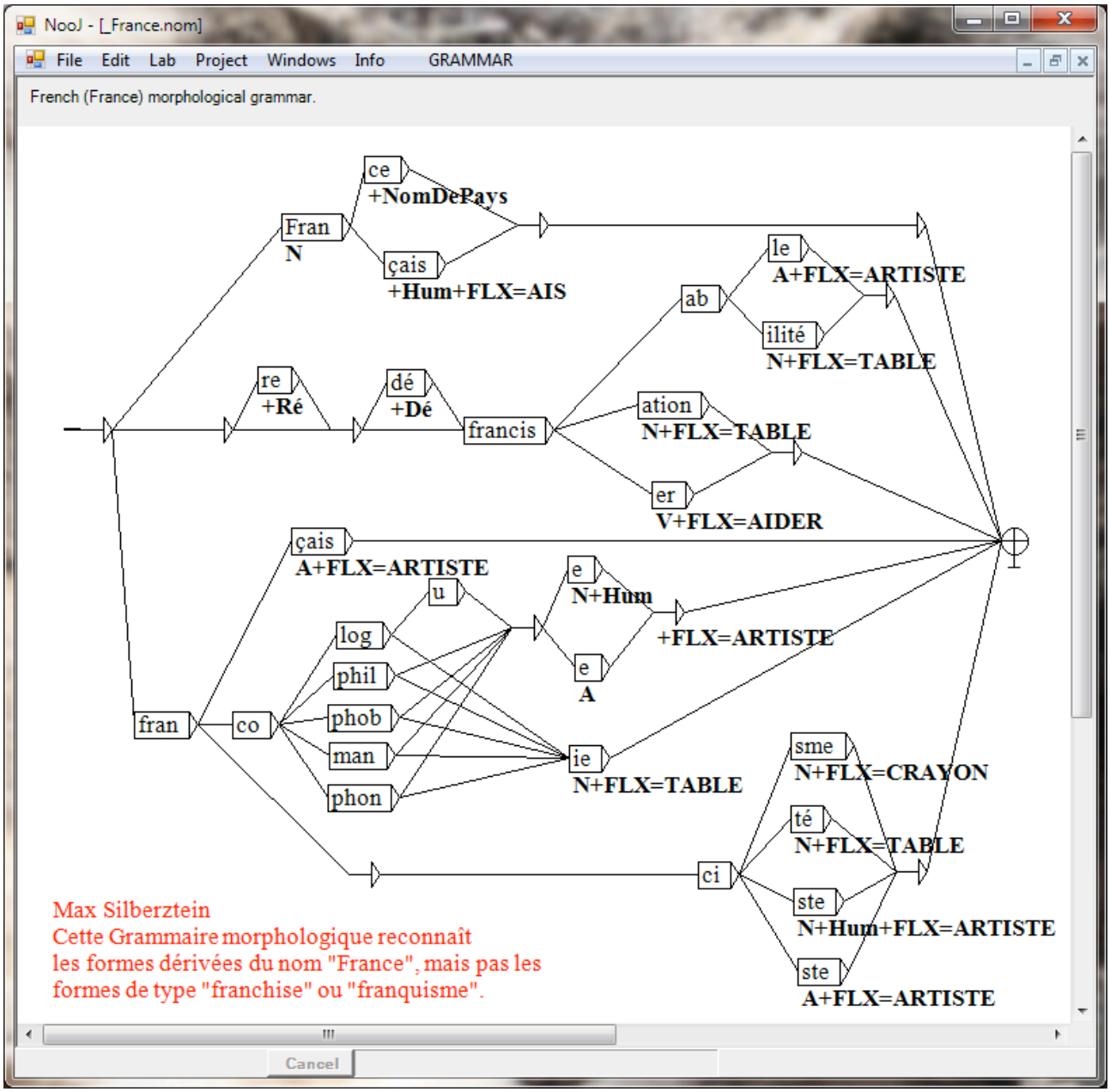

Figure 4. A morphological grammar

NooJ's simple command GRAMMAR > Generate Language can be used to automatically construct the dictionary that contains all the word forms recognized by the grammar, and associates each of the word forms with the corresponding grammar output.

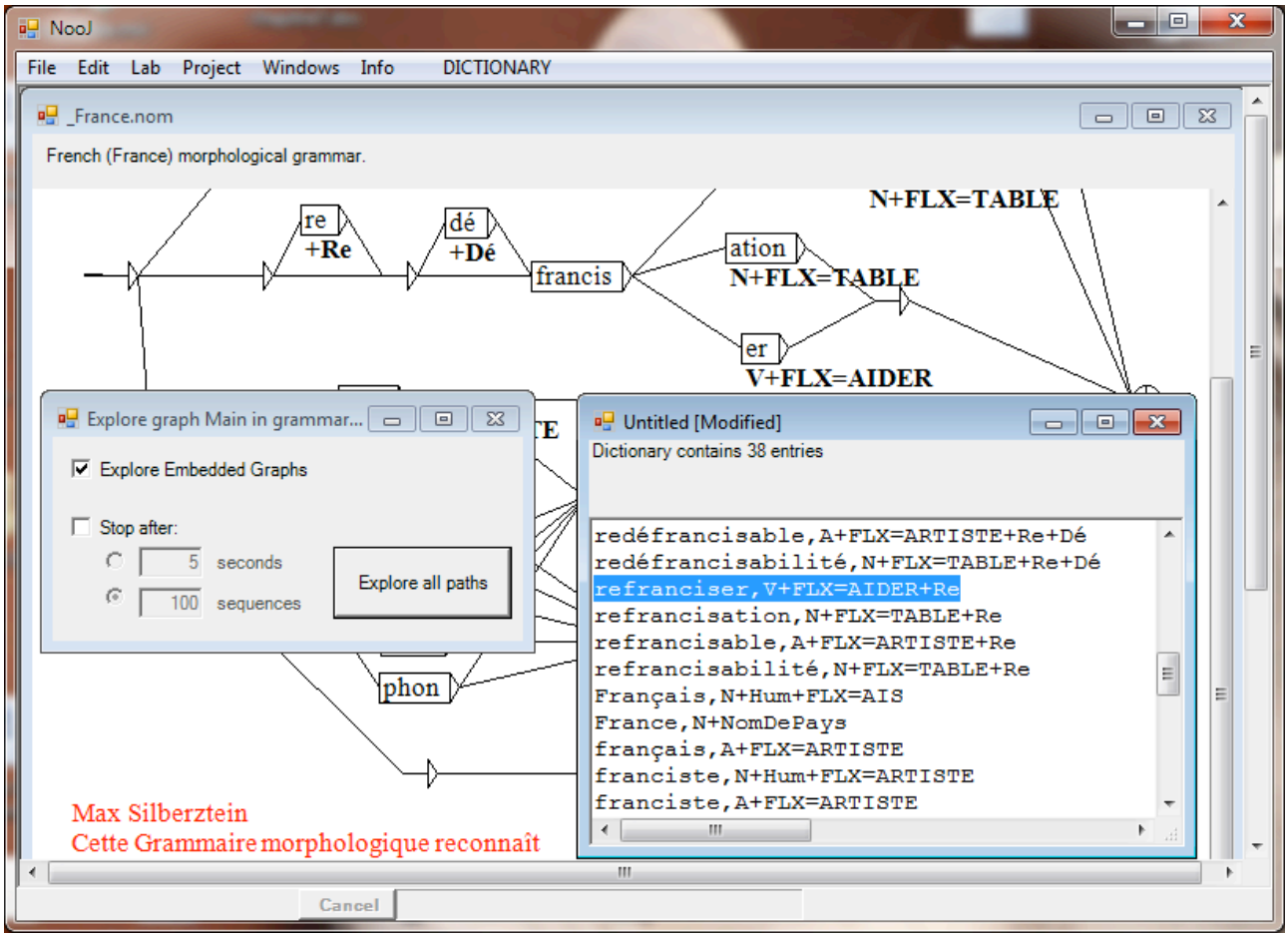

Figure 5. Generate all the word forms derived from *France*

Note that in the resulting dictionary, the linguistic information associated with each lexical entry corresponds in a sense to its linguistic analysis: analyzing the word form *refranciser* as a verb with the **+Re** (repetition) feature is similar to analyzing the sentence *John does not like apples* with the **+Neg** (negation) operator.

2. Automatic generation of syntactic paraphrases

If one constructs a syntactic grammar that recognizes all the paraphrases of a given elementary sentence, automatically generating all these paraphrases then becomes a matter of exploring the grammar. There is, however, a major difference between morphological and syntactic grammars : as opposed to morphological grammars that by design describe a specific family of derived forms, one cannot afford to build one grammar per elementary sentence: one needs to represent abstract structures such as N_0 *V* N_1 rather than plain sentences such as *John eats an apple*.

In order to produce the sentence *An apple is eaten by John* from the latter sentence, one needs to somehow process the sentence at the structure level:

$$N_0\ V\ N_1\ [\text{Passive}] \rightarrow N_1\ \textit{is}\ V\textit{-pp}\ \textit{by}\ N_0$$

and then manage variables so that N_1, *V-pp* and N_0 are correctly set respectively to *An apple*, *eaten* and *John*. And one wants the same exact grammar to produce the elementary sentence *John eats an apple* from the sentence *An apple is eaten by John*.

The solution is then (1) to construct a grammar that will recognize the elementary structure as well as all its paraphrases, (2) make sure the grammar is totally reversible, i.e. that it recognize every paraphrase (not only the elementary one), and (3) adapt NooJ's variable management system and its morphological operators so that it can move variables' values around, and at the same time perform morphological operations if necessary.

The new grammar is very similar to any traditional NooJ grammar that would be used to recognize a sentence and its variants:

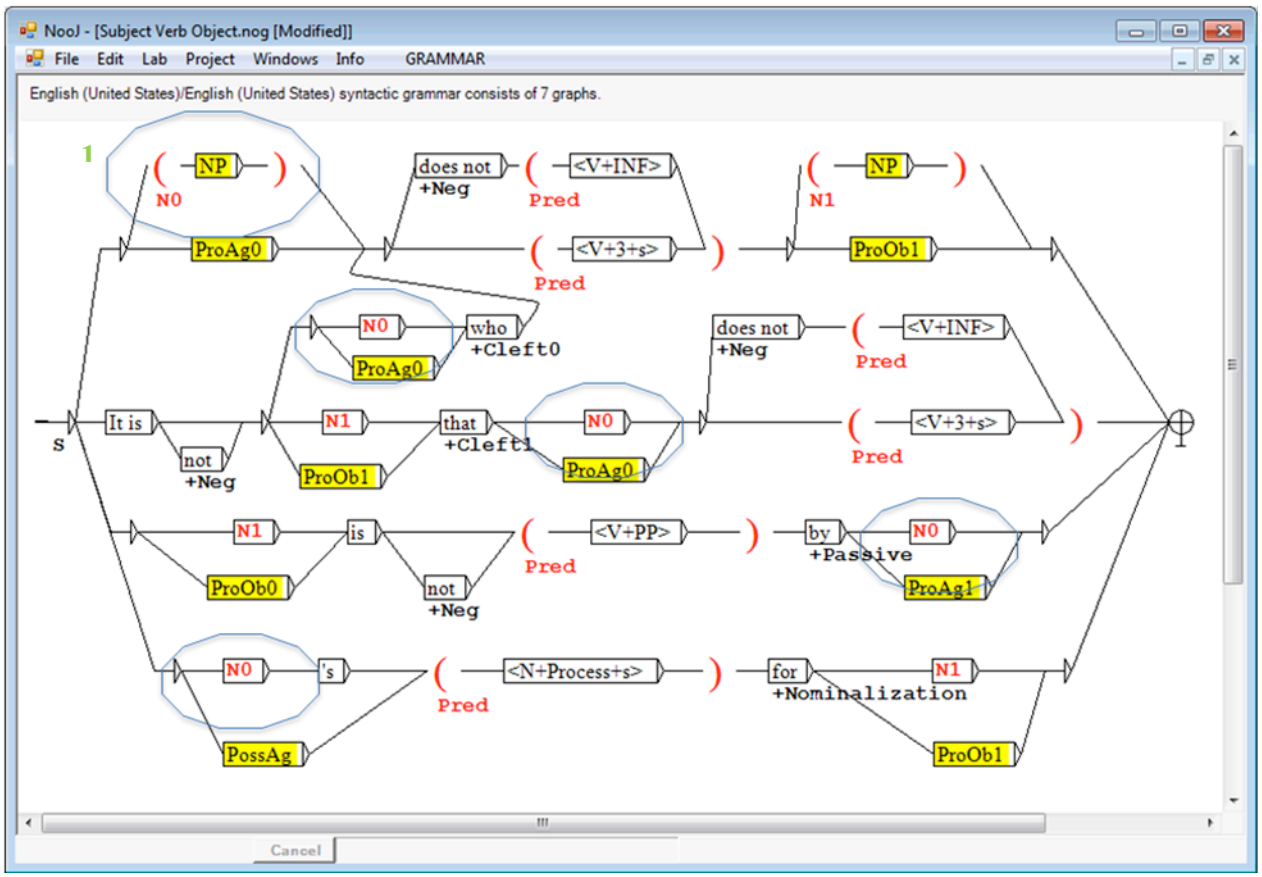

Figure 6. A syntactic grammar recognizes an elementary sentence and its variants

The grammar presented in Figure 6 is a "lighter" version of a syntactic grammar that could be used to perform a structural syntactic analysis of sentences, as described in (Silberztein 2010) and (Vietri 2010), which produce structural trees (I removed the informational structure in the interest of clarity).

3. Enhancements to syntactic grammars

The constraints and enhancements I have added to syntactic grammars so that they can be used as transformational grammars are:

— each subsequence that might be displaced as a whole by a transformation needs to be stored in a variable, here: $N0, $Pred and $N1[6];

— Any variable must be defined exactly once in a given grammar, because NooJ will unify the definition of the variable with every single instance of it in the grammar: this allows NooJ to move the content of the variable around, as well as to perform morphological operations on it.

For instance, in Figure 6, the variable $N0 has been set to the subgraph NP only once (see 1), and all other uses of $N0 in the same grammar refer to this same exact value: all the references $N0 in the grammar are linked to its definition (in the left top of the graph in Figure 6): "$(N0 :NP $)". This constraint allows any of this grammar's paths to match the corresponding paraphrases, rather than only the path at the top of the grammar.

Users might see this feature as a mere facility that makes grammars lighter and thus more readable (it is not necessary to redefine each of the occurrences of a given variable). In fact, however, solving references to a variable is crucial in order to allow NooJ to recognize not only elementary sentences (the path at the top of the graph) but also any paraphrase described in the grammar.

The way NooJ will compute variables' values has been enhanced so that the path $(Pred <V+PP> $) can produce the past-participle form *eaten* from the value of the $Pred variable *eats*, and reciprocally, that the path $(Pred <V+3+s> $) can produce the third person singular form *eats*, from the value of the variable $Pred *eaten*.

Figure 7 displays the automatic generation of all the paraphrases for one given sentence. The grammar produces the name of the transformations associated with each paraphrase, in other words, its transformational analysis. The list of features associated with each paraphrase always represent a transformation chain that would be performed on the elementary sentence, and not on the (possibly complex) sentence the grammar was applied to. In Figure 7, we see that the grammar, when given the complex sentence "Eva is not loved by Paul", has produced 34 paraphrases: each of these paraphrases is associated with the chain of transformations that links it to the elementary sentence: John loves Eva. For instance, the resulting sentence "Eva is loved by him" is associated with the chain S+Passive+Pro0, where Pro0 represents the pronominalization of the subject of the elementary sentence *Paul loves Eva* rather than the subject of the sentence *Eva is not loved by Paul*.

NooJ can be used to produce all the paraphrases of a given sentence (not necessarily an elementary one), or to produce only its paraphrases compatible with a certain transformational analysis. For instance, given the transformation chain "+Passive+Neg", NooJ will produce the four sentences:

she is loved by Paul, S+Pro1+Passive
she is loved by him, S+Pro1+Passive+Pro0
Eva is loved by Paul, S+Passive
Eva is loved by him, S+Passive+Pro0

[6] This very criterion is the one (Gross 1975) uses to characterize the syntactic constituents of a sentence.

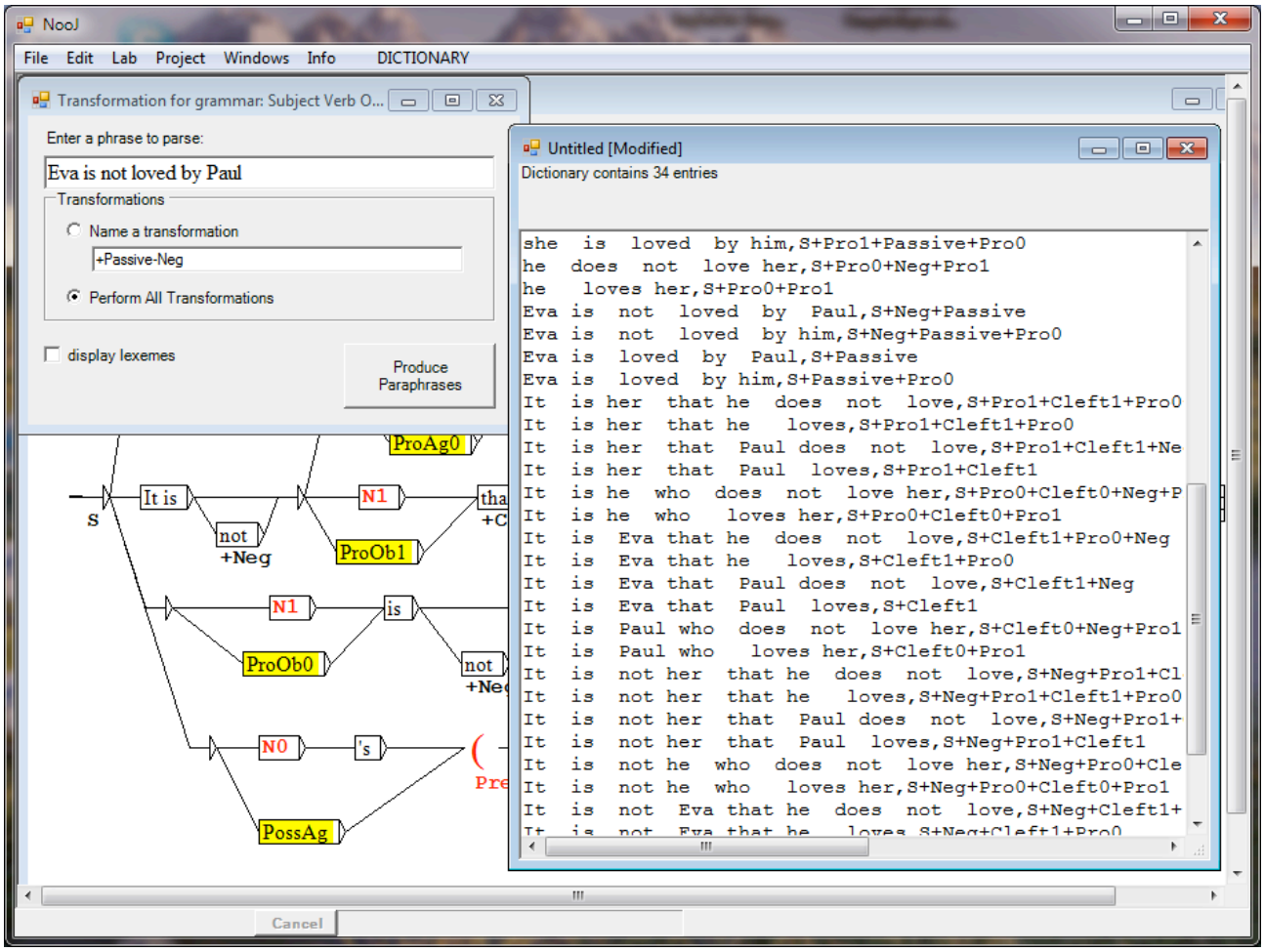

Figure 7. Produce all the paraphrases of the sentence *Paul loves Eva*

Conclusion

In this paper, I have presented an automatic transformational engine capable of producing any paraphrase from any given sentence (elementary or complex) , as well as producing for any sentence its transformational analysis: its corresponding elementary sentence and the chain of transformations that link the elementary sentence to it.

As opposed to the traditional point of view on transformational grammars, the system presented here does not require linguists to implement a new level of linguistic description that would be different from the syntactic structural level: no need to implement specific transformational operators nor design a complex set of mechanisms to process chains of transformations.

NooJ's new transformational engine might be used in several Natural Language Processing applications, such as Question Answering: a question such as *Who ate the apple?* can be considered as a paraphrase of *John ate the apple*. If NooJ can link a question (typed in by a user) to any of its potential answers (occurring in a large corpus such as the WEB), it can answer the question automatically.

The functionalities presented here are a first attempt at the construction of a large-coverage formalization of elementary transformations: it will be necessary to test it on a large scale. In the future, I will need to design new grammars to represent more complex transformations, i.e. transformations that produce a complex sentence from more than one elementary sentence.

References

Ben Hamadou A., Piton O., Fehri H., 2010. Recognition and Arabic-French translation of named entities: case of the sport places. In the proceedings of the *NooJ 2009 International Conference and Workshop*. Sfax: Centre de Publication Universitaire, pp. 271-284.

Barreiro A, 2008. Port4NooJ: Portuguese Linguistic Module and Bilingual Resources for Machine Translation. In Xavier Blanco & Max Silberztein (eds.), *Proceedings of the 2007 International NooJ Conference*. Newcastle: Cambridge Scholars Publishing, pp. 19-47.

Chomsky N., 1957. *Syntactic Structures*. The Hague: Mouton.

Fehri H., Haddar K., Ben Hamadou A., 2010. Integration of a transliteration process into an automatic translation system for named entities from Arabic to French. In the proceedings of the *NooJ 2009 International Conference and Workshop*. Sfax: Centre de Publication Universitaire, pp. 285-300.

Gross M. 1975. Méthodes en syntaxe. Paris: Hermann.

Harris Z. 1968. Mathematical Structures of Language. Englewood Cliffs, NJ: Interscience.

Silberztein M. 2010. Syntactic parsing with NooJ. In the proceedings of the *NooJ 2009 International Conference and Workshop*. Sfax: Centre de Publication Universitaire, pp. 177-190.

Vietri S. 2010. Building structural trees for frozen sentences. In the proceedings of the *NooJ 2009 International Conference and Workshop*. Sfax: Centre de Publication Universitaire, pp. 219-230.

Wu Mei. Integrating a dictionary of psychological verbs into a French-Chinese MT system. In the proceedings of the *NooJ 2009 International Conference and Workshop*. Sfax: Centre de Publication Universitaire, pp. 315-328.