Computer Aided Assessment of Mathematics Using STACK

Christopher Sangwin

Abstract Assessment is a key component of all teaching and learning, and for many students is a key driver of their activity. This paper considers automatic computer aided assessment (CAA) of mathematics. With the rise of communications technology this is a rapidly expanding field. Publishers are increasingly providing online support for textbooks with automated versions of exercises linked to the work in the book. There are an expanding range of purely online resources for students to use independently of formal instruction. There are a range of commercial and open source systems with varying levels of mathematical and pedagogic sophistication.

History and Background

Assessment is a key component of all teaching and learning, and for many students is a key driver of their activity. Computer aided assessment (CAA) has a history going back over half a century, for example (Hollingsworth 1960) reports a "grader" programme which automatically checked some aspects of students' computer programmes. "*We are still doing considerable hand filing of punched cards at this stage. This large deck of cards which includes the grader program is then run on the computer. Our largest single run has been 106 student programs covering 9 different exercises.*" By the mid-1960s computers were being used to teach arithmetic, e.g. (Suppes 1967), and by the 1980s there were a number of separate strands of research in this area including the artificial intelligence (AI) community, e.g. (Sleeman and Brown 1982). The ambitious goals of such artificial intelligence-led systems have only been achieved in confined and specialized subject areas. These difficulties were acknowledged early, for example in their preface to (Sleeman and Brown 1982).

C. Sangwin (✉)
University of Birmingham, Birmingham, UK
e-mail: C.J.Sangwin@bham.ac.uk

S.J. Cho (ed.), *Selected Regular Lectures from the 12th International Congress on Mathematical Education*, DOI 10.1007/978-3-319-17187-6_39

> Early CAI (Computer Aided Instruction) workers set themselves the task of producing teaching systems which could adapt to the needs of individual students. It is now generally agreed that this is a very difficult task, and one that will only be accomplished as a result of extensive research in both AI and Cognitive Science.

For example, in the case of multi-digit subtraction, sixty separate skills were identified by Burton (1982), including "*27. Borrow once in a problem*", and "*46. Subtract numbers of the same length*". The majority of these skills involve borrowing, and they are linked in a hierarchy. Assuming a student gives incorrect final answers, their goal was to identify which parts of the algorithm are not being performed correctly. Clearly this is necessary to provide specific feedback to the student. They defined a "bug" as a "*discrete modification to the correct skills which effectively duplicate the student's behaviour*", such as $0 - n = n$ when subtracting individual digits.

> […] we were able to design a test capable of distinguishing among 1200 compound bugs with only 12 problems! A second important property of a test is that it cause each bug to be involved often enough to determine that it is consistent. The current tests that we are using are designed to cause each primitive bug to generate at least three errors. To accomplish this it was necessary to have 20 problems on the test (Burton 1982, p. 172).

The validity of such tests can be confirmed, they are reliable and (subject to the availability of computing resources) practical. A student's consistency in making particular errors can be checked in an online *adaptive system* by choosing the next question based on the responses received so far.

> The skills were organized into a hierarchy in order that an expert system could make inferences on student knowledge from the answers previously given and select the most appropriate question to ask next. This reduces the number of questions asked for a variable ability group (Appleby et al. 1997, p. 115).

The central difficulty is designing a good network of questions.

Alternative approaches have focused more on formative assessment. Rather than generating reliable diagnoses of a student's difficulties, they provide feedback which is designed to help students learn particular skills. For example, the group at Heriot-Watt University in the United Kingdom have more than a quarter of a century of experience in this area (see Beevers et al. 1991; Ashton et al. 2006). From the outset, one goal of this project was to assess students' steps in working.

> By giving the correct answer at each stage the student is encouraged to carry on but still has to work at each part before the answer is revealed. In a traditional text book example either the only answer given will be the final one or, if the question is answered in stages, it is difficult to avoid seeing all the answers at once (Beevers et al. 1991, p. 112).

In the CALM system of (Beevers et al. 1991), as in some contemporary systems, the extent to which the *mathematical properties* of students' individual mathematical expressions can be established was limited. We shall expand on this issue below. The problem of assessing a *complete mathematical argument*, without providing a template of teacher-designed answer boxes, remains an elusive goal.

However, by the early 1980s there was a backlash from some about the use, indeed as they saw it abuse, of computers for testing in this way. E.g.

> In most contemporary educational situations where children come into contact with computers the computer is used to put children through their paces, to provide exercises of an appropriate level of difficulty, to provide feedback, and to dispense information. The computer is programming the child (Papert 1980).

One result of this dissatisfaction was LOGO. Other exploratory computer environments, sometimes called microworlds, were another, see for example Noss and Hoyles (1996).

The difference between a focus on *skills and their acquisition* and *conceptual understanding* is a classic dichotomy in mathematics education. The classical algorithms of elementary mathematics, including place-value addition, subtraction, multiplication and division of numbers, together with algebra are sophisticated. The lattice of skills in basic subtraction of (Burton 1982) provides graphic evidence of this complexity. To achieve basic competence requires focus of attention and diligent practice. Methods in calculus, linear algebra and progressively more advanced topics rely on these foundations. Practice of such material has provided, and remains, a significant and important use of computer aided assessment systems. The popularity of freely available resources such as the Khan Academy (see http://www.khanacademy.org) is testament to this assertion. Publishers are increasingly providing online support for textbooks with automated versions of exercises linked to the work in the book. There are a range of commercial and open source systems with varying levels of mathematical and pedagogic sophistication. Therefore teachers are likely to increasingly mix CAA with traditional assessments. Furthermore, CAA is increasingly being used to automate high stakes final examinations. Currently, this is not widespread for school or university exams, but CAA forms a component of professional exams and in other areas such as the UK hazard perception test, an automated component of the UK driving test. This use is likely to increase significantly over the next decade. A survey of existing CAA can be found in Chaps. 8 and 9 of Sangwin (2013).

The focus in this paper is on CAA where the teacher may, at least in principle, author their own questions. However, this requires a new level of sophistication in using technical software packages, in particular of using CAS. In this regard, (Sangwin and Grove 2006) commented that "*teachers are often neglected learners*" themselves. It is necessary to have a detailed understanding in order to automate both the mathematics itself and details of the assessment process. Therefore CAA provides us with an opportunity to reconsider assessment, both its purposes and practices. All forms of assessment, from multiple choice questions, written answers under exam conditions, project work and oral examinations provide opportunities within certain constraints. CAA is no exception and while CAA does not claim to enable us to "assess everything" current CAA systems have a mathematical sophistication which enable valid assessment of significantly more than shallow procedural skills. Charles Babbage is reputed to have said the following:

> Propose to an Englishman any principle, or any instrument, however admirable, and you will observe that the whole effort of the English mind is directed to find a difficulty, a defect, or an impossibility in it. If you speak to him of a machine for peeling a potato, he will pronounce it impossible: if you peel a potato with it before his eyes, he will declare it useless, because it will not slice a pineapple.

CAA is just a *tool*, and it has some uses while not claiming to achieve everything.

It is interesting to recall how few tools were available to help the CAA system designer during the mid-1980s, and the power and sophistication of machines available to students. Even code to evaluate an expression at a point, crucial to establishing algebraic equivalence, had to be written from scratch by Beevers et al. (1991). In the last ten years software development increasingly takes advantage of code libraries for specific purposes, for example MathJax (http://www.mathjax.org) enables most web browsers to display mathematical expressions. Furthermore, teams of like-minded individuals collaborate on software projects and freely share the results. These include content management systems, such as Moodle (http://moodle.org/). For many students, at least in the more economically developed counties, the last ten years have enabled wide access to wireless networks through laptop computers and hand-held mobile devices.

STACK

In this section we describe STACK, an advanced general CAA system for mathematics, with an emphasis on formative assessment. The primary design goal was to enable a student to enter a mathematical answer in the form of an algebraic expression. While multiple choice questions (MCQ) have a place, the student selects an option from a list provided by the teacher. The purpose of many questions is grotesquely distorted by using a MCQ for mathematics, and hence the assessment is invalid. For example, solving an equation from scratch is significantly different than checking whether each potential response is indeed a solution. We preferred a system which evaluates student provided answers.

An example question, with student's response and feedback, is shown in Fig. 1, (see Sangwin 2010). The student's answer is a mathematical expression which they must enter into a form through a web browser. STACK then establishes the mathematical properties of the answer. For many questions the teacher will seek to establish that (i) the answer is algebraically *equivalent* to the correct answer and (ii) the student's answer is in the appropriate *form*, (e.g. factored). However, the answer need not be unique and STACK establishes properties of expressions, but remains an objective testing system. In the case of Fig. 1 the teacher does not simply establish that the student's answer is "the same" as their answer. A list of separate properties is needed, and many different answers may satisfy these. Notice that the feedback in Fig. 1 is specific to the answer and directly related to possible improvement on the task. STACK may include and display results of computer

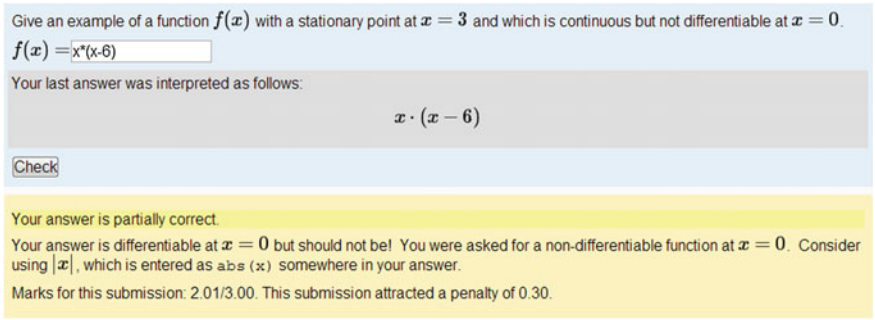

Fig. 1 An example STACK question

algebra calculations within such feedback which can be as detailed as appropriate to the question. This is a particular distinguishing feature of STACK.

In particular STACK uses the computer algebra system Maxima to

- randomly generate problems in a structured mathematical way;
- establish the mathematical properties of expressions entered by the student;
- generate feedback, as necessary, which may include mathematical computations of the student's answer;
- help the teacher analyse the attempts at one question, or by one student.

Version 1 of STACK was a stand-alone quiz system. Version 2 was partially integrated into the Moodle content management system while version 3 provides a question type for the Moodle quiz. STACK was designed and developed by the author, with substantial code for version 3 being written by Tim Hunt of the Open University in the UK. Numerous other colleagues have contributed to the design, code, testing and documentation. Version 2 of STACK has been translated into Finnish, Portuguese, German, Dutch and Japanese and is in regular use by large groups of students for formative and summative assessments. See (Sangwin 2010).

Establishing Properties

The key issue for teachers is to articulate the properties sought in an expression, and then encode an algorithm which establishes each of these. Where only some of the properties are satisfied the system may provide feedback to the student. The most common property a teacher will wish to establish is that two expressions are equivalent, for example the student's and teacher's respective answers. In many systems, e.g. (Beevers et al. 1991), the algebraic equivalence of two expressions is established by choosing random numbers and evaluating each expression at these points as floating point approximations. Numerical analysis assures us that a reasonable match between the values for each expression gives a reasonable

probability of the two expressions being identical. Since students' expressions are usually relatively simple the probability of a false result is very low indeed. This approach was also used by Appleby et al. (1997) and many others. An alternative approach is to assign the variable SA to represent the student's answer and TA to be the teacher's and use the computer algebra command simplify and evaluate the following pseudocode.

if simplify(SA − TA) = 0 then true else false.

If this test returns true then we have established the two expressions are equivalent. However, if the test returns false are we really sure they are not equivalent? What matters in this approach is the strength of the "simplify" command, and the ability to know with certainty that the zero expression really means zero, and a non-zero expression is really non-zero. Different CAS implement functions such as "simplify" in a surprising variety of ways, and (Wester 1999) provides an interesting, but dated, comparison of CAS capabilities. From a theoretical perspective implementing this test is not possible in all cases! (Richardson 1966) showed that there is no algorithm which can establish that a given expression is zero in a finite number of steps for a reasonable class of elementary expressions. These results have been refined, e.g. by Matiyasevich (1993) and sharpened, e.g. by Caviness (1970) who showed that for the sub-class of polynomials over the complex rational expressions together with unnested exponential functions then zero really means zero, i.e. we do have a canonical form. Moses (1971) comments

> In fact, the unsolvability problem may lie in Richardson's use of the absolute value function. When one adds the absolute value function to a class of functions which forms a field (e.g. the rational functions), then one introduced zero divisors. For example, $(x + |x|)(x - |x|) = 0$, although neither factor is 0.

While Richardson's result is a theoretical restriction on the effectiveness of our test for algebraic equivalence, in practice for learning and teaching such tests work very well indeed on the limited range of expressions which arise as answers to typical assessments. As (Fenichel 1966) comments "*recursive undecidability can be a remote and unthreatening form of hopelessness*".

Many systems, including STACK, make use of computer algebra in this way. What then, does "simplify" mean? This is a phrase which is in common currency throughout elementary teaching we now argue that this is ambiguous and is often used for the opposite mathematical operations. For example, 1 is simpler than 7^0 but $7^{7\text{^}(10)}$ is probably simpler than writing the integer it represents. There are many algebraic examples, e.g. compare the expanded and factored forms of $x^{12} - 1$ and $(x - 1)^{12}$. Similar examples can be found for many other pairs of forms, e.g. single fractions and partial fractions, or various trigonometrical forms. It is not difficult to find examples in textbooks. The word "simplify" may mean little more than *transform* an expression into an equivalent, but perhaps unspecified form.

Fitch (1973) gave three reasons for simplifying expressions, the last of which was deciding if an expression is identically zero. The first is what he calls compactness of expressions, to make the expression smaller and this idea can be found in older writers, for example (Babbage 1827, p. 339) comments

> whenever in the course of any reasoning the actual execution of operations would add to the length of the formula, it is preferable to merely indicate them.

Or further back (Euler 1990, x50), "*[...] the simplicity of the equation expressing the curve, in the sense of the number of terms.*" Designers of contemporary CAA have also reached this conclusion, e.g. (Beevers et al. 1991, p. 113)

> It has long been accepted in science that "the simplest answer is the right one". We have translated this premise into "the shortest answer is the right one".

For these CAA designers the length of the representation was a key property of an expression. Of course, compactness is strongly related to the way in which information is represented, so this measure only makes sense in a particular context. Simplicity can also be interpreted as the ease with which calculations can be carried out. This view was developed by (Moses 1971):

> Of course the prevalence in algebraic manipulation systems of simplification transformations which produce smaller expressions is due mostly to the fact that small expressions are generally easier to manipulate than larger ones.

The second reason (Fitch 1973) gives for simplifying expressions, also discussed by Fenichel (1966), is intelligibility. That is making it easier for users to understand. It is not immediately clear that compactness and intelligibility are different. As one example, consider replacing trigonometric functions by complex exponentials. Using these we remove the need for any trigonometric identities. The formal rules $e^x e^y = e^{x+y}$, $(e^x)^y = e^{xy}$ and $e^0 = 1$ suffice. In this process we also remove the redundancy in using tan, cosec etc. and a plethora of separate rules. Hence, these transformations render expressions much easier for the machine to manipulate, with fewer rules and fewer operations. A user, on the other hand, may expect their answer in terms of these traditional trigonometric forms rather than as complex exponentials. *Ease of computation* and *intelligibility* are different issues.

Notice here the first issue we have to address, i.e. whether the teacher's expression is equivalent to the student's, immediately raises very interesting theoretical issues in computer science, and implicitly raises pedagogic issues. What is a student to make of the instruction to "simplify"? Is it any wonder some of our students remain perpetually confused? Appreciation of this potential ambiguity of 'simplify' suggests we develop a much more sophisticated vocabulary with which to talk about algebraic operations and the senses in which two expressions can be compared. Others, e.g. (Kirshner 1989) agree with this need.

> This analysis, we believe, points the way to a new pedagogical approach for elementary algebra, an approach that requires syntactic and transformational processes to be articulated declaratively, enabling more, rather than fewer, students to escape from the notational seductions of nonreflective visual pattern matching (Kirshner 1989, p. 248).

The approach in STACK is to enable the teacher to specify many senses in which two expressions might be the same or different, and separately to enable the teacher to test whether an expression is written a number of forms. To do this, STACK provides the user with a number of answer tests. Testing for properties is

significantly different than performing calculations, and so requires specific computer algebra functionality to enable this. Inequalities, equations, and particularly systems of polynomial equations (see e.g. Badger and Sangwin 2011) all have interesting elementary mathematical issues of this type which teachers, and students would benefit from appreciating more deeply.

Question Models

In many situations the teacher will seek to establish more than one property and so need a mechanism by which a number of tests can be applied, perhaps in a specific order, and outcomes assembled on the basis of the results. An example STACK question is shown in Fig. 2 which was originally written by Vesa Vuojamo at Aalto University, Finland [see (Rasila et al. 2010) for details of their use of STACK] This question asks students to find the polynomial $p(x)$ which makes the function $f(x)$ continuously differentiable.

In this case the correct answer is the cubic spline which is unique up to algebraic equivalence. However, in practice STACK actually establishes five separate properties and provides separate feedback in each case. These properties establish that the student's answer is a cubic in x; that the polynomial passes through the points

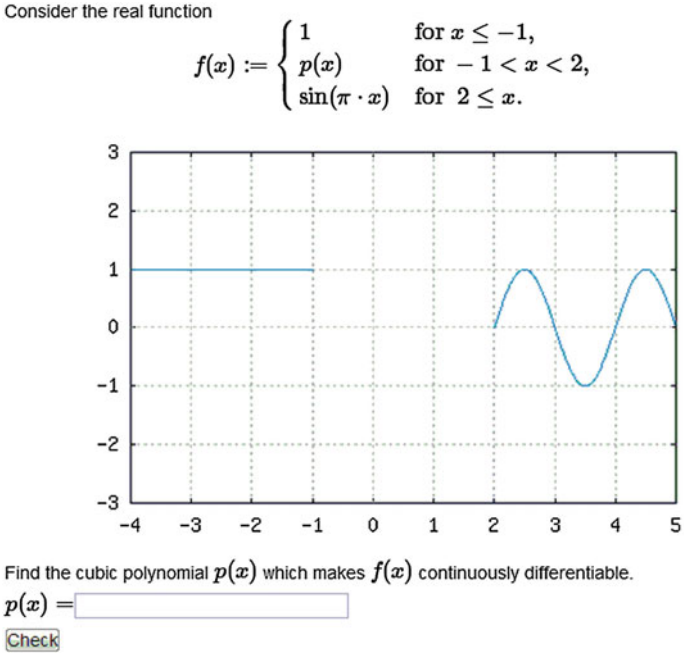

Fig. 2 Testing for individual properties in STACK

$p(-1) = 1$ and $p(1) = 0$, and the derivative of the student's answer matches $f(x)$ at $x = -1$ and $x = 1$. For brevity we have omitted a screen shot of this automatically generated feedback. Notice the significant shift here between "comparing the student's answer with the right answer" and articulating the properties needed separately. This is the central issue for the teacher.

An important goal of the STACK project was to enable teachers to write their own questions. Previous systems, in particular the AiM system of (Strickland 2002), forced the question author to become a computer programmer, in effect writing substantial pieces of computer algebra code to generate the required response processing commands for each question. While AiM does have an authoring system, many other CAA implementations do not: each question is essentially a bespoke computer programme making it impossible for anyone other than an expert developer to write questions. By providing answer tests which enable specific properties to be established we both reduce the amount of code and articulate what is intended in a way which is much clearer than expressions such as simplify (SA – TA). There is also an important conceptual shift needed here by the teacher, who must specify properties explicitly, and not use proxies for those properties. Experience has demonstrated, however, that writing reliable, valid questions remains a difficult task, requiring expertise.

While early CAA pioneers had to write everything from scratch, this did provide great freedom with respect to the underlying *interaction model* which the students are forced to use. Essentially the "model" is the flow-chart through which user's interactions change the internal state of the system until an end point is reached. For example, how many "attempts" can a user make? What form do these attempts take? What response does the system make and to what extent can a particular teacher make choices? How are questions sequenced, e.g. does the student see a fixed quiz at the outset, or are questions sequenced in an adaptive way as in the DIAGNOSYS system of Appleby et al. (1997)?

As we said before, the attempt to encode something forces you to be very specific about the details of what you are trying to do. We have discussed one example, "simplify", at the level of computer algebra. Next we consider one example at the level of the question model.

In moving the development of STACK from version 1 to version 2 we (somewhat naively) assumed that there would be a clean separation of the "question" from the "quiz". That is, it would be possible to "insert questions" into a more general quiz structure in a flexible way. This turns out to be exceedingly difficult to do, which was a significant and unexpected surprise. For example, many teachers using CAA for formative assessments will ask a group of students to complete a "quiz" of a predetermined number of questions each week. Formative feedback is available, and where necessary multiple attempts are encouraged to help students ultimately succeed in these tasks. However, there is a strict time limit, after which further attempts are prevented and the teacher's model solutions become available. This model of interaction essentially replicates traditional teaching in an online manner, which may or may not be efficacious. Notice that the concept of "due date"

is a property of the "quiz", but availability of the "worked solution" is a property of the "question". It is simply impossible to divorce the two cleanly.

In STACK version 3, we have opted to provide a question type for the quiz system in Moodle. In these recent developments the designers of Moodle have provided much greater flexibility in the separation of "question" and "quiz" by the use of "behaviours", which enable the better integration of the model used by STACK which we have designed for use with a wide variety of types of mathematical questions.

When CAA "question types" are relatively simple, e.g. multiple response or numeric input and even when they are confined to single answers, a variety of models for interactions are available which are also relatively straightforward to understand. However, in mathematics, especially when we aim to accept (i) mathematical expressions as answers, or (ii) multi-part questions, then the situation becomes much more complex. Any model must provide interactions which the student clearly understands at each stage. There should be no doubt as to the consequences of each action with the system. In this way it should not raise concerns which distract from the actual mathematics. Teachers must also be able to author questions confident that forms of use are available which are sensible.

STACK implements multi-part mathematical questions. Figure 3 shows an example of a relatively elementary calculus question. Notice that all three parts refer to a single randomly generated mathematical object. Hence, we cannot really claim this is three separate questions. Furthermore, recall that unlike early CAA systems which were application software, students interact with STACK through a web browser. There is already an implicit interaction model here, which requires the student to submit a whole page at once. I.e. asking for individual parts to be marked separately is difficult. Indeed, if a student changes all parts, but only asks for one to be marked, then the model becomes quite intricate. What should the system do with data which has changed or input fields which are empty? Notice that we have used the word "part" without a definition. To the student, there are three inputs and so they might perceive this as having three parts. To the teacher, the first and second parts are linked, so are they separate parts or one?

In STACK a key design feature is a total separation of the inputs, into which students enter their answers, and potential response trees, which are the algorithms through which the mathematical properties are established. Response trees may rely on inputs in an arbitrary fashion, e.g. one-one or many-one.

Tied to the inputs is a concept of *validity*. The prototype input is an algebraic expression, and we expect the student to enter their answer using a traditional linear syntax into a web form box. Clearly the student needs to match brackets, indeed they need to enter a syntactically valid expression. STACK also enables teachers to permit a less strict syntax, e.g. omitting explicit * symbols for multiplication, where this is unambiguous. However, it is not clear in the twenty first century that this is helpful to students. As (Beevers et al. 1991) commented:

> We would like to make input simpler but have also recognised that restrictions can be advantageous. Most students will be using computers in other areas of their work and will need to learn to adapt to the rigours of computer input. The student is also forced to think much more carefully about the format of their answer since any ambiguity is punished mercilessly. This may be frustrating at first but can lead to a better understanding whereas a written answer may contain an ambiguity which is not recognised by the student and can lead to a misunderstanding later (Beevers et al. 1991).

The design issues associated with syntax were addressed in detail by (Sangwin and Ramsden 2007) with further discussion in (Sangwin 2013). Essentially, an unambiguous informal syntax is impossible, and when combined with international differences in notational conventions the situation becomes hopeless. Some conventions, particularly those for inverse trigonometric functions, are particularly problematic. Hence STACK provides a number of options for the input, which enables a teacher to tailor the meaning to the question and their group of students. Furthermore, the concept of syntactic correctness is only one part of the validation process. In some situations the teacher may wish to reject any floating point numbers, or rational coefficients not in lowest terms, as "invalid" and not "wrong". Not only is there a subtle pedagogic difference (e.g. "Since you have floats I'm not going to think about your answer!") but the way scores are calculated may depend on the number of valid attempts. Hence rejecting answers as invalid avoids penalising students on technicalities while reinforcing issues important to that teacher in a particular situation. All these decisions are at the control of the teacher of course. There are other reasons for invalidating an expression. If a student types an *expression* in place of an *equation* then the system can reject this as invalid, with explicit feedback of course. In practice students do need educating on how to enter their answer, and they take some time to become used to the interface. However, ultimately the majority of our students cease to find the interface especially problematic for the majority of questions. Entering a particularly complex expression is always going to be difficult. Notice however, that validity is significantly more involved that a syntactic check, and that validity is a concept tied to the input. It is separate from the notion of an answer being *correct*.

The algebraic expression is the prototype input, but the separation of inputs enables a variety of other interactions to be implemented. For example, HTML elements such as drop-down lists, "radio buttons" and checkboxes for MCQs have been implemented in a relatively straightforward way. MCQs do have a place, and often these can be combined as multi-part questions with algebraic inputs. Even when used alone, the support of CAS in randomly generating questions enables STACK to provide mathematical MCQs. The HTML text area enables multiple lines to be submitted by a student, although the primary use so far has been to provide more space when entering systems of equations. Systems of equations arise naturally when answering *algebra story problems*, see for example (Badger and Sangwin 2011). Asking a student to transform an algebra story into a system of equations, and then solve these, is a basic and classical mathematical task.

Normally, the student sees some validation feedback tied to the input, as shown in Fig. 2. The first time they submit their answer it is validated, and displayed in a

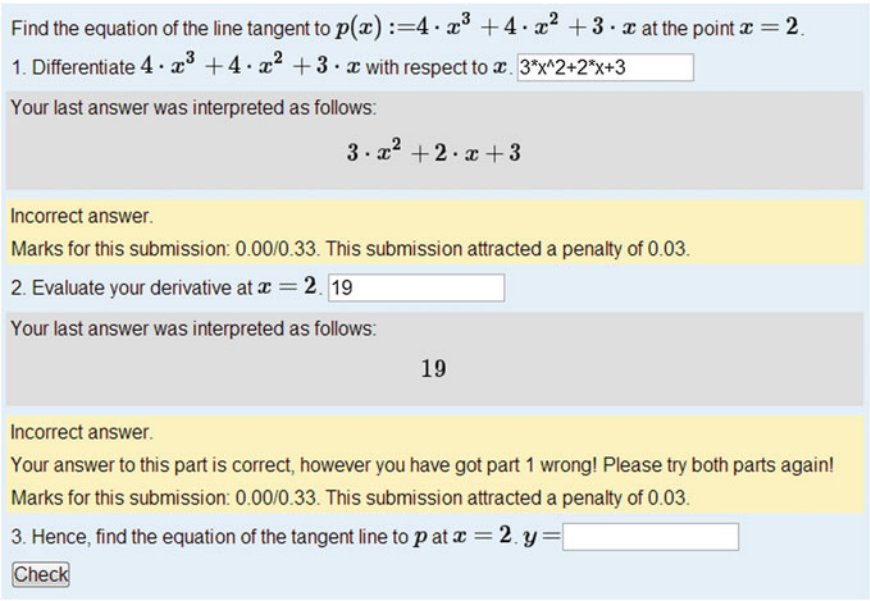

Fig. 3 Follow through marking in STACK

two dimensional format. This *double submission* is actually an artefact of the interaction model imposed by web page forms—there are other interaction models. For example, a student might see their expression build up in a two dimensional format as they type it, with brackets automatically closed, or mismatched brackets highlighted. In Fig. 3 feedback from the inputs showing expressions in two-dimensional traditional notation is separated out from feedback from the potential response trees which have established properties of the answers.

For some inputs, such as drop-down lists or multiple choice interactions, this double submission is irritating to users and the teacher can over-ride it. Another option is to make use of an equation editor to build up the expression in a two dimensional traditional way. Such equations editors are relatively standard in many CAA systems. STACK version 2 makes use of the DragMath editor, written by Alex Billingsly at the University of Birmingham.

The computer environment also enables other kinds of interactions which are not possible in a paper and pencil environment. Dynamic mathematics environments, such as GeoGebra, enable exploration and *mathematical experiments* to be undertaken. As exploration tools for learning, these are rather well established. These kinds of interactions can also be incorporated into assessments. The geometrical configuration of the diagram constitutes the mathematical answer.

One example of a STACK question with this type of interaction is shown in Fig. 4. Here, the student must move the point P by dragging on screen. As this is done, the values of the angles shown update dynamically as the dragging takes

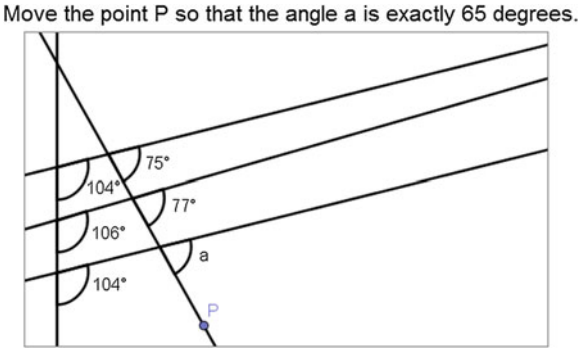

Fig. 4 GeoGebra input interactions in a STACK question

place giving a particular form of immediate feedback. Once the student is satisfied they have the correct answer they can submit the page, and the value of the angle at *a* is returned as part of the answer. Such inputs potentially accompany other interactions in a multi-part question. Extensions to STACK to accommodate interactions such as this were first made at the University of Aalto in Finland, see (Rasila et al. 2010).

Now we turn to establishing the properties of an answer. Each potential response tree is an algorithm which relies on some (at least one) of the inputs. The fundamental requirement is that each input should provide a valid CAS expression. Once all the inputs upon which a potential response tree relies are valid, then the tree can be evaluated.

The potential response tree, technically an acyclic directed graph, consists of nodes with branches. Each node takes two expressions, e.g. the student's input and the teacher's answer, and compares them using a specified *answer test.* As discussed above the answer test might establish algebraic equivalence or one of a range of other properties. On the basis of this test, either the true or false branch is executed. The branches update numerical scores, generate feedback for students, create a "note" for later statistical analysis and then link to the next node or halt. Notice the first three of these actions generate outcomes which correspond broadly to the summative, formative and evaluative functions of assessment. Which outcomes are available to the student is an option which the teacher must choose. In Fig. 2 we have included only textual feedback and numerical scores have not been shown.

This separation between inputs and the algorithms which establish the properties of answers is not an obvious design advantage, however it actually enables many useful situations to be immediately implemented without having to have separate models for each "type" of interaction. The possibility of arbitrary mappings between inputs and outputs, together with an ability to place feedback in any position, enables a richer set of questions to be implemented.

The student in Fig. 3 has chosen to answer the first two parts. Notice that follow-through marking has been implemented, i.e. the student's expression in the first part has been used when marking the second. The student has correctly

evaluated their expression, but actually they need to correct their original error and re-evaluate to correctly answer the whole question.

Immediately, a whole host of questions arise about how the student will interact with this system. Currently in STACK, all the "parts" of the question are visible to the student immediately. The question as phrased in Fig. 3 presupposes a particular method. Of course, other methods for finding tangent lines are perfectly valid. Does the teacher expect a particular method to be used, or are they simply interested in whether the student can find a correct answer using a valid method? Unless steps are laid out, to what extent can CAA assess which method is used?

From the outset, CALM system of Beevers et al. (1991) made a serious attempt to automate the assessment of steps in students' working. The interaction model they developed still had pre-defined templates through which students answered a question. However, in their system steps were revealed in a number of possible ways. For example, a student could ask for steps (possibly sacrificing some implied method marks as a result), or this could be triggered automatically if a student was unable to complete the whole question unaided. This kind of interaction model has been successful, and is widely used, e.g. see (Ashton et al. 2006). The STACK system has been modified and extended at Aalto University, Finland and is in regular use by large groups of engineering students. Their extensions include "question blocks" which can be revealed by correct or incorrect responses, providing very similar mechanisms to CALM. It is likely that a combination of the interaction model used by CALM, the mathematical sophistication underlying STACK, and the depth of adaptive design of DIAGNOSYS (see Appleby et al. 1997) will ultimately combine into a single CAA system.

Students' Reactions to STACK

The most important issue is students' reactions to tools such as STACK. Their reaction includes a number of aspects, such as their level of engagement, their reported affective reaction and ultimately their achievement. CAA tools do not operate in isolation: they are part of the whole experience. Clearly tools could be used in inappropriate, or even harmful ways, or they could simply replicate traditional teaching with little or no examination of the strengths and weaknesses relative to traditional methods. The difficulty in evaluating the effectiveness of such tools is isolating a specific effect from the general teaching. We also need to take account of the potential for innovator/novelty effects which might exaggerate the actual long term benefits. Of course, if a genuine benefit exists, even temporarily, some students will have taken advantage of this.

A common use of CAA at the university level is to automate routine practice exercises alongside traditional paper based tasks. This enables students to receive immediate feedback on tasks which assess competencies with manipulative skills. Teachers no longer have to undertake repetitive marking of such work by hand. This was our motivation for introducing CAA to the University of Birmingham

mathematics degree programme in 2001. In the majority of situations where this kind of activity takes place, colleagues report a strong correlation between engagement with, and success on, STACK-based formative exercise questions and final marks in traditional exams. For typical data, see (Rasila et al. 2010). This is not surprising, and corresponds with the author's experience at the University of Birmingham. However, evidence and appreciation of this reinforces the ongoing need for regular monitoring of student activity to identify and support students who are not engaging with the online activities. These students have a high probability of ultimately failing the course. Many university courses have very large student groups, and with inevitable delays in marking paper-based formative work and collating such marks it is otherwise difficult to monitor such students individually.

To try to evaluate STACK we have undertaken focus groups with students to ask specifically about their experiences and reactions. The author has undertaken such focus groups, both at the University of Birmingham and with students at other institutions using STACK. A semi-structured interview provides a freedom to follow up themes or concerns in a way a paper-based questionnaire does not. From these interviews, some consistent themes emerge. Capital letters refer to individuals, although the quotations below are representatives from more than one focus group session.

Syntax is initially a problem

> A: I agree, but when you get used to STACK all that goes away [B: "yes"] but when you start that is a problem. It is very annoying when you try to type something… well you have the "check syntax", but if the check syntax is always incorrect you are like !!!

This is particularly problematic if a diagnostic test is the student's first experience of a university course. It is relatively common for students to sit such a test at the start of their course, Lawson (2003). However, the novelty of the university setting and unfamiliarity with the syntax combine to make this an unhappy experience.

> C: in the beginning, September or something like that, […] I think it was really annoying to use it then because you didn't know how to write it down, the syntax, […] You had a time to do the problems and it was very annoying that it said "wrong, wrong" all the time when it was the syntax. […] I didn't know how to write it down, so I got the wrong answer.

It was clear that it was syntax which was a barrier here. For one student prior knowledge removed this problem.

> B: Yes, I took this test but because I have a little bit of background of computer programming so I […] knew the syntax a bit. I was more frustrated because I didn't know the answers myself! [C: laughs] So I guess I have time to deal with the real mathematical problems, so I guess my frustration is based on my own lack of mathematical knowledge. So, I think the test worked quite well for me. But there you have it, I had some background with things like this.

Ultimately the syntax is learned in a relatively short space of time by the majority of our students. While there are differences between mathematical input notations, many systems share a common core of notational conventions. This in

itself constitutes a valuable skill, as identified by Beevers et al. (1991), quoted above. However, it needs to be addressed specifically. I.e. we need to teach students how to express themselves unambiguously using this syntax. The feature most appreciated by students is the immediate feedback.

> C: Yes, I think it is good. Because of the feedback. […] with the paper you have to wait and then when you see the right answers you can look through those with the teacher probably too quickly, and you can't take your time to understand, but with STACK you can take your own time with those exercises. So that is the good thing with them.

Of course, a student could simply use a textbook for this. When specifically asked about this students responded

> A: yeah, yes but then you look at the answer before you have solved the problem. STACK won't tell you!
> B: it is a bit of cheating.
> A:You don't learn if you just go ahead and look at the back. And usually when we have homework during the course from the book they are usually problems you don't have the answers to so you can't find out if you are wrong or right.

When combined with the random questions, particularly when the teacher encourages repeated practice, this gives an interesting environment for self-motivated repeated practice. Students can try questions, respond to the feedback, perhaps look at complete worked solutions and then work on new problems from the same template. The use of random problems for each student provides a behaviour which does not occur when using fixed and static books.

> D: The questions are of the same style and want the same things but they are subtly different which means you can talk to a friend about a certain question but they cannot do it for you. You have to work it all out for yourself which is good.

This view was appreciated by other students on another occasion.

> B: I think one of the best things about STACK was the way it created the values, or the problems, for, like, meant for you. But they are still the same as your friend has so you can, like, collaborate on them and do some team work, and work on the difficulty with your friends, but you still have to do the exercise for yourself [A: yeah!] you have values and
> A: so you can't just copy!
> B: it won't help if you just copy the answer from your friend.

Notice here the appreciation of the surface variation in the context of an underling structure. In many situations, particularly in mastery of technique, the purpose of routine exercises is precisely to enable students to reach a point where they can recognise and successfully complete any problem from a particular class. Students who use STACK exercises at universities for mathematics are often amongst the highest achieving in their generation. It is not surprising to find recognition and appreciation of the mathematical sophistication.

> E: Recognising the turning points of the functions produced in question 2 was impressive, as there are a lot of functions with stationary points at x = 1 and it would be difficult to simply input all possibilities to be recognised as answers.

This was in response to a question such as show in Fig. 1 where the student is asked to find examples. In many situations the properties required by a correct answer can be established automatically, although it would be time consuming and somewhat tedious for a teacher to do so by hand. These kinds of questions have been widely discussed, e.g. (Watson and Mason 2002), but they reinforce the fundamental issue for CAA: the teacher must articulate the specific mathematical properties which an answer should satisfy.

Conclusions

Notice, however, the fundamental challenge remains here. We, as yet, have very few effective tools to encode a complete elementary mathematical argument on a machine. This document was originally prepared in LaTeX, which despite the steep learning curve still sets the standard for quality of mathematical *typesetting*. It was then converted to MSWord, which has very poor support for mathematics. What we cannot do is easily encode the meaning of an expression, and combine this with simple logic and automatic CAS calculations. Until we can achieve this simple interface, marking student's extended work automatically will be impossible. Given the theoretical difficulties of establishing equivalence of two expressions, establishing the validity of whole arguments automatically appears totally hopeless. For example, in finding the tangent line to answer the question posed in Fig. 3, the student could simply find the remainder when the polynomial is divided by $(x-2)^2$. The remainder after polynomial long division of $p(x)$ by $(x\text{-}a)^2$ always yields the tangent line at $x = a$ (see (Sangwin 2011) for details and other methods) without using calculus. Did the teacher want the answer using *a* valid method or using *the* method as taught? If we seek to automatically assess the working without providing a template this issue must be addressed.

Despite the fact that it is impossible to assess extended working, CAA is routinely used by thousands of students in many settings. These students and their teachers find many aspects of CAA very helpful. The ability to generate random questions, the immediacy of feedback and the detailed reporting are all cited as benefits. In particular, the use of randomly generated questions to enable discussion, and the ability to assess example generation tasks where the answers are difficult for a teacher to mark, are affordances which are unique to CAA. It is clear, at least to the author, that uptake of CAA will increase, in informal and self-directed situations and in formative settings. It is highly likely that CAA will become used in high-stakes examinations. In mathematics we do have objective notions of *correctness* and the progressive automation of mathematical knowledge provides our subject with an opportunity to implement valid assessment which are not apparent in essay or more subjective artistic disciplines. Hence, we have a responsibility to ensure the tools we use move beyond multiple choice questions or primitive *string match* to check expressions.

I end this paper with two comments. Firstly, for the arguments we encounter in many areas of elementary mathematics the theoretical difficulties do not arise: we can automatically decide if they are correct, or not (Beeson 2003). Secondly, automatic tools can be combined to establish the correctness, or otherwise, of parts of an argument. For example, a routine calculation within a longer proof can be checked automatically. It would be potentially very helpful to a student and teacher to have this confirmed automatically before the whole piece of work is submitted to an intelligent human marker. This semi-automatic approach, a pragmatic combination of human and automatic marking, seems to offer the most promising direction for future effort in computer aided assessment.

References

Appleby, J., Samuels, P. C., & Jones, T. T. (1997). DIAGNOSYS—a knowledge-based diagnostic test of basic mathematical skills. *Computers in Education, 28*, 113–131.

Ashton, H. S., Beevers, C. E., Korabinski, A. A., & Youngson, M. A. (2006). Incorporating partial credit in computer-aided assessment of mathematics in secondary education. *British Journal of Educational Technology, 27*(1), 93–119.

Babbage, C. (1827). On the influence of signs in mathematical reasoning. *Transactions of the Cambridge Philosophical Society, II*, 325–377.

Badger, M., & Sangwin, C. (2011). My equations are the same as yours!: Computer aided assessment using a Gröbner basis approach. In A. A. Juan, M. A. Huertas, & C. Steegmann (Eds.), Teaching mathematics online: Emergent technologies and methodologies. IGI Global.

Beeson, M. (2003). *The Mechanization of mathematics*. In Alan Turing: Life and legacy of a great thinker. (pp. 77–134). Berlin: Springer.

Beevers, C. E., Cherry, B. S. G., Foster, M. G., & McGuire, G. R. M. (1991). *Software tools for computer aided learning in mathematics*. Avebury Technical.

Burton, R. R. (1982). Diagnosing bugs in a simple procedural skill. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 157–183). Academic Press.

Caviness, B. F. (1970). On canonical forms and simplification. *Journal of the ACM, 17*(2), 385–396.

Euler, L. (1990). Introduction to analysis of the infinite (Vol. II). Springer. (Translated by Blanton, J. from the Latin Introductio in Analysin Infinitorum, 1748).

Fenichel, R. R. (1966). An on-line system for algebraic manipulation. Ph. D thesis, Harvard Graduate School of Arts and Sciences.

Fitch, J. (1973). On algebraic simplification. *Computer Journal, 16*(1), 23–27.

Hollingsworth, J. (1960). Automatic graders for programming classes. *Communications of the ACM, 3*(10), 528–529.

Kirshner, D. (1989). The visual syntax of algebra. *Journal for Research in Mathematics Education, 20*(3), 274–287.

Lawson, D. (2003). Diagnostic testing for mathematics. LTSN MathsTEAM Project.

Matiyasevich, Y. (1993). *Hilbert's tenth problem*. Cambridge: MIT.

Moses, J. (1971). Algebraic simplification a guide for the perplexed. *Communications of the ACM, 14*(8), 527–537.

Noss, R., & Hoyles, C. (1996). Windows on mathematical meanings: Learning cultures and computers. Berlin: Springer.

Papert, S. (1980). Mindstorms: Children, computers and powerful ideas. Harper Collins.

Rasila, A., Havola, L., Majander, H., & Malinen, J. (2010). Automatic assessment in engineering mathematics: Evaluation of the impact. In *Reflektori 2010: Symposium of engineering education*, Aalto University, Finland.

Richardson, D. (1966). Solvable and unsolvable problems involving elementary functions of a real variable. Unpublished doctoral dissertation, University of Bristol.

Sangwin, C. J. (2010). Who uses STACK? A report on the use of the STACK CAA system (Tech. Rep.). The Maths, Stats and OR Network, School of Mathematics, The University of Birmingham.

Sangwin, C. J. (2011). Limit-free derivatives. *The Mathematical Gazette, 534*, 469–482.

Sangwin, C. J. (2013). *Computer aided assessment of mathematics*, Oxford: Oxford University Press.

Sangwin, C. J., & Grove, M. J. (2006). STACK: addressing the needs of the "neglected learners". In *Proceedings of the First WebALT Conference and Exhibition January* 5–6, Technical University of Eindhoven, Netherlands (pp. 81–95). Oy WebALT Inc, University of Helsinki, ISBN 952-99666-0-1.

Sangwin, C. J., & Ramsden, P. (2007). Linear syntax for communicating elementary mathematics. *Journal of Symbolic Computation, 42*(9), 902–934.

Sleeman, D., & Brown, J. S. (Eds.). (1982). *Intelligent tutoring systems*. Academic Press.

Strickland, N. (2002). Alice interactive mathematics. . *MSOR Connections, 2*(1), 27–30.

Suppes, P. (1967). Some theoretical models for mathematics teaching. *Journal of Research and Development in Education, 1*, 5–22.

Watson, A., & Mason, J. (2002). Student-generated examples in the learning of mathematics. *Canadian Journal for Science, Mathematics and Technology Education, 2*(2), 237–249.

Wester, M. (1999). Computer algebra systems: A practical guide. Wiley. Chapman, O. (2003). Facilitating peer interactions in learning mathematics: Teachers' practical knowledge. In M. J. Hψines & A. B. Fuglestad (Eds.), *Proceedings 28th Conference of the International Group for the Psychology of Mathematics Education* (Vol. 2, pp. 191–198). Bergen, Norway: PME.