MATHEMATICA-BASED NATURAL LANGUAGE PROCESSING IN APPLIED MECHANICS

Nikolaos I. Ioakimidis
Division of Applied Mathematics and Mechanics, School of Engineering, University of Patras,
P.O. Box 1120, GR-261.10 Patras, Greece
Tel.: +30 61 432-257, Fax: +30 61 433-962, E-mail: ioakimidis@otenet.gr

ABSTRACT
The probably novel possibility of using the computer algebra system Mathematica in NLP (natural language processing) is briefly studied in few elementary applications from applied mechanics. At first, Buchberger's Mathematica-based Theorema system was found able to lead to a completely natural proof of an elementary propositional-type example from applied mechanics. The same system, Theorema, was also found useful for the syntactic analysis of related very simple sentences, a classical field of application of the Prolog logic-oriented language. The next step has been to use Prolog itself as an external package to Mathematica. Finally, elementary NLP has been achieved directly with Mathematica. The present results show that NLP is within Mathematica's reach, offering the further advantage of performing symbolic computations (contrary to logic-oriented languages). Therefore, probably, gradually, Mathematica (independently of Theorema and Prolog) could lead to an automated environment of both text and formula checking (including symbols and computations) in applied mechanics and mechanical engineering in general.

KEYWORDS

INTRODUCTION
As is extremely well known, computer algebra systems have played (and still play) a very important role in applied mechanics and mechanical engineering in general long ago. Among these systems, we will pay again attention to Wolfram's Mathematica [1], which is the most recent (the first version was released in 1988) and powerful one (especially in its latest versions; here we will use version 4). Thanks to (i) its own efficient logical commands (such as LogicalExpand), (ii) the very recent and extremely powerful Mathematica-based theorem proving Theorema system [2] by Buchberger and the Theorema Group (1997-today), (iii) Maeder's Prolog interpreter for Mathematica [3] (released in 1994) as well as (iv) the ability of Mathematica to run external programs too (e.g. standard Prolog and McCune's OTTER, which we extensively used in [4], both based on resolution), Mathematica is completely able to undertake difficult computational tasks related not only to algebra, but also to logic. We have already seen in the companion paper [5] that Mathematica can verify logical and algebraic formulae by using one of the above approaches.

It is this author's impression that Mathematica could also become much more friendly to its user if it could be possible for it to accept its logical input commands in a more natural, human language. For example, in the companion paper [5], we met the logical formula Fracture ⇒ Replacement (as a fracture axiom, better an assumption, there). Most probably, it would be much better if it could be possible for us to write this formula in a more natural form, i.e. "fracture causes replacement". Similarly, if we have a helical spring S in machine design [6], we are obliged to write the Mathematica commands Spring[S] and Helical[S] or, preferably, their logical conjunction Spring[S] ∧ Helical[S] and, next, proceed to our reasoning with the help of Mathematica. This is the approach having been followed so far. Alternatively, instead of this approach, in this paper we simply suggest the use of a more NL (natural language) in Mathematica, e.g. simply by...
writing: “S is a helical spring” and letting Mathematica proceed with this much more natural, human sentence. Of course, when using Prolog (see, e.g., [7, 8] and for engineering applications [9]), which is the favorite language of the related scientific area of computational linguistics (called Natural Language Processing or, simply, NLP), this approach is very well known. There is a huge amount of results in NLP (see, e.g., [10-20]), but essentially all of them concerning the actual use of a computer give a preference to Prolog (see, e.g., [15-20]) and this author knows nothing about any attempt of transfer of this approach to Mathematica and computer algebra systems in general. Under these circumstances, in this paper we will present few elementary examples of application of NLP with Mathematica so that simple NL (natural language) sentences can be used as an input and further processed by Mathematica purely logically exactly as in the previous paper. Yet, the “translation” from the natural (human) language (even for a small fragment of English sentences) by no means is it a trivial task although we were able to make some progress.

One way of working is to try to transform NL sentences into logical formulae (commands), correctly understood by Mathematica, through their transformation by using standard Mathematica string commands. This rather naïve approach can be used in very elementary sentences and is not of sufficient generality. Therefore, the preferable approach is to use the already available Prolog programs in computational linguistics: NLP (more explicitly, in computational semantics), such as the elementary ones by Cooper, Lewin and Black [19] and the more general ones by Blackburn and Bos [20] through an appropriate call of Prolog from inside Mathematica so that the natural language sentences can be transformed into logical formulae and, next, to bring the Prolog output back to Mathematica for further processing. This approach has been successful in simple cases, but it requires a kind of interface between Mathematica and Prolog and back, which is non-trivial especially inside a completely automated working environment. Yet, with this approach we have been already able to work with simple NL sentences inside Mathematica.

The third (preferable) way of working consists in an attempt to prepare our own programs inside Mathematica, itself (and letting Mathematica do the rest). We have been moderately successful in our experiments so far and we present the related results in simple examples obtained from applied mechanics.

Of course, it should also be emphasized that Mathematica is much more powerful than Prolog in numerical and symbolic computations (but, surely, not in logic), graphics, interface facilities, Mathematica notebooks, etc. and, therefore, in general it cannot be substituted by Prolog in applied mechanics and mechanical engineering problems. Furthermore, it can also be mentioned that beyond ordinary words (such as determiners, nouns, pronouns, adjectives, verbs, adverbs, etc.) it is also convenient to use mathematical symbols in our applications (such as k for the constant of the helical spring S [6] above) as a separate category of words, yet quite similar to proper names in computational linguistics. In the future, it is hoped that the present results may prove useful for a more direct and simple communication between the user and Mathematica and, moreover, they can easily be extended to questions in natural language (already so popular with Prolog) and analogous replies by Mathematica, to discourses, etc.

The dream is to prepare an applied mechanics and/or mechanical engineering simple text (let’s say a paragraph of ordinary text) including mathematical symbols and elementary operations (such as arithmetic operations, differentiation, integration, etc.) in a completely natural (human) language and let Mathematica proceed with this text (after transforming it into its own language and syntax) and provide us with its conclusions (e.g. about the truth or the falsity of the statements in this text) and further computations (e.g. numerical, symbolic and logical calculations, equation solutions, etc.) automatically. Nevertheless, naturally, we are still far away from this dream and much work has to be done even for a small applied-mechanics-oriented fragment of English. Perhaps, after one decade, but we wish to believe in this dream and be optimists!

A THEOREMA COMPLETELY NATURAL PROOF

As a first (and the most elementary) application of Mathematica to NLP, we reconsider the fracture mechanics problem of propositional logic in the companion paper [5], but now in an attempt to improve the Theorema [2] assumptions and its natural proof from the NL point of view. More explicitly, the assumptions in this problem [5] include the implication sign (⇒), which is quite clear from the mathematical logical point of view, but not from the NL point of view. Similarly, the original automated proof of Theorema (not displayed here for the sake of space) is really perfect both from the logical point of view as well as from the text accompanying it and, therefore, completely clear. Yet, this proof includes the implication sign (⇒) as well. In this section, we will show that in this particular fracture mechanics problem, quite natural sentences (free from any mathematical sign) could be used in the assumptions and, after some more effort, in the Theorema proof itself. Therefore, the whole proof can essentially consist only of text. This approach will be demonstrated in brief in the present section.

At this point it should be emphasized that the Theorema version having been used here is the a-version. Much more improved and powerful versions will appear in the near future.

To the above-described aim, we decided to employ the word “causes” instead of the implication sign (⇒). Therefore, the modified, NL, Theorema assumptions (denoted by Ass1 to Ass4) get the following forms:

Assumptions[“Fracture NLP”, “excessive loading causes very high stress intensity factor” “Ass1” “very high stress intensity factor causes fracture” “Ass2”] “fracture causes replacement” “Ass3” “replacement causes cost and delay” “Ass4”
Next, what is of actual interest here, we have been able to transform the above assumptions into the standard format acceptable by _Theorema_. This has been made just by isolating both the subjects and the objects in the sentences constituting the above assumptions, replacing the verb "causes" by the implication sign (⇒), essentially the _Implies_ command in _Mathematica_ and the similar _Theorema_ command in _Theorema_. This can be rather easily done through string operations by using standard _Mathematica_ commands. The details (to be sincere of a somewhat technical nature) are displayed below:

\[ m = \text{Assumptions}["Fracture NLP"]\text{Length; Table}[a[i] = \text{Assumptions}["Fracture NLP"][4, i], \{i, m\}]; \]

\[ \text{sb1} = \text{Table}[b[i] \rightarrow a[i], \{i, m\}]; \]

\[ \text{sb2} = \text{Table}[b[i] \rightarrow a[i], \{i, m\}]; \]

Then _Theorema_ has been able (on the basis of both the _Grammar_ and the _Lexicon_) at first to prove the lemma that the phrase "the excessive loading" is a noun phrase, whereas the phrase "caused a direct yielding" is a verb phrase through the use of its predicate prover. Next, _Theorema_ also proved the _Proposition_ that the phrase (complete sentence) "the excessive loading caused a direct yielding" is really a sentence, based both on the grammar and on the above lemma. (The related details and proofs will not be displayed here.) Naturally, a much more detailed grammar and lexicon could, possibly, permit a more expanded variety of lemmata and propositions concerning natural language (from the syntactic point of view) to be proved, but it is also clear that the _Theorema_ possibilities (at least in its available -version) are limited in this task especially compared to those of _Prolog_.

A BEAM CONCLUSION WITH _THEOREMA_ / _PROLOG_

The standard computer language being used in NLP is surely _Prolog_. In this section, we will continue to use _Mathematica_ as our main computer language and _Theorema_ for our proofs, but we will also use _Prolog_ (in fact _SICStus Prolog_ ) for the derivation of the semantics of our elementary NL (natural language) sentences so that they can next be imported into _Theorema_. Our present four assumptions (_Ass1_ to _Ass4_ denoted as "Beam assumptions" to _Theorema_), concerning two separate beams in strength of materials, have as follows:

\[ \text{beam[beam1]} \ "\text{Ass1}" , \text{beam[beam2]} \ "\text{Ass2}" , \text{straight[beam1]} \ "\text{Ass3}" , \text{elastic[beam2]} \ "\text{Ass4}" , \]

i.e. both objects, _beam_1 and _beam_2, are beams and, moreover, _beam_1 is assumed to be _straight_, whereas _beam_2 to be elastic. Naturally, the predicate prover (_PredicateProver_) of _Theorema_ is easily able to prove conclusions on the basis of these assumptions such as the composite "Beam conclusion":

\[ \text{beam[beam1]} \ \land \ \text{beam[beam2]} \ \land \ \text{straight[beam1]} \ \land \ \text{elastic[beam2]} \]
Our aim here is simply to declare the same assumptions in a much more natural way such as “Beam assumptions NLP”

“beam1 is a beam” “Ass1”, “beam2 is a beam” “Ass2”, “beam1 is straight” “Ass3”, “beam2 is elastic” “Ass4”

From the typesetting point of view, obviously, this is direct, but, naturally, Theorema cannot understand NL (natural language). Therefore, we have to process these assumptions so that they can be transformed into the natural way of writing in Theorema (as well as in Mathematica). At first, we can isolate these NL assumptions through the Mathematica commands

\[
m = \text{Assumptions["Beam assumptions NLP"]}/\text{Length};
\]

\[
\text{Table}[a[i] \rightarrow \text{Assumptions["Beam assumptions NLP"]}, \{i, 4, 2\}, \{i, m\}]/\text{InputForm}
\]

Then we obtain the related list (with four elements \(a[i]\))

\[
\{\text{"beam1 is a beam"}, \ "beam2 is a beam", \
 \ "beam1 is straight", \ "beam2 is elastic"\}
\]

ready to be processed through Prolog (NLP: natural language processing). Surely, there are so many ready NLP Prolog programs, which be called from inside Mathematica with their outputs brought back to Mathematica by using appropriate input/output files. (The use of MathLink for this connection and collaboration between Mathematica and Prolog, in our case SICStus Prolog, is a more difficult task not having been undertaken so far, but it is based on the same principles, i.e. to ask Prolog to proceed to the NLP of our NL Mathematica sentences and send the non-NL results back to Mathematica.)

The Prolog program we have adopted in this task has been prepared by Cooper, Lewin and Black [19]. In this Prolog program, the employed operators are defined, next, the definitions and rules for well-formed logic formulae, terms, λ-terms and quantifiers, etc. are given, β-conversion and reduction are also undertaken and, finally, what is most important in our NLP problem, an elementary English grammar is defined (for simple phrases and sentences such as those in our above NLP problem, an elementary English grammar is defined). Theorema cannot understand NL (natural language). Therefore, we have to process these assumptions so that they can be transformed into the natural way of writing in Theorema (as well as in Mathematica). At first, we can isolate these NL assumptions through the Mathematica commands

\[
\text{sb1} = \text{Table}[a[i] \rightarrow b[i], \{i, m\}];
\]

\[
\text{sb2} = \text{Table}[b[i] \rightarrow a[i], \{i, m\}];
\]

The above substitutions: these lexical examples referring to a proper name (pn), a noun (n), an adjective (a), a transitive verb (tv) and the auxiliary verb av “is”, respectively. Additional words have also been introduced and, alternatively, an actual special-purpose lexicon for this task could be used. What seems to be of much importance here is the logical formula defining each word we introduced (frequently in λ-calculus form, such as X^beam(X) for a beam), which constitutes the basis for the transformation of the NL phrases into the corresponding logical forms.

In any case, Prolog and the above composite program have been easily able to transform our NL assumptions into their equivalent logical forms (through the appropriate use of related input and output files from Mathematica), e.g., \text{beam(beam1)}, \text{beam(beam2)}, \text{straight(beam1)} and \text{elastic(beam2)}, respectively. Finally, a quite technical detail (based on the Mathematica commands \text{StringReplace} and \text{ToExpression}) concerns the slight change of notation (mainly from parentheses to brackets, e.g. \text{beam[beam1]}) so that our assumptions are completely ready to be used in Mathematica and Theorema now and, in fact, they essentially coincide with the above-displayed forms really accepted by Theorema. The NL forms have been already denoted by \(a[i]\), whereas the corresponding logical forms are denoted by \(b[i]\). The related Mathematica final substitution commands are again

\[
\text{show} = \text{Table}[a[i] \rightarrow b[i], \{i, m\}];
\]

(Naturally, it is confessed that a little experience with Prolog and NLP in Prolog is required for the understanding of these rules! The interested reader can consult References [15, 16].)

Finally, the above Prolog program makes also extensive use of lexical definitions (with the predicate \text{lex} for lexicon) for the various categories of words used through their type (e.g. \text{pn} for proper names, \text{tv} for transitive verbs, etc.). Beyond these general definitions of lexical terms, we had also to introduce our own special word lexical definitions such as

\[
\text{lex(beam1, pn, beam1)}.
\]

\[
\text{lex(beam, n, X^beam(X)).}
\]

\[
\text{lex(isotropic, a, X^isotropic(X)).}
\]

\[
\text{lex(has, tv, Y^X^have(X, Y)).}
\]

\[
\text{lex(is, av, be)}
\]

and, therefore, the last step of the whole approach is to use these substitutions in the Theorema \text{Prove} command so that the NL assumptions can be inserted in it (but next transformed into their logical equivalents). The \text{Theorema} proof (naturally using the logical forms of the assumptions) can, next, be brought back to the NL form by using the \text{sb2} substitutions:

\[
\text{show} = \text{Prove[Proposition["Beam conclusion"], using -> Assumptions["Beam assumptions NLP"]/sb1, by -> PredicateProver, presentation -> \text{"SuccessBranch"}, in-notebook -> \text{"Separate"}/sb2]}
\]
This is the command actually having been used in Theorema and the sole indications of the NLP approach are the substitutions $\langle sb1 \rangle$ and $\langle sb2 \rangle$ in this command. Of course, a similar method can be used for the propositions to be proved, etc.

At this point, we can also add that beyond the above rather simple assumptions, we can also consider (and actually did) more complicated assumptions such as those including the usual quantifiers “for all” ($\forall$) and “exists” ($\exists$). For example, we may consider the two assumptions ($Ass1$ and $Ass2$)

\[ \text{“beam[beam1]” and “}\forall x \, (\text{beam}[x] \Rightarrow \text{elastic}[x])\]" wising to prove the simple conclusion Conclusion that

\[ \text{“elastic[beam1]”} \]

In this case, we have also used the quantifier “for all” ($\forall$) in the second of our assumptions and, naturally, Theorema has been actually and easily able to prove our conclusion Conclusion. But, perhaps, the NL-equivalent writing of these statements (both the assumptions and the conclusion)

\[ \text{“beam1 is a beam”, “every beam is elastic”, “beam1 is elastic”} \]

is sufficiently more natural, the elimination of the direct use of the quantifier $\forall$ taken also into account (in favor of the simple word “every”). Inversely, by providing the NL forms of our assumptions and conclusion, we have been able to derive their logical equivalents through the aforementioned Prolog program and, next, use them in Theorema (exactly in the way already described) in order to obtain a natural proof not only in the text, but also in the logical formulae used too, which are now exactly human sentences, e.g. “every beam is elastic”.

To become a little more concrete, in this way of working, we received the following Theorema proof (again by using the predicate prover) in completely NL (natural language):

Prove:

- (Conclusion) beam1 is elastic
- under the assumptions:
  - (Ass1) beam1 is a beam,
  - (Ass2) every beam is elastic.

For proving (Conclusion), by (Ass2), it suffices to prove

\[ \text{(2) beam1 is a beam.} \]

\[ \text{Formula (2) is true because it is identical to (Ass1).} \]

Naturally, this has been a simple proof, but much more complicated similar Theorema proofs can also be obtained.

A typical somewhat more complicated example concerns the case when we have both quantifiers ($\forall$ and $\exists$) inside the same formula. For example, following the same NL approach, we could simply give the assumption (or even the conclusion) instead of its logically equivalent (but more complex) form

\[ \forall x \, (\text{beam}[x] \Rightarrow \exists y \, (\text{end}[y] \land \text{has}[x, y])) \]

which is logically correct, but, undoubtedly, not so easy for the user of Mathematica and/or Theorema (e.g. the engineer) to declare. Therefore, the first writing seems to be preferable, the task of transforming it into the second one left to Prolog and the related Mathematica interface as was already explained.

It is also understood that the present approach is of limited applicability so far and generalizations to much more complicated cases are indispensable and, possibly, they will be undertaken in the future. Finally, it is clear that transforming NL sentences into their logical equivalents in Mathematica [1] is not of an exclusive importance for Theorema [2], but it can also be used in any Mathematica logical computations such as those in the companion paper [5], partially based on Maeder’s Prolog interpreter [3] in Mathematica and not in Theorema.

In the next section, we will very briefly illustrate the possibility of performing the task of transformation of NL phrases/sentences into their logical equivalents inside Mathematica (i.e. without any resort to Prolog), but this has been achieved-implemented only in few simple cases so far.

**SIMPLE NLP INSIDE MATHEMATICA**

In this section, we will transfer to Mathematica few of the NLP well-known methods traditionally used in Prolog. At first, we can mention that Mathematica offers the not so popular command Function (with an arguments list and a body), which can be used for the logical equivalents of the lexicon words and the further work in computational semantics for phrases and sentences (just as in the popular $\lambda$-calculus).

Beginning with the lexicon, a first possibility is to use the new command Lex (with two arguments), e.g.

\[
\begin{align*}
\text{Lex[beam1]} &= \{\text{pn[beam1]}, \{\}\}; \\
\text{Lex[beam]} &= \{\text{n[beam]}, \text{Function}[X, \text{beam}[X]]\}; \\
\text{Lex[straight]} &= \{\text{[a[straight]], \text{Function}[X, \text{straight}[x]]}\}; \\
\text{Lex[has]} &= \{\text{[v[has]], \text{Function}[X, \text{have}[X]]}\}; \\
\text{Lex[is]} &= \{\text{[cop[is]], \text{Function}[X, X]}\}; \\
\text{Lex[every]} &= \{\text{[d[every]], \\
\quad \text{\quad \quad Function[P, \text{Function}[Q, \forall x, (P[X] \Rightarrow Q[x])]])}; \\
\text{Lex[one]} &= \{\text{[d[one]], \\
\quad \quad \text{\quad \quad Function[P, \text{Function}[Q, \exists y, (P[X] \land Q[x])])]}\}; \\
\end{align*}
\]

From this short sample of lexical terms (words), at first we observe that these are classified into categories, i.e. as proper names (pn), nouns (n), adjectives (a), verbs (v), the copula (cop), determiners (d), etc. (e.g. pronouns not illustrated here). What are also very important are the logical expressions of these words, generally including the command Function although we could also have used the symbol $\lambda$ instead (for lambda expressions) through the command

\[ \lambda x, y \Rightarrow \text{Function}[x, y] \]
but, evidently, this is of minor importance.

In engineering texts, it is also clear that instead of proper names (pn), it is preferable to use a new category of objects: mathematical symbols (ms), e.g. beam1 above (for a beam), L (for the length of the beam), EI (for its stiffness), y (for its deflection), etc., etc. What should also not be ignored is the special interpretation of the copula (is), which essentially means that this is ignored (through the identity function defining it), e.g. the phrase “beam1 is elastic” is logically transformed into “elastic[beam1]” (with the verb “is” not appearing in it). It can also be mentioned that the definitions of the determiners “every” and “one” are somewhat complicated (with the quantifiers “for all” and “exists” appearing in them plus the implication and conjunction symbols, respectively). We can also add that a simpler way of working is through lists of words:

Adjectives = {anisotropic, clamped, curvilinear, dynamic, elastic, fixed, free, isotropic, long, static, straight, vertical};

e etc. etc. and proceeding to the logical definition of these words (here adjectives, a) in a uniform way inside a Lex module, e.g.

If[MemberQ[Adjectives, w], Return[{a[w, x[X, w[X]]]};

We have also prepared simple (although not complete and Prolog-competitive) Mathematica modules (for the semantic analysis of noun phrases, verb phrases and sentences), which can be used both for parsing and for semantics of simple NL phrases and whole sentences so that both their structure and their logical equivalent can be derived. For example, for the simple sentence “every beam has one end”, we obtained

{sn[np[d(every), n[beam]], vp[tv[has], np[d(one), n[end]]], ∃, Implies[beam[x], 3 y (end[y] && has[x, y]]};

with the first line above referring to the syntactic analysis of this sentence and the second line to its meaning, which, it is believed, is not trivial to write down manually especially in more complicated cases (e.g. with more than two quantifiers).

CONCLUSIONS

Concluding, we feel that the present results may be of some mechanical engineering interest and possible applicability especially when whole technical paragraphs (including text, symbols and formulae) are to be automatically logically analyzed (and, hopefully, proved syntactically meaningful and both logically and computationally correct) with the aid of the computer. Naturally, such an integrated task is not expected to take place in the near future in spite of the present and (we believe) moderately encouraging preliminary results.

ACKNOWLEDGMENTS

Again the author is extremely thankful to Professor Bruno Buchberger and the Theorema Group for their having given him the opportunity to use Theorema in its preliminary, α-version.

REFERENCES