Proceedings of the ASME – Greek Section,
First National Conference on
Recent Advances in Mechanical Engineering
September 17-20, 2001, Patras, Greece

ANG1/P094

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

Applied mechanics. Natural language processing. Computational semantics. *Mathematica*. *Theorema*. *Prolog*.

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 either by using Maeder's *Prolog* interpreter or, better, directly with standard *Mathematica* commands. Naturally, we always need to have a lexicon, a grammar and a parser, to use λ-calculus, etc. (exactly as in *Prolog*). 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 *Theorema* 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"

(Unfortunately, the typing of the above *Theorema* command does not exactly correspond to the real *Theorema* input.)

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 **™Implies** 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 = Assumptions["Fracture NLP"][[4]]//Length;
Table[a[i] = Assumptions["Fracture NLP"][[4,i,2]], {i,m}];
sp = Table[StringPosition[a[i], "causes"][[1]], {i, m}];
s1 = Table[StringTake[a[i], sp[[i, 1]] - 2, {i, m}];
s2 = Table[StringTake[a[i], {sp[[i, 2]] + 2,
     StringLength[a[i]]}], {i, m}];
Table[b[i] = ™Implies[s1[[i]], s2[[i]], {i, m}];
sb1 = Table[a[i] -> b[i], {i, m}];
sb2 = Table[b[i] -> a[i], {i, m}];
```

We can add that the last two commands constitute just substitutions from the original, NL (natural language) present notation of the assumptions to the *Mathematica/Theorema* standard notation. This is necessary for the transformation of our NL assumptions into the *Mathematical/Theorema* mathematical notation (the **sb1** substitutions) so that the *Theorema* proof can be achieved in the ordinary, formal, non-NL way and back (the **sb2** substitutions) so that the same proof can contain NL sentences (essentially the above original, NL assumptions) instead of ordinary logical formulae (i.e. without the implication sign in favor of the ordinary verb **"causes"**). The derived proof is rather interesting from the NLP point of view, but (for the sake of space) it will not be displayed here.

A SYNTACTIC PROOF WITH *THEOREMA*

A sufficiently different possible use of *Theorema* is in the syntactic analysis of relatively simple NL (natural language) sentences. We considered the extremely small fragment of English consisting of the following three *Theorema* elementary assumptions, denoted as **Grammar** (with the symbols **p1**, **p2** and **p3** in them referring to sequences inside lists)

$$(\mathbf{NounPhrase}[\langle p1\rangle, \langle p2\rangle] \wedge \mathbf{VerbPhrase}[\langle p2\rangle, \langle p3\rangle]) \Rightarrow \mathbf{Sentence}[\langle p1\rangle, \langle p3\rangle] \qquad \text{"SD"}$$

$$(\mathbf{Determiner}[w1] \wedge \mathbf{Adjective}[w2] \wedge \mathbf{Noun}[w3]) \Rightarrow \mathbf{NounPhrase}[\langle w1, w2, w3, p1\rangle, \langle p1\rangle] \qquad \text{"NPD"}$$

$$(\mathbf{Verb}[w1] \wedge \mathbf{NounPhrase}[\langle p1\rangle, \langle p2\rangle]) \Rightarrow \mathbf{VerbPhrase}[\langle w1, p1\rangle, \langle p2\rangle] \qquad \text{"VPD"}$$

which express the well-known syntactic rules that (i) a noun phrase and a verb phrase (in this order) constitute a sentence, (ii) a determiner, an adjective and a noun constitute a noun phrase, and (iii) a verb and a noun phrase constitute a verb phrase. (Obviously, there are many more ways to form noun phrases, verb phrases and sentences beyond the above ones!)

Beyond these assumptions (the rules of our grammar for an extremely small fragment of sentences), we must also have a lexicon, denoted as **Lexicon** in our present session. The adopted (very small and so incomplete lexicon) has been

Determiner[the]	**"the"**
Determiner[a]	**"a"**
Noun[loading]	**"loading"**
Noun[yielding]	**"yielding"**
Adjective[excessive]	**"excessive"**
Adjective[direct]	**"direct"**
Verb[caused]	**"caused"**

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 *a*-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:

beam[beam1]	**"Ass1",**	**beam[beam2]**	**"Ass2",**
straight[beam1]	**"Ass3",**	**elastic[beam2]**	**"Ass4",**

i.e. both objects, **beam1** and **beam2,** are beams and, moreover, **beam1** is assumed to be **straight**, whereas **beam2** 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"**

$$\mathbf{beam}[beam1] \wedge \mathbf{beam}[beam2] \wedge \mathbf{straight}[beam1] \wedge \mathbf{elastic}[beam2]$$

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 = Assumptions["Beam assumptions NLP"][[4]]//Length;
Table[a[i] = Assumptions["Beam assumptions NLP"]
        [[4, i, 2]], {i, m}]//InputForm
```

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

{"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 assumptions). For example, the *Prolog* rule for a sentence is

```
sn(NP*VP)  - ->  np(NP), vp(VP).
```

which simply means that a noun phrase (**np**) **NP** and a verb phrase (**vp**) **VP** constitute a sentence. Other similar rules explain that a determiner **d** and a noun **n** constitute a noun phrase **np**, etc. The determiners **"every"** and **"a"** (or **"an"**) are somewhat special using the predicates **forall** and **exists**, respectively, in their definitions and having the following *Prolog*-rule forms:

```
d(Q^P^forall(X, Q*X  - - ->  P*X))  - ->  [every].
d(Q^P^exists(X, Q*X  &  P*X))  - ->  [a]; [an].
```

(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 **lex** for lexicon) for the various categories of words used through their type (e.g. **pn** for proper names, **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

```
lex(beam1, pn, beam1).
lex(beam, n, X^beam(X)).
lex(isotropic, a, X^isotropic(X)).
lex(has, tv, Y^X^have(X, Y)).
lex(is, av, be)
```

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., **beam(beam1)**, **beam(beam2), straight(beam1)** and **elastic(beam2),** respectively. Finally, a quite technical detail (based on the *Mathematica* commands **StringReplace** and **ToExpession)** concerns the slight change of notation (mainly from parentheses to brackets, e.g. **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

```
sb1 = Table[a[i] -> b[i], {i, m}];
sb2 = Table[b[i] -> a[i], {i, m}];
```

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

```
Show[Prove[Proposition["Beam conclusion"],
  using -> Assumptions["Beam assumptions NLP"]/.sb1,
  by -> PredicateProver, presentation -> "SuccessBranch",
  in-notebook -> "Separate"/.sb2]
```

This is the command actually having been used in *Theorema* and the sole indications of the NLP approach are the substitutions (**/.sb1** and **/.sb2**) 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" (∀) and "exists" (∃). For example, we may consider the two assumptions (**Ass1** and **Ass2**)

"beam[beam1]" and **"$\forall_x$ (beam[x] ⇒ elastic[x])"**

wishing to prove the simple conclusion **Conclusion** that

"elastic[beam1]"

In this case, we have also used the quantifier "for all" (∀) 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)

"beam1 is a beam", **"every beam is elastic",**
"beam1 is elastic"

is sufficiently more natural, the elimination of the direct use of the quantifier ∀ 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
(2) beam1 is a beam.
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 (∀ and ∃) inside the same formula. For example, following the same NL approach, we could simply give the assumption (or even the conclusion)

"every beam has an end"

instead of its logically equivalent (but more complex) form

$\forall_x$ (beam[x] ⇒ $\exists_y$ (end[y] ∧ 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 λ-calculus).

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

Lex[beam1] = {pn[beam1], {}};
Lex[beam] = {n[beam], Function[X, beam[X]]};
Lex[straight] = {a[straight], Function[X, straight[x]]};
Lex[has] = {v[has], Function[X, have[X]]};
Lex[is] = {cop[is], Function[X, X]};
Lex[every] = {d[every],
Function[P, Function[Q, $\forall_x$ (P[X] ⇒ Q[x])]]};
Lex[one] = {d[one],
Function[P, Function[Q, $\exists_x$ (P[X] ∧ Q[x])]]};

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 λ instead (for lambda expressions) through the command

λ[x_, y_] := 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};

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], λ[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 logic 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]]]],
$\forall_x$ Implies[beam[x], $\exists_y$ (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

[1] Wolfram, S., 1999, *The Mathematica Book,* 4th edition (*Mathematica*, version 4), Wolfram Media, Champaign, IL, and Cambridge University Press, Cambridge, UK.

[2] Buchberger, B. *et al.* 1997, "A Survey on the *Theorema* Project", *Proceeding of the International Symposium on Symbolic and Algebraic Computation (ISSAC) '97*, Maui, HI.

[3] Maeder, R. E., 1996, *The Mathematica Programmer II*, Academic Press, San Diego, CA, Chapter 2, pp. 21-56.

[4] Ioakimidis, N. I., 1998, "Elementary Engineering Mechanics Applications of the *OTTER* Automated Reasoning System", *Proceedings of the 5th National Congress on Mechanics* (ed. by P. S. Theocaris, D. I. Fotiadis and C. V. Massalas), The University of Ioannina Press, Vol. 2, pp. 759-766.

[5] Ioakimidis, N. I., 2001, "*Mathematica*-Based Formula Verification in Applied Mechanics", Proceedings of the ASME – Greek Section: First National Conference on *Recent Advances in Mechanical Engineering*, Sep. 2001, Patras, Greece.

[6] Dimarogonas, A. D., 1989, *Computer Aided Machine Design*, Prentice Hall, New York.

[7] Sterling, L. and Shapiro, E., 1994, *The Art of Prolog: Advanced Programming Techniques*, 2nd ed., The MIT Press, Cambridge, MA.

[8] Bratko, I., 1990, *Prolog Programming for Artificial Intelligence*, 2nd ed., Addison-Wesley, Harlow, England.

[9] Syrmakezis, C. A. and Mikroudis, G. K., 1991, *Application of Expert Systems to Problems of Structures* (in Greek), Technical Chamber of Greece, Athens.

[10] Partee, B. H., ter Meulen, A. and Wall, R. E., 1993, *Mathematical Methods in Linguistics*, Kluwer, Dordrecht.

[11] Carpenter, B., 1997, *Type-Logical Semantics*, The MIT Press, Cambridge, MA.

[12] Heim, I. and Kratzer, A., 1998, *Semantics in Generative Grammar*, Blackwell, Malden, MA.

[13] Ralli, A., Grigoriadou, M., Philokyprou, G., Christodoulakis, D. and Galiotou, E., 1997, *Working Papers in Natural Language Processing* (in English), Diavlos, Athens.

[14] Allen, J., 1995, *Natural Language Understanding*, 2nd ed., Benjamin/Cummings, Redwood City, CA.

[15] Pereira, F. C. N. and Shieber, S. M., 1987, *Prolog and Natural-Language Analysis*, Center for the Study of Language and Information (CSLI), Menlo Park, CA.

[16] Matthews, C., 1998, *An Introduction to Natural Language Processing Through Prolog*, Longman, London.

[17] Dougherty, R. C., 1994, *Natural Language Computing: An English Generative Grammar in Prolog*, Lawrence Erlbaum Associates, Hillsdale, NJ.

[18] Dik, S. C., 1992, *Functional Grammar in Prolog: An Integrated Implementation for English, French and Dutch*, Mouton de Gruyter, Berlin.

[19] Cooper, R., Lewin, I. and Black, A. W., 1992-93, *Prolog and Natural Language Semantics: Notes for AI3/4 Computational Semantics* (available on the Web).

[20] Blackburn, P. and Bos, J., 1999, *Representation and Inference for Natural Language: A First Course in Computational Semantics*, Saarbruecken (available on the Web).