Arithmetic, Logic, Syntax and MathCheck

Antti Valmari* and Johanna Rantala
Faculty of Information Technology, University of Jyväskylä, Jyväskylä, Finland

Keywords: Computer-aided Education, Mathematics Education, Computer Science Education.

Abstract: MathCheck is a web-based tool for checking all steps of solutions to mathematics, logic and theoretical computer science problems, instead of checking just the final answers. It can currently deal with seven problem types related to arithmetic, logic, and syntax. Although MathCheck does have some ability to perform symbolic computation, checking is mostly based on testing with many combinations of the values of the variables in question. This introduces a small risk of failure of detection of errors, but also significantly widens the scope of problems that can be dealt with and facilitates providing a concrete counter-example when the student's solution is incorrect. So MathCheck is primarily a feedback tool, not an assessment tool. MathCheck is more faithful to established mathematical notation than most programs. Special attention has been given to rigorous processing of undefined expressions, such as division by zero. To make this possible, in addition to the two truth values "false" and "true", it uses a third truth value "undefined".

1 INTRODUCTION

MathCheck is a web-based program for giving students feedback on their solutions to mathematics, logic and theoretical computer science problems in elementary university courses. Compared to well-known systems such as STACK (Sangwin, 2015; STACK, 2017), MathCheck has three distinctive features: it gives feedback on all steps of the solution, instead of just the final answer; it can deal with some novel problem types; and it features many commands with pedagogical motivation. Although an examination version of MathCheck exists and has been used in 2017 and 2018, MathCheck has been designed not for giving points but for providing feedback, very much in the spirit of (Gibbs and Simpson, 2004; Gibbs, 2010).

As an example of the first feature, assume that the student has been asked to simplify $\cos^2\omega - \cos 2\omega$.[1] Forgetting parentheses when applying $\cos 2x = \cos^2 x - \sin^2 x$, the student types

```
cos^2 om - cos^2 om - sin^2 om
= -sin^2 om
```

as the solution.

MathCheck yields the feedback shown in Figure 1. It shows the starting point $\cos^2\omega - \cos 2\omega$ in

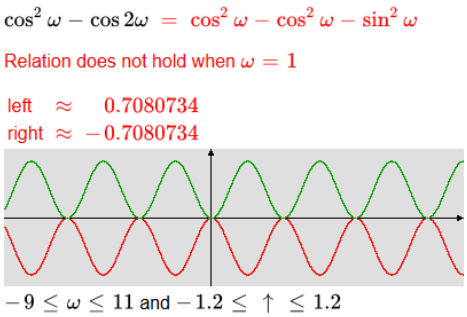


Figure 1: An example of arithmetic mode feedback.

black, an equals sign, and the first part of the student's answer. The latter two are shown in red, because the equality does not hold. Next comes a counter-example to the equality. Finally, graphs of the functions on the left and right hand side of the invalid equality are shown as curves. The last part `= -sin^2 om` of the student's solution is not processed, because MathCheck stops at the first error that it encounters.

It is clear from the feedback that there is a sign error. (Of course, finding the cause of an error is not always this easy.) The student fixes the error by adding the forgotten parentheses. Not yet willing to re-think the rest of the solution, the student removes it:

```
cos^2 om - (cos^2 om - sin^2 om)
```

Now MathCheck replies with $\cos^2\omega - \cos 2\omega = \cos^2\omega - (\cos^2\omega - \sin^2\omega)$ and the remark "The complexity of the final expression is 14, while it must be at most 5." The remark is in magenta, because the an-

*Most of the work was done while the author was with Tampere University of Technology, Finland.

[1]The reader is invited to try this at http://users.jyu.fi/∼ava/C19simplify.html. The reader may edit the answer in the answer box.

swer is correct in the mathematical sense, but is longer than the maximum length stated by the teacher.

The equals sign is in green, because in this case MathCheck is able to *prove* that the equality holds. When MathCheck is unable to prove an equality, MathCheck just tests it with many value combinations of the variables in question. This involves the risk that a wrong answer goes undetected, by matching the correct one in all test cases. Fortunately, for reasons discussed in Section 3, this risk is most of the time so small that it is not a serious problem. In the case of inequalities, MathCheck also applies a numeric hill-climbing method. Therefore, it finds that, for instance, `assume x > 0; x^2 / 10 < 1000 + x log x` does not hold. (This and many other small examples of this study can be tried at http://users.jyu.fi/∼ava/C19others.html).

Finally the student adds `= sin^2 om` to the end of the solution. MathCheck replies with $\cos^2\omega - \cos 2\omega = \cos^2\omega - (\cos^2\omega - \sin^2\omega) = \sin^2\omega$ and reports that "No errors found. MathCheck is convinced that there are no errors."

As an example of a novel problem type, the student was asked to write a predicate saying that H is a palindrome, where H is an array that is indexed from 0 to $l-1$.[2] The student writes

```
AA i; 0 <= i < l:  H[i] = H[l-i]
```

to which MathCheck replies that

model-answer $\Leftrightarrow$ $\forall i; 0 \le i < l : H[i] = H[l-i]$
Relation does not hold when $H = [0]$ and $l = 1$
left = T
right = U

It means that in the case of the array that consists of one element 0, the model solution given by the teacher yields T meaning true, but the student's solution is undefined, denoted with U. Indeed, $\forall i; 0 \le i < l$: tries only the value $i = 0$, because $l = 1$. With it, $H[l-i]$ reduces to $H[1]$, which is undefined, because 1 is outside the range from 0 to $l-1$, that is, from 0 to 0. MathCheck deems the following answer as correct:

```
AA i; 0 <= i < l:  H[i] = H[l-i-1]
```

MathCheck checks the mathematical meaning of the answer instead of the precise written form. Therefore, MathCheck also accepts the following as correct:

```
!EE n:  n >= 0 /\ n < l /\ H[n] !=
H[l-1-n]
```

That is, $\neg\exists n : n \ge 0 \wedge n < l \wedge H[n] \neq H[l-1-n]$. Also $\forall i; 0 \le i < l : H[i] \le H[l-1-i]$ is accepted, but $\forall i; 0 < i < l : H[i] \le H[l-1-i]$ yields the counterexample $H = [1,0]$ and $l = 2$. This is because the former tests both $H[0] \le H[l-1]$ and $H[l-1] \le H[0]$, but the latter does not test $H[0] \le H[l-1]$.

Restricting the maximum length of the final answer is an example of a pedagogically motivated command that the teacher may use. There are also commands for banning the use of chosen operators in the final answer, requiring that it is in a certain form such as a product of sums, and affecting the feedback that MathCheck gives.

The development of MathCheck began in January 2015. The programming has been a one-person part-time project. Pedagogical experiments have been performed by him and by mathematics teachers and bachelor's or master's thesis authors in three universities or schools.

MathCheck has been reported in (Valmari, 2016; Valmari and Kaarakka, 2016). However, that version only had what is called *arithmetic mode* in this study, and also the arithmetic mode has been improved since then. Among other things, the presentation of graphs of the left and right hand sides of an incorrect (in)equality has been added.

The present study focuses on features that MathCheck offers but most other mathematics pedagogy tools lack. In Section 2, the problem modes of MathCheck are introduced. In half of them, MathCheck checks the solution by incomplete testing. The reasons and consequences of this are discussed in Section 3. When programming MathCheck, two issues required performing original technical research: faithful processing of the syntax of arithmetic expressions as used in everyday mathematics (as opposed to most mathematical software), and the rigorous treatment of undefined expressions (or partial functions) in logic. These are dealt with in Sections 4 and 5. This study is concluded in Section 6.

2 PROBLEM MODES

2.1 Arithmetic Mode

The arithmetic mode was illustrated in Section 1. In it, MathCheck checks chains consisting of expressions and the relation symbols $<$, $\le$, $=$, $\ge$, and $>$ by testing each (in)equality with many combinations of values of the variables in question. MathCheck has the familiar basic arithmetic operators and functions, excluding the inverse trigonometric functions. MathCheck has neither summations $\sum_{i=1}^{n}$, limits $\lim_{x\to}$, nor integrals $\int$, because checking expressions by testing does not work well enough with them, because they are hard to evaluate numerically. MathCheck

[2] This example can be tried at http://users.jyu.fi/∼ava/C19array.html.

Table 1: Results on a pedagogical experiment.

	< 1 hour		≥ 1 hour	
	n	points	*n*	points
Wolfram Alpha	19	7.2	31	8.2
MathCheck	26	7.1	30	9.7

has derivatives with respect to any real-valued variable. To keep the number of value combinations used in testing reasonably small, MathCheck only allows three simultaneous variables.

Because of pedagogical reasons, MathCheck takes the issue of undefined expressions seriously, but because of technical reasons, not perfectly. For instance, given `x/x = 1` or `1/(x^2) >= 0` , MathCheck replies that the relation does not hold when $x = 0$, because then the left hand side is undefined but the right hand side yields 1 or 0. However, MathCheck does not systematically try to find situations where one side is undefined and the other side is not.

Value combinations can be ruled out by writing `assume` *condition* `;` to the front of the relation chain. For instance, MathCheck deems `assume x != 0; 1/(x^2) >= 0` correct. All basic propositional logic operators can be used in the condition. If the condition is too exclusive, then MathCheck fails to detect errors. For instance, MathCheck does not find any counter-example to `assume x > 100; x = 0` .

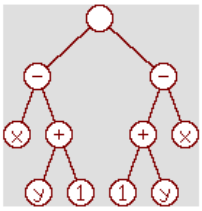
In an experiment by Terhi Kaarakka and her student Veera Hakala (Hakala, 2016), 106 students first solved 10 exercises. Some were told to use MathCheck for help (the old version mentioned in Section 1), while the rest were told to use Wolfram Alpha. The students were asked how much time they spent with the program. Then there was a small examination with a maximum of 16 points. Table 1 shows the average number of points each group received in the examination. Using resampling, we found that the difference between the two MathCheck groups is significant with $p = 0.02$.

2.2 Equation Mode

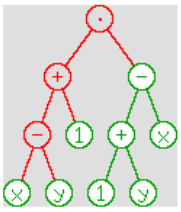
In the equation mode, the teacher gives MathCheck an equation on one variable. The teacher also gives its roots or an indication that the equation has no roots. The task of the student is to solve the equation.

In the example at http://users.jyu.fi/~ava/C19equation.html, the teacher has written

```
equation
x=0 \/ x=pi/3 \/ x=pi \/ x=4pi/3
ends
/*`x` must be at least `0` and less
than `2 pi`*/
```



model-answer



$(x - y + 1) \cdot (1 + y - x)$

Your last tree is not the same as the teacher's tree

Figure 2: An example of equation mode feedback.

```
f_nodes 25
0 <= x < 2pi /\
2 sin^2 x = sqrt(3) sin 2x /**/
```

The structures `/*...*/` are comments that are shown on the feedback page. An empty comment `/**/` causes a line break. Inside comments, text enclosed by ` is shown as mathematics. The command `f_nodes 25` sets an upper limit to how complex the final solution is allowed to be, counted as the number of nodes in the expression tree (see Section 2.3).

The next lines specify the equation and that only the roots that are at least 0 and less than 2π are taken into account. So the equation $2\sin^2 x = \sqrt{3}\sin 2x$ can be presented as a problem although it has infinitely many roots.

Figure 2 shows the feedback to an answer by the student. In its last step, the student has replaced $\sin x = 0$ with $x = 0$, failing to notice that also $\sin \pi = 0$ and $0 \le \pi < 2\pi$. The student fixed the problem. The rest of the feedback after $0 \le x < 2\pi \wedge (\sin x = 0 \vee \sin x = \sqrt{3}\cos x)$ became the following:

$\Leftrightarrow 0 \le x < 2\pi \wedge (\sin x = 0 \vee \sin^2 x = 3\cos^2 x)$
$\Leftrightarrow 0 \le x < 2\pi \wedge (\sin x = 0 \vee \sin^2 x = 3(1 - \sin^2 x))$
$\Leftrightarrow 0 \le x < 2\pi \wedge (\sin x = 0 \vee 4\sin^2 x = 3)$
$\Leftrightarrow 0 \le x < 2\pi \wedge (\sin x = 0 \vee \sin x = \frac{\sqrt{3}}{2} \vee \sin x = -\frac{\sqrt{3}}{2})$
$\Leftrightarrow x = 0 \vee x = \pi \vee x = \frac{\pi}{3} \vee x = \frac{2\pi}{3} \vee x = \frac{4\pi}{3} \vee x = \frac{5\pi}{3}$
The equation does not hold when $x \approx 2.094395$
The first expression with this solution may be $0 \le x < 2\pi \wedge (\sin x = 0 \vee \sin^2 x = 3\cos^2 x)$

To be able to exploit the formula $\sin^2 x + \cos^2 x = 1$, the student had squared both sides of $\sin x = \sqrt{3}\cos x$, forgetting that in addition to its roots, $\sin^2 x = 3\cos^2 x$ has also the roots of $\sin x = -\sqrt{3}\cos x$. The student replaces the problematic line with

```
==> 0 <= x < 2pi /\
    ( sin x = 0 \/ sin^2 x = 3 cos^2 x )
/**/
```

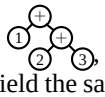
and, after thinking about the signs of $\sin x$ and $\cos x$, replaces

```
original <=> x = 0 \/ x = pi \/ x =
pi/3 \/ x = (4 pi)/3
```

for the last line. MathCheck accepts this solution. The implication sign $\Rightarrow$ tells that one-sided reasoning was applied, so the subsequent equations must have all the roots of the original equation, but they may also have additional roots. The word `original` switches the banishment of additional roots on again. Between $\Rightarrow$ and `original`, repeating $0 \le x < 2\pi \wedge$ is unnecessary.

The example above reveals that the starting point and steps of a solution are not necessarily equations, but more general claims consisting of comparisons put together using propositional operators. For each claim in the student's solution, MathCheck checks (to the extent it can, please see Section 3) that every root given by the teacher also satisfies the claim. If $\Rightarrow$ has not been used or its effect has been cancelled with `original`, MathCheck also checks that each explicit root given by the student satisfies the original claim. The explicit roots in $x = 0 \vee x = \pi \vee \sin x = \sqrt{3}\cos x$ are 0 and π. On the other hand, $0 \le x < 2\pi \wedge (x = 0 \vee x = \pi \vee \sin x = \sqrt{3}\cos x)$ has no explicit roots, because it is pending the reasoning whether $0 \le x < 2\pi$ rules out 0, π, or both.

2.3 Tree Comparison Mode

Mathematicians, logicians, and especially computer scientists often think of expressions as representations of abstract *expression trees*. The expressions $1+2+3$ and $(1+2)+3$ have the same expression tree  , but $1+(2+3)$ has a different tree  , although $(x+y)+z$ and $x+(y+z)$ always yield the same value in mathematics.

In elementary schools, pupils are taught that in the absence of parentheses, multiplication is evaluated before addition. That is, $x+yz$ is interpreted like $x+(yz)$ and not like $(x+y)z$. In programming languages, the same is expressed by saying that multiplication has higher *precedence* than addition. For instance, C++ had 18 precedence levels already in

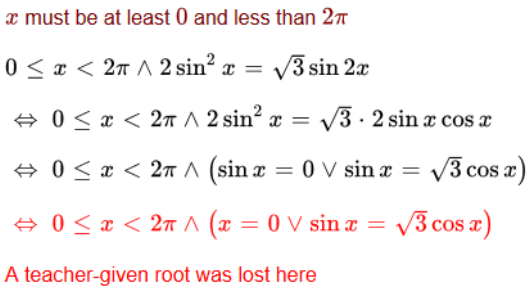


Figure 3: An example of tree comparison mode feedback.

1997 (Stroustrup, 1997). Precedence does not resolve whether, for instance, $8-5-2$ should be interpreted like $(8-5)-2$ or like $8-(5-2)$, because subtraction obviously has the same precedence with itself. Subtraction and most other operators are *left-associative*, that is, $x \circ y \circ z$ is interpreted like $(x \circ y) \circ z$. However, 2^{2^3} is interpreted like $2^{(2^3)} = 2^8 = 256$ and not like $(2^2)^3 = 4^3 = 64$, so the power operator of mathematics is right-associative.

Although precedence and left- or right-associativity are not complicated ideas, it is the experience of the present authors that some students have problems with them. For instance, left- and right-associativity are sometimes confused with associativity, that is, the property that for all x, y, and z, $(x \circ y) \circ z = x \circ (y \circ z)$. Because left- and right-associativity (and precedence) are syntactic concepts while associativity is a semantic concept, this means that the student does not master the distinction between syntax and semantics. Mathematics teachers often tell that students have difficulties in applying the chain rule in calculus, that is, $\frac{d}{dx} f(g(x)) = \frac{d}{dy} f(y)\big|_{y=g(x)} \frac{d}{dx} g(x)$. The present authors believe that understanding expression trees would help learning to apply the chain rule.

Figure 3 shows feedback by MathCheck to an incorrect answer to the expression tree problem at http://users.jyu.fi/~ava/C19tree.html. On the problem page, the upper expression tree was shown, and the student was asked to write an expression that yields the same tree. The hidden information given by the teacher was `tree_compare (x-(y+1))(1+y-x);` . The student replied `(x-y+1)*(1+y-x)` . The feedback indicates a problem with the first factor.

In mathematics (and unlike many programming languages), $\le$ is neither left- nor right-associative, but *conjunctional* (Gries and Schneider, 1993). That is, $1 \le i \le n$ means neither $(1 \le i) \le n$ nor $1 \le (i \le n)$ but $1 \le i \wedge i \le n$. To emphasize this, MathCheck draws all relation operators in a relation chain as a single tree node that has one more subtrees than there are relation symbols in the node.

The tree comparison mode was implemented in early 2017. The first author used it on a mid-level computer science course. The students were given a series of 16 problems to solve at home. The 19 students who claimed to have solved all or most of them were able to easily solve them again in the class. Every student who collected any credit from any of the numerous voluntary exercises available in the course did at least these tree comparison problems.

2.4 Array Claim Mode

This mode was already illustrated in Section 1. Its pedagogical goal is to support development of precise logical thinking that is important in programming and capturing user requirements (Lethbridge, 2000; Surakka, 2007; Valmari, 2003).

The teacher wrote the following in the example in Section 1:

```
array_claim H[0...l-1]
f_nodes 20
AA i; 0 <= i < l:  H[i] = H[l-i-1]
<=>
```

The first line specifies that the name of the array in question is H and its indices range from 0 to $l-1$, where l is an integer variable. Then the length of the final version of the student's answer is restricted. (The student may develop the answer in steps separated with `<=>`, that is, $\Leftrightarrow$. Only the last version need be short enough.) The model answer by the teacher is $\forall i; 0 \leq i < l : H[i] = H[l-i-1]$.

In one problem, K was indexed from 0 to M, and the student was asked to write a predicate saying that the first, second, and last element exist and are different from each other. During the winter 2016–2017, some students and colleagues of the first author first gave the incorrect answer $M \geq 2 \wedge K[0] \neq K[1] \neq K[M]$. MathCheck gave the counter-example $K = [0,1,0]$ and $M = 2$. It made them realize that although $K[0] \leq K[1] \leq K[M]$ implies $K[0] \leq K[M]$, the same does not happen with $\neq$. Then they solved the problem correctly. A correct answer is $M \geq 2 \wedge K[0] \neq K[1] \neq K[M] \neq K[0]$.

MathCheck checks the student's answer by trying all arrays of size at most 4 whose elements are integers in the range from 0 to 3. The teacher must take this into account when designing problems, so that the checking is capable of revealing errors. For instance, the teacher should not ask to write a predicate saying that the array contains at least one negative element.

2.5 Propositional Logic Mode

In this mode, MathCheck checks reasoning chains built from truth value constants, propositional variables, the propositional operators $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, and the reasoning operators $\Leftarrow$, $\Leftrightarrow$, and $\Rightarrow$. The teacher chooses whether the undefined truth value U is in the logic. The number of simultaneous variables is restricted to 10, facilitating exhaustive checking. So in this mode MathCheck never fails to detect existing errors (assuming, of course unrealistically, that MathCheck contains no programming errors). The teacher can declare that the final formula must be in conjunctive (or disjunctive) normal form.

This mode was implemented mostly as a preliminary step en route to the equation, array claim, and modulo modes, which use propositional and reasoning operators in a wider context. We now discuss an issue that applies also in these other modes.

The distinction between the propositional operators $\rightarrow$ and $\leftrightarrow$ on one hand and the reasoning operators $\Rightarrow$ and $\Leftrightarrow$ on the other hand is often not properly dealt with in textbooks on logic. Consider solving the equation $2x-6=0$. In a first step it is transformed to $2x=6$ and in a second step to $x=3$. In MathCheck and some courses on mathematics, $\Leftrightarrow$ is used as handy notation for expressing the progress of the reasoning: $2x-6=0 \Leftrightarrow 2x=6 \Leftrightarrow x=3$.

When used like this, $\Leftarrow$, $\Leftrightarrow$, and $\Rightarrow$ are not propositional operators that yield truth values. The propositional operator that yields T if and only if both sides yield the same (defined) truth value is $\leftrightarrow$. Although $2x-6=0 \Leftrightarrow 2x=6 \Leftrightarrow x=3$ is correct reasoning, $2x-6=0 \leftrightarrow 2x=6 \leftrightarrow x=3$ does not yield T when $x=0$, because then it reduces to $\mathsf{F} \leftrightarrow \mathsf{F} \leftrightarrow \mathsf{F}$, further to $\mathsf{T} \leftrightarrow \mathsf{F}$, and finally to F. That is, the propositional operators are comparable to arithmetic operators such as $+$ and $\cdot$, and the reasoning operators are comparable to arithmetic relations such as $=$ and $\leq$. Please see (Valmari and Hella, 2017) for more differences between $\Leftrightarrow$ and $\leftrightarrow$.

2.6 Modulo Mode

This mode is chosen with `modulo` M, where M is an integer constant in the range $2 \leq M \leq 25$. In it, MathCheck checks reasonings in the quotient ring whose elements are $\{0,\ldots,M-1\}$. They may consist of reasoning and propositional operators, comparisons, and restricted arithmetic expressions. For instance, sin and ln are not allowed, because they have no natural meaning in the ring. To emphasize that there are no negative numbers, $|x|$ is not allowed. Powers and roots are allowed, but the exponent must be a non-

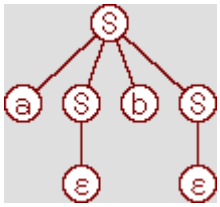


Figure 4: An example derivation tree.

negative integer constant and the degree of the root must be a positive integer constant.

This mode was implemented because it facilitates complete checking of reasoning. It may have pedagogical use when studying modular arithmetic, which is an important topic for computer science students. It has proven very useful on problems outside modulo arithmetic, for instance, when the student is expected to answer with a comparison such as $n < i+1$. Even if the problem is on integers, checking the answer in modulo arithmetic yields appropriate feedback.

2.7 Context-free Grammar Mode

This mode was implemented in autumn 2018. MathCheck can test whether a given string belongs to a given language specified as a context-free grammar (CFG), and draw a derivation tree when it does. It can also check whether two given CFGs specify the same language. The teacher may ban ambiguous CFGs.

Figure 4 shows the derivation tree of *ab* when the CFG is $S ::= \varepsilon \mid aSbS \mid bSaS$. This CFG generates the strings with an equal number of a's and b's.

During the autumn 2018, feedback from 28 students was obtained. The students experienced this teaching method very positively. All but one replied "weakly agree" or "strongly agree" to "it was more pleasant to study in this way than with traditional exercises" and to "I believe that I learnt more than I would have learnt with traditional exercises." The last student chose the neutral reply. These gave averages 4.4 and 4.3 in the scale from 1 to 5, satisfying $p = 0.1\%$. Altogether 11 questions had $p = 0.1\%$, two more had $p = 1\%$, two more had $p = 5\%$, and three had less statistical significance. Among the last three was "There were many too easy problems".

The problem whether two CFGs generate the same language is undecidable, that is, no algorithm can solve it completely. MathCheck performs the check by generating strings according to both CFGs in shortlex order. If it finds a string that is generated by one CFG but not the other, it reports it as a counter-example. Otherwise, it eventually gives up.

3 INCOMPLETE CHECKING

The ability of MathCheck to check the answers is not perfect. MathCheck does contain some theorem proving and symbolic computation capability, but they cannot solve all situations. It follows from (Richardson, 1968) that no computer program can perform perfectly the tasks that the arithmetic mode of MathCheck performs imperfectly. This means that a trade-off has to be made between the level of perfection and programming effort. Therefore, MathCheck checks many solutions by testing.

Checking by testing allows MathCheck to give useful feedback to the students. Replying with "this simplification step is incorrect" would be less useful to the student than what MathCheck gives, that is, a numerical counter-example and graphs of the left and right hand sides of the incorrect relation.

Two different functions built from everyday mathematical operators obtain the same value typically only on isolated points. When hundreds of points are tested, it is possible but very unlikely that they all happen to be among those where the funtions agree. This makes the testing approach rather reliable. MathCheck becomes unreliable when most of the test values are excluded for one reason or another. For instance, MathCheck fails to see that $\sqrt{x-100}$ is not the same function as $\ln(x-99)$, because they are both undefined when $x \leq 99$. In this kind of cases, MathCheck gives a warning. MathCheck can also be fooled relatively easily with $100 - x = |100 - x|$ and with the floor and ceiling operators.

After an inequality such as $f(x,y) > g(x,y)$ has passed a test value combination (x_0, y_0), MathCheck tries whether changing the value of x_0 or y_0 would make $f(x,y) - g(x,y)$ smaller. Therefore, MathCheck finds the counter-example 10000.11 to $(x - 10000.1)(x - 10000.2) \geq 0$, although it is not near 0.

The most harmful imperfection is that MathCheck is not good with cases like $\frac{x-12}{x-12} = 1$, where there are isolated points making one but not the other function undefined. This problem was discussed in Section 2.1, in the context of undefined expressions.

Because computer arithmetic is not precise, MathCheck cannot always use precise values. MathCheck uses precise rational number arithmetic when it can, and then switches to a representation consisting of a lower bound and upper bound represented as double-precision floating point numbers. MathCheck also keeps track of whether the value may be undefined. The interval as a whole must be a counter-example for MathCheck to report it as a counter-example. As a consequence, MathCheck does not deem $\sin\pi < 0$ as incorrect, but does deem $\sin\pi < -7.66 \cdot 10^{-16}$.

4 SYNTAX ISSUES

MathCheck tries to obey the established mathematical notation as strictly as possible, and much better than other computer programs. The established notation proved surprisingly ill-defined and self-contradictory.

The formula $\sin 2x = 2\sin x\cos x$ is well-known. If invisible multiplication has lower precedence than sin, then the left hand side is interpreted incorrectly as $(\sin 2)x$. With the opposite assumption, the right hand side is interpreted incorrectly as $2\sin(x\cos x)$. That is, no ordinary precedence rule yields the intended interpretation $\sin(2x) = 2(\sin x)\cos x$.

Because of this kind of confusions, some people always write parentheses around the argument of a function. To accept but not demand this convention, MathCheck obeys the complicated rule that invisible multiplication has higher precedence than function calls except if the multiplicand is a function call or $\frac{\partial}{\partial x}$, or the argument of the original function begins with (. The latter exception is because without it, the rule would make $\ln(x+1)x$ mean the same as $\ln((x+1)x)$, conflicting with the expectation of many people.

The product $|x|y|z|$ may be interpreted as both $(|x|)y(|z|)$ and $|x(|y|)z|$. MathCheck uses $(|x|)y(|z|)$. Mixed numbers such as $1\frac{2}{3}$ are inputted as `1 2/3` . When $x = 2$ we have $2\frac{1}{x} = 2 \cdot \frac{1}{2} = 1$, but literally writing 2 in the place of x in $2\frac{1}{x}$ would yield the mixed number $2\frac{1}{2} = 2.5$. So MathCheck disallows the input `2 1/x` , but allows `2 (1/x)` and prints it as $2\frac{1}{x}$.

Multiplication is sometimes written as $\cdot$ or $\times$ in mathematics. MathCheck allows $\cdot$, which is inputted as * (or as such, because MathCheck allows Unicode versions of many mathematical symbols, making copying and pasting largely possible). Its precedence is lower than those of the invisible product and function calls, and higher than that of $+$ and $-$. It can, for instance, be used to clarify $|x|y|z|$, by writing either $|x \cdot |y| \cdot z|$ or $|x| \cdot y \cdot |z|$.

We emphasize that these problems are in the established mathematical notation, and thus exist independently of MathCheck. Making MathCheck obey the established notation required hard thinking and programming effort. We are aware of no other program that is as faithful to the established notation.

5 LOGIC IN MathCheck

Most elementary-level textbooks on logic (such as (Gries and Schneider, 1993)) only use the two truth values F and T. Consider the claim $\forall x; x \neq 0 : \frac{1}{x^2} > 0$. It would be attractive to think that because of the part "$x \neq 0$", the evaluation of the claim avoids evaluating $\frac{1}{0^2} > 0$. However, by the definitions of bounded quantifiers, $\rightarrow$, and $\neq$, and by the commutativity of $\vee$, the claim is logically equivalent to $\forall x : \frac{1}{x^2} > 0 \vee x = 0$. This is undefined, because $\frac{1}{0^2} > 0$ is undefined.

Re-defining bounded quantification in the obvious way would force us to refrain from writing $\forall x : \tan x = \frac{\sin x}{\cos x}$ and write instead something like $\forall x; (\neg\exists n \in \mathbb{Z} : x = (n + \frac{1}{2})\pi) : \tan x = \frac{\sin x}{\cos x}$. This is not attractive.

This problem was solved without using U in (Spivey, 1992). Unfortunately, the solution is unnatural, because it makes many intuitively undefined claims T.

Intuitively, the negation of an undefined claim is also undefined. This can be obtained by employing U and declaring that $\neg$U yields U. MathCheck uses the 3-valued propositional logic of Kleene (Kleene, 1964; Fronhöfer, 2011). It is also used in (Jones, 1991). The 3-valued propositional logic of Łukasiewicz (Łukasiewicz, 1930; Fronhöfer, 2011) provably fails a property that is crucial for the reasoning system discussed below.

As was discussed in Section 3, MathCheck may use intervals. This implies that a comparison such as $\sqrt{\sin\pi} = 0$ may yield any non-empty combination of F, U, and T. The truth value data type of MathCheck implements such combinations. For instance, the combination FUT means that nothing is known, while FU means that the result cannot be true but it is not known whether it is false or undefined.

In the equation and modulo modes, $f(x) = 0 \Leftrightarrow x = x_1 \vee \cdots \vee x = x_n$ means that the set of those values of x where $f(x)$ *is defined and* yields 0 is $\{x_1, \ldots, x_n\}$. To make this work with such cases as $3\sqrt{x} = x + 2 \Leftrightarrow x = 1 \vee x = 4$, it was necessary to let U $\Leftrightarrow$ F, because with negative values of x, $3\sqrt{x} = x + 2$ is undefined but $x = 1 \vee x = 4$ yields F. Unlike the truth values of claims, this does not cause problems with negation, because $\neg(\varphi \Leftrightarrow \psi)$ is a syntax error. The distinction between $\leftrightarrow$ and $\Leftrightarrow$ proved again important. The symbols $\Rightarrow$ and $\Leftrightarrow$ do not yield truth values but express reasoning steps that are either valid or invalid.

Please see (Valmari and Hella, 2017) for more information on logic in MathCheck.

6 CONCLUSIONS

We discussed the MathCheck tool that gives students feedback on their solutions to mathematics and theoretical computer science problems. It supports both some traditional problem types, mainly simplification, derivatives, and equations; and some unconventional problem types related to syntax and logic. New

problem modes (such as set theory) and extensions to existing problem modes (such as solving inequalities) are in the dream list.

In some modes, MathCheck often checks the answer only incompletely, by testing with many value combinations of the variables in question. As a consequence, MathCheck may fail to find an error. Fortunately failure is unlikely as long as the user does not intentionally exploit the weaknesses of MathCheck. In return, the testing approach facilitates providing feedback on all steps of the student's solution, even in the absence of a teacher-given solution and independently of the path that the student chose towards the final answer. Furthermore, if the solution proves incorrect, the student gets a concrete counter-example.

In the equation mode, checking is slightly unreliable due to numerical imprecision. Spurious roots are detected later than one might wish. Again, these are small problems. In return, the mode can deal with very many kinds of equations, instead of being restricted to, say, quadratic equations. The array claim mode is sufficiently reliable only if the teacher takes the checking algorithm into account when designing problems. On the other hand, it offers a service that is absent from most, or perhaps all, other tools.

We also discussed some problems in the established mathematical notation. Because of them, most programs force the user to deviate from the established notation, usually by writing additional parentheses or explicit multiplication symbols. A lot of effort was made so that MathCheck would not force such deviations.

Traditional binary logic does not suffice for MathCheck. This was solved by introducing a truth value representing undefined and by using $\Rightarrow$ and $\Leftrightarrow$ as reasoning operators with significantly different properties from propositional operators.

Not many pedagogical experiments have been made with MathCheck. In those that have been made, the results have been very encouraging.

REFERENCES

Fronhöfer, B. (2011). Introduction to many-valued logics. https://web.archive.org/web/20131225052706/, http://www.wv.inf.tu-dresden.de/Teaching/SS-2011/mvl/mval.HANDOUT2.pdf; accessed 2017-11-03.

Gibbs, G. (2010). Using assessment to support student learning. Technical report, Leeds Metropolitan University.

Gibbs, G. and Simpson, C. (2004). Conditions under which assessment supports student learning. *Learning and Teaching in Higher Education*, 1:3–31.

Gries, D. and Schneider, F. B. (1993). *A Logical Approach to Discrete Math*. Texts and Monographs in Computer Science. Springer, New York.

Hakala, V. (2016). Mathcheck ja wolfram alpha opiskelun tukena. Project Report.

Jones, C. B. (1991). *Systematic software development using VDM (2. ed.)*. Prentice Hall International Series in Computer Science. Prentice Hall.

Kleene, S. C. (1964). *Introduction to Metamathematics*. Bibliotheca mathematica. North-Holland Pub. Co.

Lethbridge, T. (2000). What knowledge is important to a software professional? *IEEE Computer*, 33(5):44–50.

Łukasiewicz, J. (1930). Philosophische Bemerkungen zu mehrwertigen Systemen des Aussagenkalküls. *Comptes rendus des séances de la Societé des Sciences et des Lettres de Varsovie*, 23(Cl. III):51–77.

Richardson, D. (1968). Some undecidable problems involving elementary functions of a real variable. *J. Symbolic Logic*, 33(4):514–520.

Sangwin, C. (2015). Computer aided assessment of mathematics using stack. In Cho, S. J., editor, *Selected Regular Lectures from the 12th International Congress on Mathematical Education*, pages 695–713, Cham. Springer International Publishing.

Spivey, J. M. (1992). *Z Notation - a reference manual (2. ed.)*. Prentice Hall International Series in Computer Science. Prentice Hall.

STACK (2017). System for teaching and assessment using a computer algebra kernel. http://stack.bham.ac.uk/; accessed 2017-07-16.

Stroustrup, B. (1997). *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition.

Surakka, S. (2007). What subjects and skills are important for software developers? *Commun. ACM*, 50(1):73–78.

Valmari, A. (2003). Software mathematics as a course topic. In Kurhila, J., editor, *Kolin Kolistelut – Koli Calling 2003, Proc. Third Finnish Baltic Sea Conference on Computer Science Education, Oct. 3-5, 2003 Koli, Finland*, pages 101–109. Helsinki University Printing House.

Valmari, A. (2016). Mathcheck relation chain checker. In Karhumäki, J. and Saarela, A., editors, *Proceedings of the Finnish Mathematical Days 2016*, number 25 in TUCS Lecture Notes, pages 44–46.

Valmari, A. and Hella, L. (2017). The logics taught and used at high schools are not the same. In Karhumäki, J. and Saarela, A., editors, *Proceedings of the Fourth Russian Finnish Symposium on Discrete Mathematics*, number 26 in TUCS Lecture Notes.

Valmari, A. and Kaarakka, T. (2016). MathCheck: A tool for checking math solutions in detail. In *44th SEFI Conference, Engineering Education on Top of the World: Industry University Cooperation, 12-15 September 2016, Tampere, Finland*. European Society for Engineering Education SEFI.