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

ANG1/P093

MATHEMATICA-BASED FORMULA VERIFICATION 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

Mathematica is a modern and powerful computer algebra system offering all types of computational facilities (symbolic, numerical and graphical) to its user in an integrated environment. Therefore, it has been repeatedly used in mechanical engineering applications. In this paper, we will show that *Mathematica* can also be useful for formula verification (either logical or algebraic) by employing either its own internal commands or extensions of these commands such as Maeder's *Prolog* interpreter in *Mathematica* and, recently, Buchberger's *Theorema*, based also on *Mathematica*. External reasoning systems (such as *OTTER*) can also be called from *Mathematica*. The engineering applications of this paper are obtained from applied mechanics and illustrate these not so well known possibilities of *Mathematica* related to formula verification.

KEYWORDS

Applied mechanics. Formula verification. Logical computations. Symbolic computations. *Mathematica*.

INTRODUCTION

Computer algebra systems have played (and still play) a very important role in applied mechanics and mechanical engineering long ago. For example, such systems have been successfully used for the computation of stiffness matrices in finite element analysis since about thirty years. Today, it seems that the most modern and powerful computer algebra system, offering also efficient tools for numerical computations and graphics, is Wolfram's *Mathematica* [1], having been originally released by Wolfram Research, Inc. in 1988. Naturally, as is extremely well known, *Mathematica* offers a large variety of very carefully prepared powerful commands and a strong programming language and, therefore, it has been already employed for the solution of many mechanical engineering problems appearing in everyday practice and research.

In this paper, we will try to show the usefulness of the *Mathematica* integrated computational environment in formula verification, i.e. in the derivation of conclusions about the validity of a formula concerning an equation/inequality, a logical formula or more general formulae. Perhaps, the prototype in such questions (about the truth or the falsity of a formula) is to ask whether **1 + 1 == 2** (in numerical computations) or **a == a** (in symbolic computations). Naturally, the *Mathematica* reply to both of these commands (essentially questions) is **True**. Undoubtedly, these are trivial examples of formula verification, but much more complicated formulae, arising in more difficult problems, can be tested and verified as being true or false. It can also be mentioned that beyond equational formulae (such as the above ones) and analogous inequational formulae, one can also consider logical formulae such as $\mathbf{A \wedge B \Rightarrow A}$ or $\mathbf{A \vee \neg A \Rightarrow True}$ (the symbols $\wedge$, $\vee$ and $\neg$ denoting conjunction, disjunction and negation, respectively), which are also true, and combinations of algebraic and logical formulae.

Much more important work has been already made in *Mathematica* with respect to theorem proving through the extremely powerful *Theorema* system by Buchberger and the *Theorema* Group [2] with respect to theorem proving with *Mathematica*. Moreover, Maeder [3] prepared a *Prolog* interpreter in *Mathematica* so that the logical capabilities of *Prolog* can be (partially) transferred to the *Mathematica* mainly algebraic environment and be directly available there.

The aim of the present paper is to show the usefulness of *Mathematica* as well as of *Mathematica*-based packages and programs (which can be called from *Mathematica*) for the direct, black-box verification of formulae arising in applied mechanics and mechanical engineering of either a logical or an algebraic nature. The term verification means that no analytical proofs are provided (contrary to what is the case with *Theorema* and exactly to what is the case with *Prolog* and Maeder's *Prolog* emulation in *Mathematica*), but the truth or the falsity of the formula under consideration is completely studied and the correct conclusion is drawn. In general, this is sufficient for the mechanical engineer.

Among the problems of applied mechanics and mechanical engineering that could be considered (through appropriate formulae verification) we can mention:

1. A conclusion that excessive loading causes cost and delay drawn on the basis of four axioms such as **Fracture ⇒ Replacement** and the data **ExcessiveLoading**.

2. Conclusions about the appropriate numerical method in the boundary (singular) integral equation method for the solution of crack problems in plane elasticity and fracture mechanics and the determination of stress intensity factors (SIFs) at the crack tips, naturally this method depending on the type of the crack (interior, edge or interface) and the extrapolation approach to be used (polynomial or natural) if any.

3. Conclusions about the conditions (deflection, slope, bending moment and shear force) at a simple beam end *B* derived on the basis of available conditions at the other beam end *A* and the related valid assumptions in beam problems.

4. Conclusions concerning the motion of a particle on a circular contour (velocity and acceleration).

5. A conclusion about the lack-of-contact conditions between the two faces of a penny-shaped crack under a normal loading in three-dimensional elasticity and contact mechanics.

6. A conclusion about an inequality constraint for the temperature distribution (the temperature is required to nowhere exceed an upper bound) along a bar under a constant rate of heat generation in heat transfer.

For the sake of space, only the first four of the above six applied mechanics conclusions will be studied in this paper.

In the first three of the above conclusions, we will use in *Mathematica* (or through *Mathematica*) logical-type tools such as the simple *Mathematica* **LogicalExpand** command and Maeder's *Prolog* emulation in *Mathematica*. Next, in the fourth of the above conclusions, we will use an advanced algebraic tool: Buchberger's Groebner bases also offered by *Mathematica*. Of course, further extensions to even more complicated, non-trivial applied mechanics and, in general, mechanical engineering problems are also completely possible.

It seems that the present results (formal verification of formulae either of a logical or of an algebraic nature through *Mathematica*) may prove to be of interest in applied mechanics and, more generally, in mechanical engineering and, therefore, they may offer a new area of applicability of *Mathematica* to engineering practice, which seems not having been sufficiently investigated so far contrary to its symbolic, numerical and graphical capabilities, which have already proved so useful in engineering computations. Of course, although Maeder' *Prolog* [3] has been actually used in theorem proving since 1994 by Maeder himself, a further significant extension of this possibility will become available as soon as the already mentioned Buchberger's *Theorema* system [2] (a *Mathematica* package capable to derive formal proofs of valid formulae in natural language, compared to black-box verifications only in *Prolog* and in the present approach too) be released (in final, commercial form) and this is expected to take place during 2002 or 2003.

A FRACTURE MECHANICS PROBLEM

We begin with a rather simple problem in mechanical engineering involving logic. This is the problem of a specimen with a crack, where fracture can appear. This problem was originally considered by Ioakimidis [4] by using the *OTTER* automated deduction system for the proof of a related conclusion. Here we will use *Mathematica* [1] instead for the verification of this and similar conclusions. We assume that the following reasonable premises hold true:

1. An excessive loading causes the appearance of a very high stress intensity factor (SIF) at the crack tip.

2. Such a very high stress intensity factor causes the fracture of the specimen.

3. The fracture of the specimen has as a consequence its replacement.

4. The replacement of the specimen has as a consequence a related cost as well as the delay in the actual possible use of the specimen in production.

These assumptions can easily be declared to *Mathematica* through the following simple commands:

p1 = ExcessiveLoading ⇒ VeryHighStressIntensityFactor;
p2 = VeryHighStressIntensityFactor ⇒ Fracture;
p3 = Fracture ⇒ Replacement;
p4 = Replacement ⇒ CostAndDelay;

Let us also assume that we really have an excessive loading applied to the specimen (at a particular moment), i.e.

p0 = ExcessiveLoading;

Then we wish to verify that we will also have cost and delay under the above assumptions, i.e. that the following conclusion will hold true:

c0 = CostAndDelay;

To this end we should give the verification command

r0 = (p0 ∧ p1 ∧ p2 ∧ p3 ∧ p4) ⇒ c0

Unfortunately, the *Mathematica* output is not satisfactory simply since the standard command **LogicalExpand** should also be used. Therefore, after the logical expansion of **r0**

LogicalExpand[r0]

we get **True** and this completes the verification of the formula. We have thus verified that in our specimen excessive loading causes cost and delay. In an analogous manner, we have also verified that our assumptions **p1** to **p4** imply the conclusion

c1 = ExcessiveLoading ⇒ CostAndDelay;

and, naturally, this is also obvious. The related command is

r1 = LogicalExpand[(p1 ∧ p2 ∧ p3 ∧ p4) ⇒ c1]

Finally, there is also the refutational way of verification. For example, the conclusion **r1** can also be verified through the related refutational command

r1 = LogicalExpand[p1 ∧ p2 ∧ p3 ∧ p4 ∧ ¬c1]

Naturally, the related *Mathematica* output is now **False** instead of **True** before. It is also obvious that any attempt to logically verify incorrect conclusions (on the basis of the above assumptions: **p1** to **p4**) failed. Therefore, it is clear that *Mathematica* can verify not only algebraic formulae, but also logical conclusions on the basis of its own command **LogicalExpand**.

Beyond the **LogicalExpand** command, we can also work with *Prolog* emulation in *Mathematica*. The related package has been prepared by Maeder [3]. This seems to be a significant enhancement of *Mathematica* with respect to logical conclusions. For this task at first we have to load the three related packages of Maeder: **Unify.m, Lisp.m** and **LogicProgramming.m** (in this particular order). Next, we can insert our already-mentioned assumptions (**p1**, **p2**, **p3** and **p4**) as well as our data (**p0**) about the present fracture problem. This can be easily done through the special **Assert** command:

Assert[VeryHighStressIntensityFactor, ExcessiveLoading]
Assert[Fracture, VeryHighStressIntensityFactor]
Assert[Replacement, Fracture]
Assert[CostAndDelay, Replacement]
Assert[ExcessiveLoading]

respectively. Now we are ready to ask our queries to Maeder's *Prolog* interpreter in *Mathematica*, e.g.

Query[ExcessiveLoading]
Query[CostAndDelay]

the reply being (in both cases) **Yes**. More explicitly, the first query is trivial because of our identical assertion, whereas for the second query *Mathematica* and its *Prolog* interpreter took also into account all of the above assertions. A somewhat more complex command to the *Prolog* interpreter could be

Map[Query, {ExcessiveLoading, VeryHighStressIntensity-Factor, Fracture, Replacement, CostAndDelay}]

with a list of five **Yes** in the reply because of all of our above assumptions and data. Additional questions can also be put to the *Prolog* interpreter (through the **Query** command) on the basis of the assumptions and data available (being introduced through the **Assert** command and removed through the **Retract** command). In this (alternative) way of working, we can again verify the validity (truth) of a formula (reply **Yes**) or show the lack of validity (falsity) of such a formula (reply **No**). (Incidentally, the command **Map** permits us to check the validity of a set of formulae through just one command.)

Now we will proceed to one more application of formula verification from the areas of numerical analysis and fracture mechanics by using (again) *Mathematica*.

THE APPROPRIATE NUMERICAL METHOD IN A CRACK PROBLEM

This problem concerns the selection of the appropriate method for the numerical solution of the boundary (singular) integral equation of a plane elasticity crack problem, to which it has been reduced, and it has been already studied by the author [4] by using the *OTTER* automated deduction system.

The following abbreviations are used: (i) From the physical point of view, we assume we may have either an internal crack (**IC**) or an edge crack (**EC**) or even an interface crack (**FC**). (ii) From the numerical analysis point of view, we can use either Gauss-type rules (more explicitly, the Gauss-Chebyshev rule, **GC**, or the modified Gauss-Legendre rule, **MGL**, or the Gauss-Jacobi rule, **GJ**) or Lobatto-type rules (more explicitly the Lobatto-Chebyshev rule, **LC**, or the modified Lobatto-Legendre rule, **MLL**, or the Lobatto-Jacobi rule, **LJ**). The Chebyshev-type rules are convenient only for ICs, the modified Legendre-type rules only for **EC**s and the Jacobi-type rules only for **FC**s. Moreover, the Gauss-type rules are used when we wish to compute the stress-intensity factors (SIFs) at the crack tips with either a polynomial extrapolation (**PE**) or a natural extrapolation (**NE**). In the case we do not wish to use extrapolation techniques (no extrapolation, **NO**) for the computation of the SIFs, then a Lobatto-type rule should be used. This situation can easily be declared to *Mathematica*:

{p1 = IC ⇒ (GC ∨ LC) ∧ ¬(MGL ∨ MLL) ∧ ¬(GJ ∨ LJ),
p2 = EC ⇒ ¬(GC ∨ LC) ∧ (MGL ∨ MLL) ∧ ¬(GJ ∨ LJ),
p3 = FC ⇒ ¬(GC ∨ LC) ∧ ¬(MGL ∨ MLL) ∧ (GJ ∨ LJ)};

(as far as the type of crack is concerned) and

{p4 = PE ∨ NE
⇒ (GC ∨ MGL ∨ GJ) ∧ ¬(LC ∨ MLL ∨ LJ),

p5 = NO ⇒ ¬(GC ∨ MGL ∨ GJ) ∧ (LC ∨ MLL ∨ LJ)};

(as far as the possibility of extrapolation is concerned).

We are now ready to proceed to verifications concerning the appropriate quadrature rule in the case of an assumed type of crack and an extrapolation/no-extrapolation method for the computation of the SIFs. For example, we can consider the validity of the conclusion

c1a = FC ∧ (PE ∨ NE) ⇒ GJ

i.e. the Gauss-Jacobi (**GJ**) quadrature rule is the appropriate one in the case of an interface crack (**FC**) together with the use of either a polynomial extrapolation (**PE**) or a natural extrapolation (**NE**) method for the computation of the SIFs at the crack tips. Naturally, this conclusion, assumed correct, cannot be verified by itself, but it can, possibly, be verified by using our above assumptions **p1** to **p5** already known to *Mathematica*. This can be easily done through the simple command

r1a = p1 ∧ p2 ∧ p3 ∧ p4 ∧ p5 ⇒ c1a//LogicalExpand

and we get the expected result **True** under our assumptions. Therefore, our conclusion has been verified to be correct. Alternatively, but equivalently, we could have written

r1b = FC ∧ (PE ∨ NE) ∧ p1 ∧ p2 ∧ p3 ∧ p4 ∧ p5 ⇒ GJ
//LogicalExpand

getting again the expected verification result **True**.

In a similar way, we can verify the validity of the conclusion that in the case of an interface crack (**FC**), but now no extrapolation (**NO**) for the computation of the SIFs, the Lobatto-Jacobi quadrature rule is the appropriate one. To this end, we can just write the *Mathematica* command

r1c = FC ∧ NO ∧ p1 ∧ p2 ∧ p3 ∧ p4 ∧ p5 ⇒ LJ
//LogicalExpand

with the result **True** again. Therefore, this conclusion is also a correct one (always on the basis of our original assumptions).

Assuming that we have loaded Maeder's *Prolog* interpreter packages (**Unify.m, Lisp.m** and **LogicProgramming.m**) exactly as we did in the previous application, we can also use the Maeder *Prolog* interpreter in *Mathematica* again. In this way, of working, it is very convenient to introduce the *Prolog* predicates **CrackAppropriate** and **ExtrapolationAppropriate** (with obvious meanings) as follows:

{Assert[CrackAppropiate[GC], IC],
Assert[CrackAppropiate[LC], IC],
Assert[CrackAppropiate[MGL], EC],
Assert[CrackAppropiate[MLL], EC],
Assert[CrackAppropiate[GJ], FC],
Assert[CrackAppropiate[LJ], FC]};

{Assert[ExtrapolationAppropriate[GC], PE ∨ NE],
Assert[ExtrapolationAppropriate[MGL], PE ∨ NE],
Assert[ExtrapolationAppropriate[GJ], PE ∨ NE]};

{Assert[ExtrapolationAppropriate[LC], NO]
Assert[ExtrapolationAppropriate[MLL], NO],
Assert[ExtrapolationAppropriate[LJ], NO]};

Naturally, all of the above assertions represent our aforementioned theoretical hypotheses about the appropriate quadrature rule in each particular case of a crack and an extrapolation/no-extrapolation method used. We can now introduce one more assertion through the logical rule

Assert[Appropriate[rule_], CrackAppropiate[rule_]
∧ ExtrapolationAppropriate[rule_]]

where the new predicate **Appropriate** has been introduced. This assertion means that when a quadrature rule (denoted by **rule_** in this *Mathematica* assertion) is simultaneously appropriate for the type of the crack and for the type of extrapolation/no-extrapolation method, then it is definitively appropriate for the solution of the boundary (singular) integral equation and the computation of the SIFs at the crack tips.

Beyond the above assertions, which concern our fundamental theoretical results about the appropriateness (and, in *Prolog*, implicitly, the lack of appropriateness) of numerical integration rules, we should also declare our particular data for a concrete crack problem to be solved. For example, we have

{Assert[FC], Assert[NE]};

for an interface crack and with the natural extrapolation formula to be used for the computation of the SIFs. Then we can ask the *Mathematica*-based Maeder's *Prolog* interpreter that we use several queries such as

Map{Query, Map[CrackAppropriate,
{GC, LC, MGL, MLL, GJ, LJ}]]

with *Mathematica*'s reply

{No, No, No, No, Yes, Yes}

revealing that only the Gauss-Jacobi (**GJ**) and the Lobatto-Jacobi (**LJ**) quadrature rules are crack-appropriate in our case (of an interface crack). Similarly, through an analogous command, *Mathematica* finds that all three **GC**, **MGL** and **GJ** quadrature rules are extrapolation-appropriate (reply **Yes**) in our case of natural extrapolation, the other three quadrature rules (**LC**, **MLL** and **LJ**) being inappropriate (reply **No**).

Finally, through the more specialized single command

Query[Appropriate[GJ]]

we get the reply **Yes** and, therefore, we have verified through *Mathematica* (plus its *Prolog* emulation package by Maeder) that really the Gauss-Jacobi (**GJ**) quadrature rule is an appropriate rule in our case of an interface crack and natural extrapolation. (For all other rules the reply would be **No**). Alternatively, we can also use the more general command

QueryAll[Appropriate[rule_]]

with the answer revealing that only the Gauss-Jacobi (**GJ**) quadrature rule is an appropriate rule.

The conclusion is that with the help of *Mathematica* either directly or with the further help of its Maeder *Prolog* interpreter we can be sure about the quadrature rule to be used in each particular case and, next, we can safely proceed with the numerical solution and the SIFs computation. We will now proceed to a third application concerning a beam problem.

A SIMPLE BEAM PROBLEM

We consider the problem of an ordinary beam with our interest in the conditions at its two ends. We will use again Maeder's *Prolog* interpreter in *Mathematica*.

At first, we will introduce the predicates **BeamEnd** and **BeamEnds**, for which the following three assertions hold true:

Assert[BeamEnd[a_], BeamEnds[a_, b_]]
Assert[BeamEnd[b_], BeamEnds[a_, b_]]
Assert[Fixed[b_], BeamEnds[a_, b_] ∧ Free[a_]]

The first two of the above assertions are trivial for a human, but not for a computer program such as *Mathematica*. The third assertion is less trivial: it simply states that if one end of the beam is free, the other end is fixed (clamped). Perhaps, a little more knowledge from applied mechanics is required for the following also elementary assertions for the deflection, the slope, the bending moment and the shear force at a beam end:

Assert[Deflection[a_, 0],
BeamEnd[a_] ∧ (Fixed[a_] ∨ SimplySupported[a_])]
Assert[Slope[a_, 0], BeamEnd[a_] ∧ Fixed[a_]]
Assert[BendingMoment[a_, 0],
BeamEnd[a_] ∧ (SimplySupported[a_] ∨ Free[a_])]
Assert[ShearForce[a_, 0], BeamEnd[a_] ∧ Free[a_]]

The meanings of the above four new predicates seem to be obvious from their names as well as the meanings of all four above assertions for a beam end.

Let us proceed now to a concrete example. We assume that we have a beam with two ends, **a1** and **b1**, and we know that the first of them is free. Then we have the following two additional assertions for our present special beam problem:

Map[Assert, {BeamEnds[a1, b1], Free[a1]}]

to be added to the already available list of rules in *Mathematica* (through its *Prolog* interpreter). At first, we can verify that at the free end, **a1**, the bending moment and the shear force should be zero. This can be done through the command

Map[Query, {BendingMoment[a1, 0], ShearForce[a1, 0]}]

with a reply **Yes** in both of these queries. Analogously, we can verify that the second end, **b1**, of the same beam is fixed. This can also be verified through the command

Query[Fixed[b1]]

again with a reply **Yes**. We can also prove that both the deflection and the slope at **b1** are zero (naturally, since **b1** was seen to be a fixed end, since **a1** is a free end) by using the command

Map[Query, {Deflection[b1, 0], Slope[b1, 0]}]

Additional and, naturally, more complicated beam-end problems can also be studied by using the same logical approach.

CIRCULAR MOTION OF A PARTICLE

Naturally, *Mathematica* can also be used for the verification of algebraic formulae, yielding the result **True** for correct formulae. In this section, we will consider a somewhat more complicated verification, based on the Groebner-basis algorithm and concerning the motion of a particle on a circumference without an external force. This problem has also been studied by Ioakimidis and Anastasselou [5] by using Maple.

Denoting the radius of the circumference by a and the position of the particle by $(x(t), y(t))$, we have the polynomial (here written as a *Mathematica* command)

h1 = x[t]^2 + y[t]^2 – a^2;

(corresponding to the related equation of the circumference with its right-hand side equal to zero). Additional polynomials (essentially equations), concerning the first two derivatives of $(x(t), y(t))$, with respect to the time t can easily be obtained as

pol1 = {h1, h1a = D[h1, t], h1b = D[h1a, t]}

Now, taking into account Kepler's law in celestial mechanics, we assume that the vector from the center of the circle to the particle sweeps over equal areas in equal times defined through a constant h. Then we will also have the polynomials

pol2 = {h2 = x'[t] y[t] – x[t] y'[t] – h, h2a = D[h2, t]}

corresponding (again) to the related polynomial equations.

In order to use the Groebner-basis algorithm in *Mathematica* (through the corresponding **GroebnerBasis** command), at first we must substitute simple variables for the functions in the polynomials. This can be directly done as

sb = {x[t] -> x, x'[t] -> Dx, x"[t] -> D2x,

y[t] -> y, y'[t] -> Dy, y"[t] -> D2y};

Now we have got the following five polynomials:

pol = {pol1, pol2}/.sb//Factor//Flatten
{$-a^2 + x^2 + y^2$, 2(Dx x + Dy y), 2($Dx^2 + Dy^2$ + D2x x + D2y y),
– h – Dy x + Dx y, D2x y – D2y x}

(corresponding to the related equations).

Our conclusions will concern the facts that both the velocity $\mathbf{v}(t)$ and the acceleration $\boldsymbol{\gamma}(t)$ of the particle should be constants (only as far as their moduli are concerned) in the present simple circular motion. More explicitly, here we will restrict our attention to the verification of the related elementary formulae (our conclusions **con1** and **con2)**

{con1 = $a^2(Dx^2 + Dy^2) - h^2$, con2 = $a^6(D2x^2 + D2y^2) - h^4$};

concerning the velocity and the acceleration respectively. Obviously, these formulae constitute the polynomial interpretations (in Cartesian coordinates and in the *Mathematica* syntax) of the facts that $v(t) = h/a$ and $\gamma(t) = h^2/a^3$.

For the verification of the first of the above two conclusions (concerning the velocity of the particle), it is computationally convenient to use only the first, the second and the fourth of the polynomials **pol**, i.e.

pol1 = {pol[[1]], pol[[2]], pol[[4]]}

plus the auxiliary polynomial **z con1 – 1** (including the new, auxiliary variable **z**) so that a refutational verification can be achieved (since **con1** should coincide to zero). Naturally, for the second conclusion, **con2** (concerning the acceleration of the particle), the whole set of polynomials **pol** is required plus the auxiliary polynomial **z con2 – 1**. Now for the verification of our conclusions, we have to compute the Groebner bases

gb1 = GroebnerBasis[{pol1, z con1 – 1},
{x, y, Dx, Dy, a, h, z}]
gb2 = GroebnerBasis[{pol, z con2 – 1},
{x, y, Dx, Dy, D2x, D2y, a, h, z}]

In both of these cases, the computed Groebner bases were just **{1}**, which simply means that the related sets of polynomial equations, including **con1** and **con2,** are incompatible. Naturally, this is due to the fact that **con1** as well as **con2** should be equal to zero (since they were assumed to be correct conclusions) and, therefore, the related polynomial equations **z con1 – 1 == 0** and **z con2 – 1 == 0** cannot be satisfied for any value of **z**. Thus, in the present case, this result, **{1}**, constitutes the verification of our conclusions with respect to the moduli of the velocity and the acceleration of the particle in the circular motion under consideration. A very large number of additional related results can also be verified in an analogous way.

CONCLUSIONS

From the above results it is concluded that the popular computer algebra system *Mathematica* offers a powerful and integrated mathematical environment not only for symbolic and numerical computations as well as graphics (as has been its normal use so far), but also for the verification of conclusions either of a logical or of an algebraic nature. This can be done either by using *Mathematica* in its original form or in an enhanced form such as that based on Maeder's *Prolog* interpreter in *Mathematica*, which has been used above. Moreover, non-trivial algebraic algorithms such as Buchberger's Groebner-bases algorithm, used in the previous section, and analogous advanced algorithms such as the Collins-Hong cylindrical-algebraic-decomposition-based quantifier elimination algorithm (recently implemented in *Mathematica* too) are also particularly useful. Of course, the *Theorema* project [2] (also based on *Mathematica*) by Buchberger and his collaborators at RISC-Linz (now in β-version and expected to definitively appear after one or two years) offers an even more advanced environment, an actual proof environment in comparison to the verification environment having been illustrated in this paper. Finally, external automated reasoning systems (such as *OTTER*) can also be called and used from inside *Mathematica*.

In any case, both verifications and proofs based on *Mathematica* and *Theorema* seem to offer interesting tools to the mechanical engineering in several problems of applied mechanics such as the few ones already studied in the present paper and a very large number of additional ones in many more areas of mechanical engineering. This seems to be a novel and rather interesting use of classical computer algebra systems (such as *Mathematica*) in engineering science.

ACKNOWLEDGMENTS

The author is most 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* (edited by P. S. Theocaris, D. I. Fotiadis and C. V. Massalas), The University of Ioannina Press, Ioannina, Vol. 2, pp. 759-766.

[5] Ioakimidis, N. I. and Anastasselou, E. G., 1994, "Application of Groebner Bases to Problems of Movement of a Particle", *Computers & Mathematics with Applications*, **27**, pp. 51-57.