Variable Unification in NooJ v3
Max Silberztein

▶ To cite this version:
Max Silberztein. Variable Unification in NooJ v3. Automatic Processing Of Various Levels Of Linguistic Phenomena: Selected Papers from the NooJ2011 International Conference, Jun 2011, Dubrovinic, Croatia. hal-02435917

HAL Id: hal-02435917
https://hal.science/hal-02435917
Submitted on 11 Jan 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
VARIABLE UNIFICATION IN NOOJ V3
MAX SILBERZTEIN

Abstract

NooJ’s linguistic engine integrates all its parsers (from the lexical to the syntactic level) with its morphological and paraphrase generators. In particular, both NooJ’s syntactic parser and NooJ’s transformational generator use the same single syntactic grammar. Because of this new level of integration, we had to design a new mechanism to manage variables and compute their value.

Introduction

One of NooJ’s specificities among the Natural Language Processing frameworks is that NooJ allows linguists to formalize many levels of linguistic phenomena, and offers for each level one or more specific formalisms and computational devices. NooJ uses an annotation mechanism (stored in each Text Annotation Structure, or TAS) that integrates every single piece of linguistic information; this integration makes it possible to combine morphological constraints in syntactic rules, for instance. Perhaps the most striking application of this total integration is that NooJ can perform transformations automatically without requiring linguists to “program” transformation rules.

NooJ’s transformations are different from Chomsky’s. Chomsky’s transformations are oriented derivations performed on sentences that can be combined to produce, in one direction, a deep structure (or the D-structure in the government and binding theory) or in the other direction a surface structure (or S-structure).¹ NooJ’s transformations are symmetric like Harris’ transformations, but their nature differs fundamentally from Harris’² in the sense that they are an implicit property of a syntactic grammar rather than constituting a level of grammar that needs to be explicitly represented or “programmed” by linguists.³

¹ See (Chomsky, 1965).
² See (Harris, 1981).
³ See (Silberztein, 2011).
In order for a given syntactic grammar to be able to be used both by NooJ’s syntactic (structural) parser\(^4\) and by its transformational parser/generator, I had to profoundly redesign the way NooJ’s linguistic engine was managing variables and computing their values.

In its v2 version, NooJ computed the value of each occurrence of a given variable without keeping it in memory. This feature allowed a particular variable to obtain different values along the parsing path. In consequence, grammar designers could set and use a given variable in a loop: the variable would obtain a series of different values; as many values as times the parser would visit the loop.

For instance, consider the following grammar (Figure 1), used to translate the English noun phrase \textit{the big beautiful table} into its French equivalent \textit{la grande belle table}:\(^5\)

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure1.png}
\caption{variable $V$ has multiple values}
\end{figure}

Note how the same variable $V$ obtains four values during the parsing of the English noun phrase: $V$=“the”, then $V$=“big”, then $V$=“beautiful”, then $V$=“table”. Each of these words is translated correctly because its translation ($V$FR) is used exactly where the variable $V$ is set (\textit{i.e.} $V$ is in sync with $V$FR).

This type of grammar is useful in processing sequences of text “word by word” and has other advantages as well.

For instance, variables can be encapsulated safely inside embedded graphs and/or loops because each variable’s value is always computed locally: no fear that a given variable would be used inadvertently in a

---

\(^4\) Cf. (Silberztein 2010).

\(^5\) For the sake of discussion, this grammar is overly simplified: one would need to add agreement constraints on the noun’s gender and number: the French translation for \textit{table} is feminine singular, thus the words \textit{la, belle} and \textit{grande} must be in feminine singular.
different graph for different purposes. As a result, graphs in a NooJ grammar were largely independent objects so they could — at least in theory — be shared and distributed as opaque “black boxes”.6

The way NooJ v2 handled variables looked very much like a great original idea!

A Dead End

Unfortunately, the very fact that NooJ was not unifying all values of each occurrence of a given variable prevented grammar designers from combining agreement constraints and propagating them along a parsing path. For instance, consider the following grammar (Figure 2):

![Grammar Diagram]

This grammar contains two constraints:

- \(<$D$Nb\\=$W$Nb>\) (assuming that $W$ is a noun) checks that it agrees in number with its determiner,
- \(<$P$Pers\\=$W$Pers>\) (assuming that $W$ is a verb) checks that it agrees in person with the following pronoun.

---

6 Graphs are not totally insulated: if a variable is set in a graph $A$, it can still be seen in another graph $B$, provided that its value was computed in $A$ before it was used in $B$… As a matter of fact, graphs were never intended to be autonomous objects that could be distributed and shared independently from the grammar that encapsulates them. In NooJ — as opposed to INTEX, see (Silberztein, 1993) — the atomic piece of grammar is the grammar, not the graph.
In other words, \( W \) takes the role of both a noun (in the first constraint) and a verb (in the second constraint): it does not make any sense, and such an inconsistent grammar should not produce any result.

However, because \( W \) is computed independently each time NooJ needs its value, an ambiguous word that could be both a noun (that agrees in number with the preceding determiner) or a verb (that agrees in person with the following pronoun) will indeed be recognized by this grammar, e.g. in the following sequence:

\[ ... \text{some cooks he} ... \]

the word *cooks* can be either a plural form of the noun *cook* (in which case it does agree with the plural determiner *some*), or a conjugated form of the verb *to cook*, in which case it agrees with the following pronoun *he*.

As grammars contain more and more lexical constraints and longer and longer sequences (which include ambiguous words), the parsing results become less and less reliable. For instance, consider the following grammar (Figure 3):

![Figure 3: Three constraints](image)

This grammar recognizes sequences constituted by a determiner, a noun and a verb: this useful grammar can be applied to large corpora when one wants to index and retrieve sequences such as “the director runs…”, “his car stopped”, etc.

The constraint \(<D\text{Nb}=$N\text{Nb}>\) checks that the determiner agrees in number with the noun; the constraint \(<N\text{Nb}=$V\text{Nb}>\) checks that the noun agrees in number with the verb; the constraint \(<N\text{Sem}=$V\text{SubSem}>\) checks that the noun’s semantic class corresponds to the verb’s subject’s semantic class. This grammar correctly recognizes sequences such as:

\[ ... \text{A woman says} ... \]
(“a” agrees in number with “woman”; “woman” agrees in number with “says”; “woman” is a proper agent for the verb to say). However, it will also recognize a large number of incorrect sequences, in particular if the middle word stored in variable $N$ is actually ambiguous. For instance, if $N$ is associated with three lexemes: one lexeme could agree with the determiner (but not with the verb), another lexeme could agree with the verb (but not with the determiner), and the last one could agree semantically with the verb (but not in number)…

In conclusion: although grammars such as the one in Fig. 3 are seemingly simple and very natural from a linguistic point of view, the way NooJ v1 interprets them make them unreliable at best; their application to large corpora typically produces considerably too many false results.

Clearly, the benefits of computing variables’ values dynamically without keeping track of the resulting values are negated by the fact that linguists cannot combine constraints in grammars reliably. We need to make sure a given variable in a grammar holds the same value everywhere it is used, i.e. we need to unify all the values of each variable.

**NooJ v3’s Variable Management Mechanism**

In the new v3 version, Nooj uses a single variable space to make sure that every variable in a given grammar has only one value (more precisely: one lexical unit). Thus the new engine processes the grammars in Fig. 2 and Fig. 3 reliably.

In order to allow grammar designers to use variables in loops (such as in Fig. 1), I have added the special variable $THIS$ which always refers to the current lexical unit and thus “simulates” the way v2 bound variables to their values. This time, however; there is no need to define the variable $THIS$: it always refers to the current lexical unit. The grammar shown in Fig. 1 could then be rewritten as the following grammar:
Figure 4: Variable $THIS$ can be used in loops

$THIS$ takes four different values when parsing the noun phrase *the beautiful red table*, and $THIS$FR computes their translation correctly.

However, some v2 grammars will still need to be rewritten to v3, as it is no longer possible to use the same variable name in different graphs of a single grammar to refer to different objects.

This is unavoidable; I hope NooJ users who have relied on the possibility of using one single variable name to refer to different objects in one grammar will forgive me! The new design has allowed us to escape from the dead end of the dynamic computation of each variable’s occurrence. In the meantime, I believe that the new design enforces the unity of every variable’s value and thus will help us build more robust grammars that are more natural from a linguistic point of view.

In the short term, the new design has already allowed us to add new, exciting functionalities to NooJ’s transformational module.

**Variables’ Multiple References**

Version v3 already takes advantage of the fact that each variable has indeed one unique value in order to allow grammar designers to simplify each grammar’s graph massively. Consider, for instance, the following grammar, which represents simple transitive sentences:
In this graph, note how variable $N0 is defined at the top left of the graph, and is used in the input of the grammar at the bottom right of the graph. One may see the reference to $N0 as a mere “abbreviation” of its definition, and indeed, this abbreviation allows grammar designers to draw more compact and elegant graphs.

But NooJ’s new variable system does much more than that: it actually keeps a live link between all the occurrences of a variable and its unique value: technically, if one applies the grammar of Fig. 5 to the sentence John sees Mary, NooJ’s syntactic parser sets “John” to variable $N0 and “Mary” to $N1. Then, the paraphrase generator uses these same exact values when it explores the paths at the bottom of the graph to produce the passive sentence Mary is seen by John (as opposed to John is seen by Mary). In other words: the new variable mechanism allows us to design parameterized grammars, i.e. grammars that contain variables set during a syntactic parsing.

The situation is more complicated for variable $Pred which is defined twice in the grammar: at the top of the graph, we have $Pred = <V+3>, whereas we have $Pred = <V+PP> at the bottom of the graph. That certainly looks like an inconsistency because a variable should not have more than one value. However, note that during the parsing of any sentence, this variable will, in fact, be set only once:
either the sentence is in the active mode (e.g. *John sees Mary*), then we get \( \$\text{Pred} = <V+3> \) (e.g. \( \$\text{Pred} = \text{“sees”} \)), or

- the sentence is in the passive mode (e.g. *Mary is seen by John*), then we get \( \$\text{Pred} = <V+PP> \) (e.g. \( \$\text{Pred} = \text{“seen”} \)).

The fact that the same variable \( \$\text{Pred} \) is used to hold two different values is not an issue because these values will never be alive at the same time during the syntactic parsing of any sentence. That, however, is no longer the case for NooJ’s paraphrase’s generator which computes both \( \$\text{Pred} = \text{“sees”} \) when it produces paraphrases such as *John sees her*, and \( \$\text{Pred} = \text{“seen”} \) when it produces passive sentences such as *Mary is seen by John*.

What then is the true value stored by NooJ for \( \$\text{Pred} \)? The answer is in fact a linguistic one: NooJ considers both lexemes \(<\text{sees, see, V+PR+3+s}> \) and \(<\text{seen, see, V+PP}> \) to be instances of the single lexical unit \(<\text{see, V}> \), but with different properties. In other words, both word forms “sees” and “seen” are processed by NooJ as two variants of a unique linguistic unit. When NooJ’s parser matches the variable \( \$\text{Pred} \) in the context of the symbol \(<V+3> \), it produces the form “sees” whereas when the parser matches \( \$\text{Pred} \) with the symbol \(<V+PP> \), it produces the form “seen”.

Note that this behavior is just a generalization of the way NooJ v2 already processed complex variables: just as \( \$V\text{Gender}, \$V\text{Nb} \) or \( \$V\text{Tense} \) refer to different property values for the same lexical unit, \( \$V \) in the context of the lexical symbol \(<V+3> \) and in the context of the lexical symbol \(<V+PP> \) produce two different lexeme values.\(^7\)

In conclusion: just like the two variables \( \$N0 \) and \( \$N1 \), the variable \( \$\text{Pred} \) takes its (unique) value when NooJ parses the input sentence, e.g. *John sees Mary*. When NooJ’s paraphrase generator produces sentences in the active such as *John sees her* (thanks to the symbol \(<V+3> \)), NooJ instantiates \( \$\text{Pred} \) with the lexeme value “\(<\text{sees, see, V+PR+3+s}> \)”. When it produces passive sentences such as *she is seen by him*, NooJ matches the lexical unit \( \$\text{Pred} \) with the symbol \(<V+PP> \), which in effect computes the lexeme “\(<\text{seen, see, V+PP}> \)”.

---

\(^7\) Internally, NooJ’s parser uses the notation \( \$V\_V+PP \) and \( \$V\_V+3 \). This functionality, unique to NooJ, is made possible by the fact that all of NooJ’s engines (including the morphological parser and generator) are integrated. Note that \( \$V\_V+3 \) produces actually three conjugated forms: “sees”, but also “see” (third person plural) and “saw” (third person in the preterit). The grammar contains an agreement constraint that filters out all the forms that do not agree with the subject.
This mechanism is reversible when the number of linguistic units is constant: for instance, if one enters the sentence *John is seen by Mary*, then the same grammar will produce the sentence *Mary sees John*: the constraint $<V+PR>$ will produce the form “sees” from the lexeme $<seen, see, V+PP>$. However, if one enters the sentence *he sees her*, NooJ cannot compute values for the lexical symbols $<DET>$ and $<N>$ (used in the NP0 and NP1 graphs); as a result, NooJ confines itself to displaying these symbols.

**Agreements**

The grammar shown in Fig. 5 can parse any direct transitive sentence that contains a noun phrase or a pronoun followed by a verb and then a noun phrase or a pronoun, as well as its passive form. There are two levels of agreement checks that need to be discussed:

1. One needs to take several syntactic agreement constraints into account, such as the one between the determiner and the noun and the one between the subject and the verb. For instance, the grammar must reject the incorrect sentence *The men sees the apple*

   NooJ v2 was already capable of processing these types of constraints: we simply need to add them into the grammar (Figure 6):
Figure 6: Agreement constraints used by the syntactic parser

The constraint <$THIS$Nb=$Head0$Nb> located under the node <$V+3$> checks that the verb’s number property is identical to the head of its subject’s noun phrase; $HeadN0$ is defined in the embedded graph NP0:
In the same way, the constraint \(<\$\text{THIS}\$\text{Nb}=?\text{PR00}\text{Nb}>\) checks that the verb agrees with its subject pronoun.

(2) One also needs to take care of agreement constraints between the two noun phrases and their corresponding pronouns, which are produced by the transformational generator. For instance, when NooJ produces paraphrases for the sentence *John sees Mary*, it must not produce sentences such as *she sees Mary* nor *she sees them*. Although these latter sentences are correct syntactically, they do not constitute correct paraphrases of the original sentence and thus should not be produced by the paraphrase generator.

Thanks to NooJ v3’s new variable management mechanism, though, this type of agreement constraint is actually implemented in the most straightforward way: one just needs to add the corresponding constraint, as if the variables that hold the pronouns had actual values in the input sentence. In a way, the grammar already tells us that the pronouns he and her are somehow implicitly present in the sentence *John sees Mary*; we just need to filter out the wrong pronouns and keep the one that agrees with John (i.e. *he*) and the one that agrees with Mary (i.e. *her*).
Let’s turn to the final version of our grammar:

The two constraints <$PR00$Nb=$Head0$Nb> and <$PR00$Gender=$Head0$Gender> check that the subject pronoun — which is produced by the paraphrase generator — agrees in number and in gender with the subject noun phrase (i.e. John) of the input sentence.

In conclusion, the new v3 engine allows variables to be used not only by the syntactic parser, but also by the paraphrase generator. (Vietri, 2012) shows how the new engine is used to produce transformations for idiomatic expressions formalized in a lexicon-grammar table.

---

I am using indices to refer to the role and to the syntactic function of each pronoun. For the input sentence John sees Mary, $PR00$ is the pronoun associated with the agent (0) role in the subject (0) syntactic position, i.e. he; $PR01$ is the pronoun associated with the agent (0) role in the complement (1) syntactic position, i.e. him; $PR10$ is the pronoun associated with the theme role in the subject position, i.e. she; $PR11$ is the pronoun associated with the theme role in the complement position, i.e. her.
Perspective: NooJ CESAR project

I apologize for the compatibility problems that the new variable engine might create for some NooJ users; I do believe that the “flexible” dynamic way of processing variables was indeed a dead end, and I have shown that with the new engine, not only can we design grammars that are as powerful as the ones created with NooJ’s previous versions (thanks to the new $THIS variable): we can also design grammars that can be used both by NooJ’s syntactic parser and by its transformational generator. I believe this makes NooJ truly different from all other Natural Language Processing Tools with which transformations have to be “programmed” explicitly.
The European META-SHARE CESAR project has chosen to use NooJ as their linguistic corpus processor, and a team at the Mihajlo Pupin Institute (Belgrade) has already started to port NooJ into the MONO framework, which will allow it to run on most operating systems, including LINUX, UNIX and Mac OSX. The upcoming v3.1 version will be based on v3.0 (the .NET version) and will incorporate the new variable management mechanism.

References