Intelligent Assignment Environments for Mechanical Engineering with Computer Algebra

Burkhard Alpers, FH Aalen, University of Applied Sciences,
Department of Mechanical Engineering, 73428 Aalen, Germany,
e-mail: balper@fh-aalen.de

In this article, we show how computer algebra systems (CAS) can be used to quickly develop intelligent assignment environments. Intelligence here means that students working on assignments get as much help as they need to carry on when they get stuck. We provide guidelines for setting up such environments with commercial CAS and we describe a set of generic diagnostic procedures which can be used to set up help procedures for many classes of engineering assignments. The built-in intelligence of CAS with respect to symbolic computation is exploited for implementing these procedures with relatively low effort.

1. INTRODUCTION

An assignment environment is a program which provides students with facilities to solve certain types of assignments. In this broad sense any Computer Algebra System (CAS) can be considered as environment for assignments which include mathematical modelling and computation. When such assignments are offered in worksheets, CAS can be used to solve equations, compute derivatives, plot functions, and so on. Results computed by hand can be checked this way. According to (Holland, 1994), giving additional error information which helps students to find and correct mistakes, thus "coaching" them through the problem solving process, makes such an environment "intelligent" in the sense of so-called "Intelligent Tutoring Systems" (ITS). Whereas a full-fledged ITS also includes student modelling and tutorial strategies using knowledge of student progress and hence requires an enormous effort, the restriction to intelligent help for solving a certain class of problems can be implemented with less resources yet coaching students sufficiently to let them progress.

The main intention of the research presented in this paper is to investigate how capabilities of Computer Algebra Systems like built-in symbolic algorithms or symbolic programming can be used by an instructor in engineering to develop such intelligent assignment environments and how this development can be substantially supported by providing generic diagnostic procedures for frequently occurring classes of mathematical models.

Originally, CAS have not been implemented for didactical purposes (as learning programs). CAS do not give tutorial explanations on how the result was computed, which rules were applied, or which heuristics used. So, essentially CAS act as black boxes and, though comprising a huge amount of computational and algorithmic intelligence, do not provide any kind of intelligent tutoring. Recently, there were several approaches to combine CAS and intelligent tutoring. In the next section, we give a short overview and classification of this research and relate it to our own approach. In section 3, we state guidelines for setting up intelligent assignment environments within existing commercial CAS. Applying these guidelines is substantially supported by diagnostic procedures for certain classes of mathematical models which are described in section 4. Then, typical examples from mechanical engineering demonstrate how easily such an assignment environment can be set up by an instructor. Finally, we discuss restrictions of our approach and directions of future research.

2. CAS AND INTELLIGENT TUTORING

Intelligent Tutoring Systems (ITS) are learning environments which imitate the behaviour of human tutors in guiding the problem solving process of students (for a comparison of human

and software tutors see (Merrill et al., 1992)). (Holland, 1994) identifies four components of such systems:
First of all, an ITS needs an *expert module* for the application area (or special problem class) under consideration. This module must be able to check student solutions for correctness and classify errors. According to Holland, giving informative error analysis which helps the students to see mistakes and perform corrections makes the system "intelligent".
Secondly, an *environment module* takes care of the dialogue between a student and the ITS. This component must provide a comfortable input facility for solutions or solution steps and a comprehensible output format.
Thirdly, a *tutor module* monitors the solution steps of the student where different tutoring modes are conceivable. Feedback can be immediate with only one solution path being allowed, or - to state the other extreme case - response is given only after completion of the task. Holland requires the implementation of "hierarchically graded help".
Finally, the *student module* containing the student model stores information on the performance of a student and uses this information to find causes for mistakes (which concepts are known with a certain probability?) and to decide on the next problem presented to the student. So, student modelling is particularly used for problem sequencing.
It is quite obvious that for areas including mathematical models, CAS can be used for implementing at least parts of the expert module, and the worksheet interface of existing CAS can serve as environment module. But beside this, there are different approaches as to combining intelligent tutoring and CAS. Table 1 gives an overview where the use of AI methods (mostly rule-based systems) and the use of existing CAS serve as classification criteria:

	implements new CAS	*uses existing CAS*
uses AI techniques	CAS on top of AI (Nisheva-Pavlova,1999), (Mitrovic, 1996)	AI on top of CAS (Winkler, 1994)
no use of AI techniques	Educational CAS (Strickland, 1999)	Tutoring Procedures in CAS (Deeba, 1998), (Alpers, 1999)

Table 1

CAS on top of AI: Building "Intelligent CAS"
(Nisheva, 1999) and (Mitrovic, 1996) implement a partial CAS as a rule-based system, called STRAMS (simplification, equation solving, symbolic differentiation and integration) resp. SINT (symbolic integration). As opposed to "traditional" CAS, mathematical knowledge is not implicitly contained in algorithmic procedures but made explicit using rules which can easily be extended or restricted if only a subset known to the student should be used. Moreover, heuristics for using the rules can be implemented (restricting the search space) and the systems are able to learn new schemata (sets of rules for a certain problem type). Such a system is called "Intelligent CAS" since it is built upon techniques of artificial intelligence.
The specific advantage with respect to intelligent tutoring lies in the possibility to use the rules and the heuristics for choosing rules also for explanations. STRAMS gives only complete explanations (list of steps) but cannot provide as yet simply the next step giving the student the chance to continue on his own. This feature is provided in SINT where the student gets immediate feedback. In SINT, a student model is held and updated according to the ongoing performance of the student and answers provided on explicit questions of the system. So, SINT can be regarded as a complete ITS including (very) restricted CAS capabilities whereas STRAMS just implements an expert, environment and (restricted) tutor module.

AI on top of CAS: Implementing rules in CAS

(Winkler, 1994) works the other way around: The rule definition facility of a CAS (here: Mathematica™) is used to construct a tutoring system for fourier transformation. The fourier transform of a certain function is computed by applying production rules ("IF condition THEN action"). Each rule has a textual description which is later used for explanation purposes. In the overall system, the CAS serves to compute the solution (including the intermediate steps) and to check the user input for correctness (by comparing the difference with 0). External parts of the system are used for user dialogue, i.e. as environment module. The system offers different modes of usage: display just the solution; display and explain various solution paths; detect, display and explain the optimal (easiest) solution path. There is no student module, so the system cannot be considered as full-fledged ITS.

Obviously, to implement such a system, knowledge in artificial intelligence techniques (rule formulation, search strategies) is required.

Educational CAS as ITS:

(Strickland, 1999) implements a CAS called TREEFROG with very restricted functionality in basic algebra. The main purpose is to guide students through elementary algebraic manipulations. For this, each input step is checked and immediate feedback is given. As yet, there is no help in form of stepwise guidance, and student modelling is not included. The program uses procedures for rewriting expressions into normal form (applying normalization rules which are not made explicit) and a matching algorithm. It does not use AI techniques for explanation purposes as in "CAS on top of AI". Again, writing such kind of program from scratch requires expertise in normalisation and term matching which will not be found with the usual application expert.

Tutoring Procedures in CAS

Another approach builds upon capabilities of existing CAS as far as possible and uses them as an implementation environment for additional tutoring procedures and as environment module. This approach is persued in ILAT (Deeba, 1998) and "Festigkeitslehre" (Alpers, 1999). Like "AI on top of CAS", this approach also exploits the expert knowledge contained in an existing CAS to provide intelligent tutoring but does not apply any AI techniques.

In ILAT which deals with linear algebra, a library of didactically modified linear algebra procedures was implemented with the Maple™ programming language. The procedures operate in three different modes which can be chosen by the student: in "demonstration" mode, single steps of an algorithm are shown (e.g. single elimination steps in Gauss' elimination algorithm); in "interactive" mode, students are prompted to input the steps on their own (yet, there is no help when they get stuck); the "no-step" mode equals the Maple library procedure which just provides the final result. The demonstration mode is meant to be used in a learning phase, the interactive mode in a knowledge testing phase.

In "Festigkeitslehre" (i.e. stress analysis), an assignment environment in form of worksheets is offered where students can follow help links (giving help text for a specific problem), check intermediate results and get a few hints on faults. For this, special check procedures were programmed in Maple. Major parts of these procedures were specific to the assignments in stress analysis and could not be used for other subjects.

Both ILAT and "Festigkeitslehre" are far from being full-fledged ITS but provide (restricted) intelligent coaching. For both, the main effort lies in designing and implementing tutorial procedures and worksheet environments which is far less than building an ITS and does not require artificial intelligence expertise. In the next section, we describe in a systematical way, how this approach could be used by an application expert in engineering for implementing intelligent assignment environments for mechanical engineering problems.

3. GUIDELINES FOR USING CAS TO IMPLEMENT INTELLIGENT ASSIGNMENT ENVIRONMENTS IN MECHANICAL ENGINEERING

As in the case of task-oriented ITS (Holland, 1994), we restrict ourselves to certain classes of problems instead of covering a whole application area. We do not intend to support concept development and learning with CAS although the experimentation capabilities of CAS can be used for this purpose. We restrict ourselves here to training of concept application. Conceptual learning in a university setting is located within the regular lectures and exercise sessions. Given this curricular embedding, a restriction to concept application seems justified.
In mechanical engineering, work on training problems and assignments often consists of the following steps:

- find the correct model
- set up the model formulae or equations correctly
- compute the formulae or solve the model equations mathematically
- interpret the mathematical result from an engineering viewpoint

For example, finding reaction forces in bearings includes:

- recognizing that in statics the sum of forces and sum of torques must zero;
- setting up the equilibrium equations for all coordinate directions and axes of rotation;
- solving the set of equations for the variables which are asked for;
- interpreting the result, e.g. seeing under which circumstances stability will be lost.

An "intelligent" assignment environment should support at least some of these steps by providing "hierarchically graded help" (Holland, 1994).

The following guidelines state the possibilities and the restrictions of using existing CAS to implement such environments for the kind of engineering tasks described above. The main properties of CAS which can be exploited for this purpose are:

- simplification enabling easy comparison of terms;
- exact computation with non-floating point numbers and symbols;
- routines for solving (systems of) equations und inequalities;
- symbolic differentiation and integration;
- routines for solving (systems of) differential equations.

In the guidelines below we frequently state "low-effort" alternatives since we believe that there should be a reasonable relation between effort and expected benefits within a certain curricular setting which is only known to the creator of tutorial support. The rationale behind the guidelines is to show how one can implement a "system of checks and hints" for solving problems of a predefined class in accordance with the coaching paradigm: give immediate feed-back and give gradual help when the student gets stuck and asks for it but otherwise do not interfere.

1. Identify a class of problems for which tutoring should be supported. This class should be not to wide in order to make support feasible. As an example, one could consider the computation of forces in a mechanical system consisting of components of a certain type, or the computation of motion functions in certain kinematic situations.
2. State examples of this class and use a CAS to compute solutions. Usage here means that all mathematical expressions like formulae and model equations should be defined in the CAS and computations should be made using the above-mentioned capabilities. In this process, intermediate results should be identified which can later be used as "check points" for students to see whether they are "on the right track". Note that there might be different ways to get to a solution. In sophisticated ITS like "Andes", plan recognition methods are applied to get information on the solution path the student might follow (cf. (Gertner et al., 1998). That is out of scope here since CAS do not offer any support in this regard.

In this step, examples can often be taken from books like (Greene, 1995) for mechanics. Moreover, there are many WWW-sites for using CAS in specific application areas.

3. Decide whether it is worthwhile to implement a solution procedure for a certain (sub-) class of problems, e.g. trusses with joints, bearings and external forces (see section 5). This can then later be used as an "expert routine" for solving all problems of the class under consideration. The "low-effort" alternative is just to work with a fixed set of example problems which are solved individually (which is not a big task once the problem has been solved in principle). In both cases, it is again useful to look for available procedures (cf. for mechanical multibody systems the survey in (Brown, 1999)).
4. Identify possible faults and points at which students can get stuck. Certainly, the intermediate results identified in step 2 can serve as starting point here. As to the kind of intermediate faults, instructors often know these from experience (assignments, exams). (Gertner, 1998) gives an overview of error types regarding equations and the frequency of their occurence in a trial use of "Andes". Many of the types listed there are also of relevance in engineering computations which will be treated in more detail in the next section.
5. Check whether existing diagnostic procedures (see next section) are suitable for checking intermediate results and giving hints on errors. If this is the case, the instructor just has to provide data for the correct mathematical model against which the user input is to be checked. If the available procedures are not sufficient, they have to be modified or new ones have to be programmed which requires familiarity with symbolic programming in a CAS environment.

   The critical point in providing help is not to give away too much information freeing the student from thinking himself but to give enough information and an incentive to proceed. This is best done with "graded help" as described in (Gertner et al., 1998) and (Gertner, 1998). Again, it is for the instructor to decide whether the benefits are worth the effort.
6. If a student does not know how to proceed there should be a textual hint at the content of the next step (or at a place in literature or a program where the whole principle solution including the next step is described). In most CAS, this can easily be implemented in form of links to sections in help worksheets as was done in (Alpers, 1999). This is normally restricted to just one solution path without any path finding as in the physics tutor "Andes".
7. Additionally, one can implement solution procedures (or modify those possibly developed in step 3) which have a demonstration mode and/or an interactive mode as in the linear algebra procedures of (Deeba, 1998) described in the previous section.
8. Finally, set up assignment worksheets as learning environment for students. Such a worksheet should contain:
   - a section with the problem description,
   - the names of the variables to be used (otherwise checks are not possible),
   - a list and explanation of the "check and hint" procedures,
   - a list of the available links to a help worksheet which also must be written.

   In the problems one should always use - if numbers are required at all - rational numbers (or functions thereof), since checking results will become very tedious if one has to take into account numerical inaccuracies going along with the usage of floating point numbers. This is not a huge restriction because the environment is to serve educational as opposed to production purposes.

4. INTELLIGENT DIAGNOSTIC PROCEDURES

Step 5 of the guidelines stated above is concerned with diagnostic procedures. In this section we first outline for which mathematical models we developped such procedures. Then, we

explain the architectural relationships between generic procedures, specific procedures and instructor data. Each of these components will then be described in more detail.

Mathematical models to be checked

One first has to clarify which kind of mathematical models occur in the application area for which the assignment environment is implemented. In engineering mechanics, particularly in statics, these are:

- mathematical expressions for variables, e.g.
  $F1x = F1 \cdot \cos(\alpha)$ for expressing the component of a force in direction x;
  $int\_Iy = x^2 \cdot (h - \frac{h}{b} x)$ which shows up in a computation of moments of inertia of a certain triangle;
  $F = (\mu_{01} + \mu_{02}) \cdot \frac{G1}{\mu_{01} \cdot \tan(\alpha) + 1} + \mu_{02} G2$ for force of friction (the μ_{0i} are friction coefficients, the Gi are weights);
- vector expressions: written with coordinates, these are simply mathematical expressions for x, y, and z-coordinates.
- linear equations and systems of linear equations, e.g. systems of equations for equilibrium of forces and torques in statics as will be outlined in the next section; such a system could also be described by matrices (coefficient matrix, right-hand-side-matrix).
- special functional expressions, e.g. the function representing the bending line of a beam under multiple loads which is a piecewise defined function.

The above types of mathematical models may be input by a student working on a given assignment. For these models one needs procedures which check the correctness of the input, identify erroneous parts and give gradual help.

Architecture of CAS procedures

In order to reduce implementation effort and care for reusability, we separated so-called generic procedures from specific diagnostic procedures (cf. Figure 1). Generic procedures are independent from any particular assignment and can be used by many specific diagnostic procedures. The latter provide the user interface for the student and are designed such that the student has similar procedures for many models and has to input as few parameters as possible.

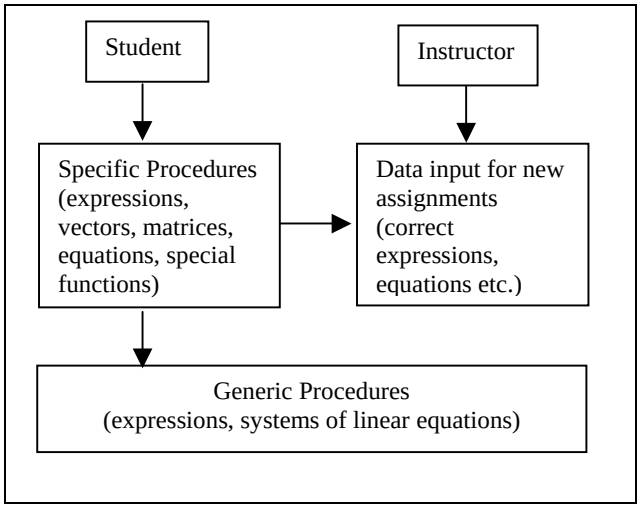

Figure 1

Example: Assume that the correct model consists of a system of linear equations. For checking correctness of an equation that was set up by the student, there is the generic procedure *G_check_equation(equation, system, var_set)* where „equation“ is the equation to be checked, „system“ is the set of correct equations, and „var_set“ is the set of unknowns (for the Maple code of this procedure cf. the appendix of this paper). Now, the student should not be bothered with the set of unknowns. Therefore, the respective checking function for the student is simply *check_equation(equation, system)* (If there is just one system within an assignment, one could also omit the name of the correct system). Moreover, this specific procedure returns a readable message like "Your equation is not correct" whereas the generic procedure just returns an error code. This way, one can also easily implement specific procedures for different user languages.
An instructor simply has to put the correct solution in a file adhering to certain naming conventions described below.

Generic procedures

We implemented two sets of generic procedures which can be used in many specific diagnostic procedures (cf. Figure 1). The procedures for expression checking compare an input expression with the correct expression. This is quite easily implemented in a CAS environment where all the necessary "intelligence" is already available. Essentially, if the difference of the input and the correct expression simplifies to 0, then the input is correct. In self-contained Intelligent Tutoring Systems, this is usually programmed from scratch which consumes many resources and is probably far more restricted.
It is more difficult to find the kind of error and the erroneous part within an expression. Some errors like a sign error, a missing symbol or a wrong symbol can be found easily, others like a wrong factor or wrong summand depend on the representation of the expression and are therefore hard to find if at all. Consider for example the expression 2x+3ax or equivalently (2+3a)*x. If the wrong input is 2x+ax, then compared with the first representation, the "outermost" error type is "wrong summand" whereas compared with the second representation it is hard to give any hint at all. One could transform 2x+ax into (2+a)*x and then return the error type "wrong factor". This simple example already shows that even in a CAS environment it is hard to give a meaningful response concerning the wrong part of an expression.

The other expression checking procedures listed in Figure 2 are auxiliary procedures. The final two provide information on the difference between the set of factors (resp. summands) of expressions 1 and 2.

Generic Procedures for expression checking
G_check_expression(expression, correct_expression)
G_find_error_expression(expression, correct_expression)
G_extract_symbol_set(any_expression)
G_compare_symbol_sets(expression1,expression2)
G_compare_product_expressions(expression1,expression2)
G_compare_sum_expressions(expression1,expression2)

Generic Procedures for equation checking
G_check_equation(equation, system, var_set)
G_check_system(system1, system2, var_set)
G_find_error(equation, system, var_set)
G_provide_missing(system1, system2, var_set, number)
G_compare_solution_sets(set1,set2, var_set)
G_compare_equations(equation1,equation2, var_set)

Figure 2

The second set of generic procedures deals with systems of linear equations. The following types of errors can be made when setting up such a system of linear equations: syntactical errors, an equation is missing, an equation is wrong. (Gertner, 1998) presents a list of error types for wrong equations and reports on the frequency of their occurrence when students used the physics tutor "Andes". Besides using wrong variables and making syntactical errors, sign errors occurred most frequently (35 out of 102). Other errors include missing, wrong or extra factor (17), wrong numbers (17), and wrong or missing terms (26).
In order to find wrong equations and errors therein, the following procedures have been implemented in Maple with relatively low effort:

- *G_check_equation(equation, system, var_set)*: Check whether an equation is "compatible" with a system of equations (in the variables given in var_set) in the following sense: If the equation is added to the system, the solution set does not change which is easy to check with a CAS (essentially solution and set comparison, cf. appendix). So, we do not need to pre-generate a list of "primitive" correct equations as was done in (Gertner, 1998).
- *G_check_system(system1, system2, var_set)*: Check whether the first system is equivalent to the second; if this is not the case, find out which equations of the first system are not compatible with the second in the sense described above.
- *G_find_error(equation, system, var_set)*: Check where the error in the equation is which makes it incompatible with the system; the kind of errors discovered are described below. The routine returns the error type, erroneous terms, and the probably correct equation. For this, it calls the routine *G_compare_equations* which can also be used separately for comparison with known "buggy equations" (like "buggy rules" in rule-based systems).
- *G_provide_missing(system1, system2, var_set, number)*: Provide "enough" equations from system2 such that together with the equations from system1 the required number of independent equations is achieved. For this, find out successively equations in system2 which are independent from those in system1 and those formerly added. If (optionally) only a certain number of equations is asked for (e.g. only one in order to try again without further help), then at most so many equations are returned.

Note that syntactical errors are discovered by a CAS automatically and a hint is given to the user (although this is not always totally satisfactory). For some of the error types listed by (Gertner, 1998) it is easy to program detection procedures which work only in "most cases" and hence provide hints on potential errors. This seems to be reasonable although one can construct examples where the procedures do not work. The types of errors currently detected are: *sign errors only, wrong factors only, wrong term only (with the correct one missing), missing term only*. It should be mentioned that the error detection described above is not as sophisticated as the one reported in (Gertner, 1998) but it can be implemented very rapidly.

Specific procedures

Specific procedures are those tutorial procedures which are meant to be used by the student. Therefore, they should be very easy to use, i.e. the input parameters should be few and easy to understand and the output should be a meaningful natural language sentence. Moreover, they should be easy to remember and hence follow a common pattern. For achieving this, the student is offered essentially three types of diagnostic procedures:

check_xxx(name (,system)),
hint_xxx(name (,system), kind_of_hint),
show_correct_xxx(name (,system))

where xxx stands for expression, equation, vector, matrix, or foeppl-function (piecewise defined function, cf. section 5), and the parameter „system“ has to be provided when dealing

with systems of equations. For more complex data structures like vectors, matrices, or systems of equations, we provide in addition the possibility to check single entries (coordinate, matrix position, equation in a system). The input parameter "name" serves to specify the name of the variable, vector, equation etc. to be checked.
Figure 3 depicts the tutorial procedures for systems of equations. They are based on the generic procedures described above. An instructor can use these procedures without any additional programming effort or he can program his own - possibly with additional verbal hints - using the generic ones.

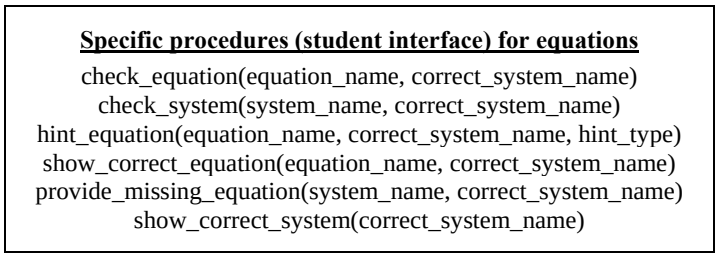

Figure 3

A student working on a certain assignment involving linear models has to use the given symbol names in order to enable the tutoring procedures to work correctly. The student is assumed to set up the model equations using these symbols. For this, a (not very deep) familiarity with the CAS used is necessary. The tutoring procedures provide a "system of checks and hints" which can and should be used by the student on demand. He should first use checks to see whether an intermediate result is correct. He can do this equation by equation or for the whole system. If an erroneous equation or system is reported the student should first try to find the error himself which will also be the case during the exam. If he gets stuck he should use the hints and explanations. Only finally should he ask for the correct equation or system. This will not always be sufficient, particularly if a conceptual misunderstanding is the cause for an error. In this case, the instructor as a human tutor must be asked.
Once a correct system of linear equations has been set up by the student, its mathematical solution by Gauss elimination can be tutored by the procedures provided in (Deeba, 1998) in demonstration or interactive mode as pointed out in section 2.

Specification of data for new assignments

If an instructor wants to set up a new assignment and is satisfied with the specific diagnostic procedures already implemented, he only has to provide the correct data for the mathematical model to be checked. He has to adhere to certain naming conventions in doing this but otherwise there is no additional work, particularly no programming work in the CAS. To illustrate this, we give a short example:
The instructor first has to define an assignment name like (we use Maple™ notation)

```
> assignment:=EngineeringMechanicsNo1;
```

Assume that the mathematical model to be checked is a system of linear equations. The instructor has to store the correct set of equations and the set of variables in a file in the following way:

```
EquationsEngineeringMechanicsNo1:=
{equilibrium={2*Fx+Ax+Bx=0, 3*Fy-Ay=0, b*Ay+c*Fy=0}};
VariablesEngineeringMechanicsNo1:={equilibrium={Ax,Ay,Bx}};
```

Here, equilibrium is the name of the correct system of equations. This way it is possible to set up several systems for one assignment which is useful in statics where often the overall structure is devided into subsystems for which separate systems are set up. In order to check an equation set up by a student (e.g. my_equation:=3*Fy-2*Ay=0), the specific procedure check_equation can be used: check_equation(my_equation, equilibrium).

5. EXAMPLES FROM MECHANICAL ENGINEERING

The assignments described in this section are part of the assignment environment for engineering mechanics I, the first mechanics course in the mechanical engineering curriculum.

Example 1: Fixing a boat with two ropes and a pile

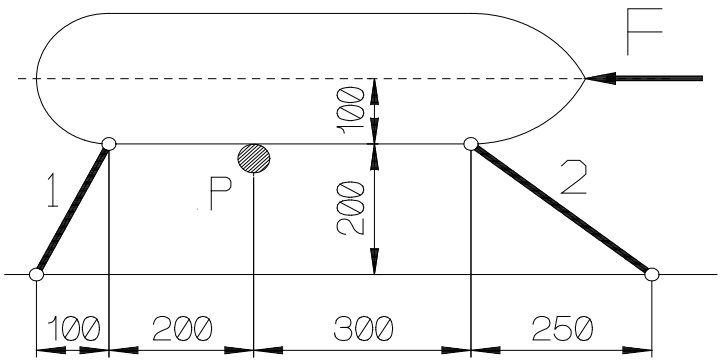

Figure 4

A boat is fixed by two ropes and a pile. The current exerts a force F=250 N on the boat. Determine the forces in the ropes and the force the pile exerts on the boat.
This is a typical assignment in statics. It is solved by setting up the equilibrium equations, i.e. forces in direction x and y resp. as well as torques wrt. an arbitrary point of rotation add up to 0. For doing this, first forces have to be identified and drawn and names for the forces have to be chosen as depicted in Figure 5. Since a CAS usually does not contain a drawing tool from which the forces and their names can be retrieved, the student is forced to use predetermined names, here: F, S1, S2, P, alpha1, alpha2 (see Figure 5). Otherwise the checking procedures cannot compare the input model with the correct model.

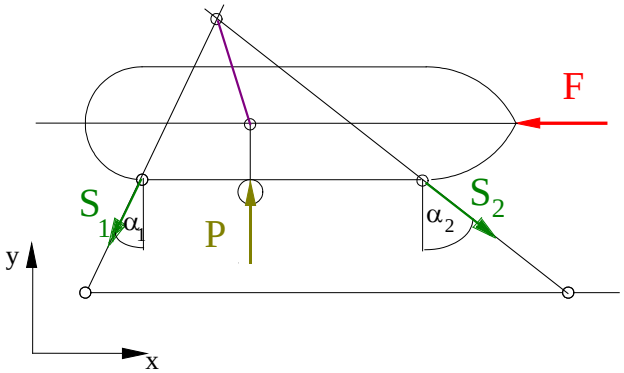

Figure 5

The correct model equations are:

*-S1*sin(alpha1) + S2*sin(alpha2) - F = 0 (equilibrium of forces wrt. x-direction)*
*P - S1*cos(alpha1) - S2*cos(alpha2) = 0 (equilibrium of forces wrt. y-direction)*
*S1*cos(alpha1)*200 - S2*cos(alpha2)*300 + F*100 = 0 (equilibrium of torques wrt. P)*

This system of equations has to be named by the instructor (like "system_xxx") and stored in a file as described in the previous section. This file as well as the file containing the tutorial procedures is read at the beginning of the assignment worksheet. Nothing more needs to be done by the instructor if he is content with the existing specific diagnostic procedures.
A student working on the problem has to set up the system equations. If the student has conceptual problems, e.g. he does not know how to proceed from the beginning, textual help could be given by proving a help link (cf. Alpers, 1999). But for this, mathematical CAS capabilities are not required. When setting up equilibrium equations, the student might as well choose the point where S1 acts as rotation-"point" resulting in a different but equivalent system of equations. Hence, there is not just one set of correct equations but – theoretically – infinitely many.
The tutoring procedures described in section 4 allow the student to check single equations or the whole system. In case the student has forgotten about equilibrium of torques and hence does not input the third equation, the *provide_missing* procedure helps.
The errors which occur frequently with such assignments are sign errors or missing terms, e.g. in the second equation *P* is forgotten or the sign of *-S2*cos(alpha2)* is wrong. Moreover, if *P* is erroneously treated as a fixed support (having reaction forces in both x- and y-direction), then *Px* and *Py* show up in the equations. Such faults are detected by the *hint_equation* routine. Using this routine, the student can get first help on the error type and then, if that is not sufficient, on the wrong terms. Finally, if the student has no idea how to fix the error, *show_correct_equation* gives the correct equation.
But also note that there are errors which are not yet found: If, e.g., the student chooses the point where S1 acts as rotation-"point" for writing up equilibrium of torques and makes a mistake, the erroneous part is not found and the student only gets as a response that the equation is faulty.

Example 2: Shearing force function for a loaded supporting beam

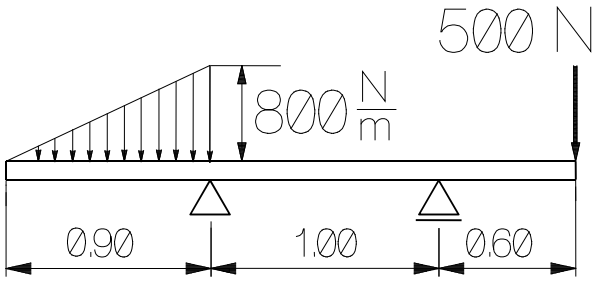

Figure 6

Figure 6 depicts a supporting beam with a linearly increasing line load (e.g. the weight of a block) and a force. Determine the shearing force function Q(x) for the beam.

For solving this problem, first the student has to find the reaction forces of the bearings A (left) and B (right, cf. Figure 7) where A is a fixed support and B is movable in x-direction. This is done by setting up equilibrium equations as described in the previous example (only the linearly increasing load has to be substituted by a force F through the centre of the triangle). Once A=168 N and B=692 N are known, the function for shearing force must be set up which is a piecewise-defined function. For this, mechanical engineers often use the so-called foeppl-symbol $\{x-a\}^n$ which is a short notation for the expression

$$(x-a)^n \cdot \text{Heaviside}(x-a) = \begin{cases} 0, & x < a \\ (x-a)^n, & x \geq a \end{cases},$$

The line load function can easily be expressed using the foeppl symbol in the following way:

$$q(x) = -\frac{800}{0.9}\cdot x + \frac{800}{0.9}\cdot\{x-0.9\}^1 + 800\cdot\{x-0.9\}^0$$

The shearing force caused by this line load is obtained by integrating this force which can be done by summand: $-\frac{800}{2\cdot 0.9}\cdot x^2 + \frac{800}{2\cdot 0.9}\cdot\{x-0.9\}^2 + 800\cdot\{x-0.9\}^1$

Moreover, the shearing force caused by the force A (resp. B) starting at x=0.9 (resp. x=1.9) can be expressed as $A\cdot\{x-0.9\}^0$ resp. $B\cdot\{x-1.9\}^0$. Therefore, the resulting correct function is:

$$Q(x) = -\frac{800}{2\cdot 0.9}x^2 + \frac{800}{2\cdot 0.9}\{x-0.9\}^2 + 800\{x-0.9\}^1 + A\{x-0.9\}^0 + B\{x-1.9\}^0$$

which is depicted in Figure 7.

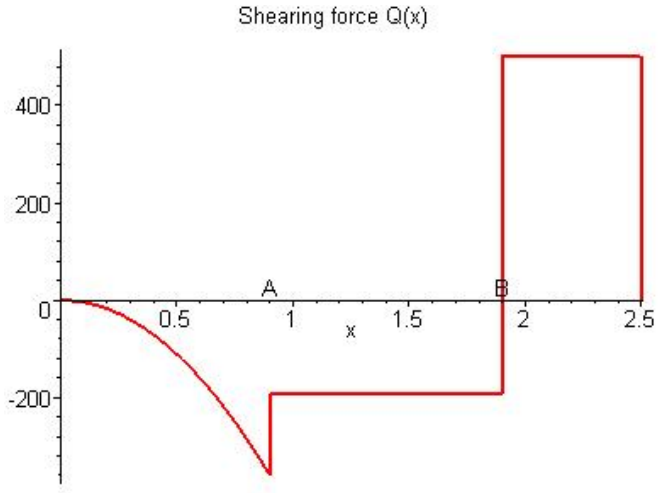

Figure 7

Similar assignments provide functions of the same type, i.e. there is a part without foeppl symbol and then there are terms "Factor*{symbol}^{n}". For such functions we implemented diagnostic procedures which check for correctness and give hints if the input is erroneous. If for example a certain symbol is missing or if the exponent or the factor is wrong, this is detected and reported. Considering the above function, a student might have forgotten to take into account the reaction force B and hence left out the part B*{x - 1.9}^{0} . Then, he first gets a more general hint saying that a certain foeppl place (here: 1.9) is missing, so he can think about his function again. If that hint is not sufficient for the student, he can get a further hint on the missing symbol.

Here again, the only work for an instructor setting up such an assignment consists of storing the correct function in a file and reading the available tutorial procedure in the worksheet.

6. Discussion and future work

Using the guidelines and procedures stated in the previous sections, we implemented an intelligent assignment environment for two classes in the first term of the mechanical engineering study course: Engineering Mechanics and Stress Analysis. The environment consists of all exercises formerly handed out on paper including the final results. It is embedded in an overall learning program to interconnect application subjects and mathematics in the mechanical engineering curriculum (cf. Alpers, 1999). Since engineering mechanics is one of the largest "obstacles" for mechanical engineering students to overcome, the material is accepted by students as additional help although they are required to get

familiar with the CAS environment to a certain extent. But since this familiarity can be used for several subjects including mathematics where they have to work on application projects using CAS and CAD, it seems to be worthwhile spending the additional effort.
Our current approach still has several restrictions which are partially to be overcome by future research:

- The diagnostic procedures heavily rely on the simplification capabilities of the CAS used. If the CAS is not able to simplify the difference of input and correct expression to 0 although the input is correct, then the error message provided will be false. Consider for example the expressions $\sqrt{x^2}$ and $|x|$ which are identical for $x \in \mathbb{R}$. Maple™ does not simplify the difference to 0 since x might also be complex. In this case, the problem can be avoided by telling Maple™ via the "assume" command that x is assumed to be real.
- Some of the diagnostic procedures work properly only with symbols and rational numbers but not with floating point numbers. If, for example, solution sets of linear equations are compared, there might be differences due to numerical inaccuracies. Currently, this results in a wrong diagnostic message. To avoid this, one could work with "near" equality up to an ε-deviation (depending on the condition of the coefficient matrix).
- More research is needed to give useful messages on the erroneous part within an expression. As stated above, when the correct expression and the user input are not identical (i.e. the difference does not simplify to 0), it is hard to find which part of the input expression is wrong. Probably, one should restrict oneself here to faults which are known to occur frequently.
- There is a danger that ill-motivated students use tutorial procedures too easily. This can only be avoided when the procedures are placed in an environment where requesting hints leads to „loosing points“ as is the case in the AIM environment (Klai et al., 2000). How such an integration works, is for further study.
- The most important restriction concerning support of the problem solving process is that the environment gives only textual help when setting up the models which can be diagnosed later. What would be really needed but cannot be realized easily within a CAS environment would be a drawing pad facility where a mechanical configuration is depicted and students can insert forces and torques. This would enable tutoring in early model construction phases. Moreover, the current restriction that certain variable names have to be used (giving information on the important variables in the configuration) would no longer be necessary. In statics, there are often different possibilities for sectioning when a larger configuration is divided into subsystems. Currently, model checking is only offered for certain predefined subsystems (giving already information on adequate sectioning). If a drawing facility would be available, support for checking different sectionings could be offered.

The last item leads to a more general research topic. The diagnostic procedures we implemented are needed in nearly any Intelligent Tutoring System (ITS) dealing with natural science or engineering where checking and error finding in symbolic mathematical user input is neccessary to provide adequate help. Therefore, it would be useful to offer a mathematical model checking agent as a service over the internet. This would properly fit into component or agent architectures currently discussed in the ITS community (cf. (Ritter, 1998) and (Lelouche, 2000)). Such architectures aim at avoiding implementation from scratch by reusing existing components which makes realization of ITS easier and cheaper.

References

Alpers, B. (1999). *Combining hypertext and computer algebra to interconnect engineering subjects and mathematics*, Proc. of the 4th Int. Conference on Technology in Math. Teaching, Plymouth.

Brown, I., Larcombe, P. (1999). *A survey of customised computer algebra programs for multibody dynamic modelling*, in: Munro, N.: Symbolic Methods in Control System Design and Analysis, IEE, London.

Deeba, E., Gunawardena, A. (1998). *Interactive Linear Algebra with Maple V*, Springer, New York.

Gertner, A.S. (1998). *Providing feedback to equation entries in an intelligent tutoring system for Physics*, in: Goettl, B.P. et al..: Intelligent Tutoring Systems, Proc. of 4th Int. Conf. ITS '98, Springer, Berlin (=LNCS 1452), pp. 254-263.

Gertner, A.S., Conati, C., VanLehn, K. (1998). *Procedural help in Andes: Generating hints using a Bayesian network student model*, in: Proceedings of the 15th National Conference on Artificial Intelligence. Madison, Wisconsin.

Greene, R. (1995). *Classical Mechanics with Maple*, Springer, New York.

Holland, G. (1994) *Intelligent Tutorial Systems*, in: Biehler, R. et al.: Didactics of Mathematics as a Scientific Discipline, Kluwer, Dordrecht, pp. 213-223.

Klai, S., Kolokolnikov, Th., Van den Bergh, N. (2000): Using Maple and the Web to Grade Mathematics Tests, in: Kinshuk, J. C. & Okamoto T. (Ed.): Advanced Learning Technology: Design and Development Issues, Los Alamitos, CA: IEEE Computer Society (IWALT 2000).

Lelouche, R. (2000). *A Collection of Pedagogical Agents for Intelligent Educational Systems,* in: Frasson, C., Gauthier, G., vanLehn, K.: Intelligent Tutoring Systems, Proc. of 5rd Int. Conf. ITS 2000, Springer, Berlin (=LNCS 1839), pp. 143-152.

Merrill, D., Reiser, B., Ranney, M., Trafton, J. (1992). *Effective Tutoring Techniques: A Comparison of Human Tutors and Intelligent Tutoring Systems*, The Journal of the Learning Sciences 2(3), pp. 277-305.

Mitrovic, A. (1996). *SINT - a Symbolic Integration Tutor*, in: Frasson, C., Gauthier, G., Lesgold, A.: Intelligent Tutoring Systems, Proc. of 3rd Int. Conf. ITS '96, Springer, Berlin (=LNCS 1086), pp. 587-595.

Nisheva-Pavlova, M.M. (1999). *An Intelligent Computer Algebra System and its Applicability in Mathematics Education*, The International Journal of Computer Algebra in Mathematics Education, Vol. 6, No.1, pp. 3-16.

Ritter, S., Brusilovsky, P., Medvedeva, O. (1998). *Creating More Versatile Intelligent Learning Environments with a Component-Based Architecture*, in: Goettl, B.P. et al..: Intelligent Tutoring Systems, Proc. of 4th Int. Conf. ITS '98, Springer, Berlin (=LNCS 1452), pp. 554-563.

Strickland, P., Al-Jumeily, D. (1999). *A Computer Algebra System for Improving Student's Manipulation Skills in Algebra*, The International Journal of Computer Algebra in Mathematics Education, Vol. 6, No.1, pp. 17-24.

Winkler, H.-J. (1994). *A Tutoring System for Fourier-Transforms in Electrical Engineering*, in: J. Fleischer et al. (Eds.): Computer Algebra in Science and Engineering, World Scientific Pub., Singapore, pp. 331-339.

BIOGRAPHICAL NOTES

Burkhard Alpers studied mathematics, history and educational sciences and received his PhD in Mathematics from the University of Hamburg. For several years, he worked at the Corporate R&D Division of Siemens AG on topics in network management. He is currently professor of mathematics and computer science in the Department of Mechanical Engineering at the University of Applied Sciences in Aalen, Germany. He is interested in educational and industrial applications of computer algebra, optimization and geometric modelling in CAD.

APPENDIX: Example for coding tutorial procedures in MAPLE ™

The following examples contain procedures for checking whether a linear equation is compatible with a system of correct linear equations. The generic version presented here needs as input the equation to be checked, the set of correct equations and the set of variables whereas the tutorial procedure used by the student only requires the equation and the name of the correct system.

```
#=========================================================================
#========Check equation for correctness  ================================
#=========================================================================

G_check_equation:=proc(equation,system,var_set)

 #=== check whether equation is compatible with the correct system in a
 #=== certain variable set var_set. Compatible means that system and system
 #=== union {equation} provide the same solution.
 #=== If so, the routine returns 0, else 1.

local correct_solution,extended_sol;   # declaration of local variables

correct_solution:=solve(system,var_set);   # solve system exactly

if correct_solution = NULL then correct_solution:={};fi;
                                          # if solve can't solve
extended_sol:=solve(system union {equation}, var_set);
                        # add equation to be checked to system and solve
if extended_sol = NULL then extended_sol:={}; fi;   # if solve can't solve

if G_compare_solution_sets(extended_sol, correct_solution, var_set)=1
      then RETURN(1);
else  RETURN(0);        # uses procedure G_compare_solution_sets, see below
fi;

end:

#=========================================================================
#====================  compare solution sets                    ========
#=========================================================================

G_compare_solution_sets:=proc(set1, set2, var_set)

 #=== set1 and set2 are assumed to be solution set output from the solve
 #=== procedure. WARNING: solve might produce a NULL sequence in which case
 #=== the sets must be set to {}.
```

```
#=== var_set is assumed to be the set of variables for which a system was
#=== solved

local variable, set_member, solution1, solution2;
                                          #declaration of local variables

# now check for each variable in var_set whether the solution in both
# sets is the same

for variable in var_set do

      for set_member in set1 do
            if variable=op(1,set_member) then
               solution1:=op(2,set_member); break;fi;
                  # find solve result for variable in set1
      od;
      for set_member in set2 do
            if variable=op(1,set_member) then
               solution2:=op(2,set_member); break;fi;
                  # find solve result for variable in set2
      od;

      if simplify(solution1 - solution2) <> 0     # compare solve results
        then RETURN(1); fi;   # sets are not equal
od;

RETURN(0);    # sets are equal

end:
```