Gains and Pitfalls of Quantifier Elimination as a teaching tool

By Reinhard Oldenburg

Universität Augsburg, Universitätsstr. 14, 86159 Augsburg, Germany, reinhard.oldenburg@math.uni-augsburg.de

Received: ??

Quantifier Elimination is a procedure that allows to simplify logical formulas that contain quantifiers. Many mathematical concepts are defined in terms of quantifiers and especially in calculus their use has been identified as an obstacle in the learning process. The automatic deduction provided by quantifier elimination thus allows students to exercise the formulation of concepts using quantifiers. This may be seen as conceptual modelling.

1 INTRODUCTION

Tarski has shown that formulas of first order predicate logic over certain fields can be decided algorithmically and algorithmic progress, especially the method of algebraic cylindrical decomposition. Tarski himself noted that this leads to a decision procedure for elementary geometry as well. Furthermore, it gives a systematic way to solve systems of polynomial inequalities over ℝ. Many notions from calculus that are expressed in terms of quantifiers can be formalized and decided for purely algebraic functions. This shows that the method of quantifier elimination is suited for several classes of problems that are relevant in math education at various levels. Thus the question arises, whether this method can be used as a teaching tool. One may hope that having access to quantifier elimination in a computer algebra system may give students the opportunity to explore the mentioned fields of application. Especially one may hope that this may provide a playground to exercise the formalisation step in mathematics. E.g. one may have an intuitive idea of what it means for a function to be convex on an interval but it is a crucial further step to be able to formalize this in the language of predicate calculus. We give examples of all kinds of didactically relevant applications and especially example on the formalizations of notions. Based on this example set we systematize the potential and the inherent problems of quantifier elimination as a teaching method.

2 QUANTIFIER ELIMINATION

Universal and existence quantifiers are the basic tools of predicate logic and they allow expressing many mathematical statements. In general, the truth of statements in unrestricted predicate logic is undecidable. However, Tarski's ingenious contribution is to identify a very strong subset that can indeed be decided. The domain of variables in this theory is the set of real numbers, quantifiers are first order (i.e. they concern variables for numbers not for functional symbols), propositional logic expressions (conjunction, disjunction, negation, implication), relations <, =, > among numbers and polynomials in the variables. The restriction to polynomials is not severe. In fact, the implementation in Maxima (Honda 2014) pre-processes e.g. rational equations as follows:

. Square roots and absolute values can be treated as well: $a = \sqrt{b} \Leftrightarrow a^2 = b \wedge a \geq 0, a = |b| \Leftrightarrow a^2 = b^2 \wedge a \geq 0$but this is not yet implemented in the current version of qepmax.

For this language, Tarski has shown that form all such formulas quantifiers can be eliminated and this provides algorithmic proving of statements in this language. E.g. eliminating the quantifier from $\forall x (x+1)^2 = x^2 + 2x + 1$ gives true. This is trivial, so let's look at a more interesting example: For which values of the parameter *c* does the function $f(x)=x^3-cx^2+cx+c$ have at least two distinct real roots? The statement to decide is $\exists x_1 : \exists x_2 : x_1 \neq x_2 \wedge f(x_1) = 0 \wedge f(x_2) = 0$. Eliminating quantifiers from this gives a condition on *c*, namely $c \leq -1$ or $c \geq 27/5$, that precisely describes all values for which there are at least two roots. The calculation carried out in the computer algebra system Maxima using the qepmax library is shown below in Fig. 1.

Application areas of quantifier elimination are wider than proving algebraic identities as many problems can be translated into algebraic-logical language. This includes proving theorems in elementary geometry and exact solution of (possibly constrained) optimization problems.

```
(%i146) f1(x):=x^3-c*x^2+c*x+c;
(%o146) f1(x):=x^3-c x^2+c x+c

(%i147) qe([[E,x1],[E,x2]],x1#x2 %and f1(x1)=0 %and f1(x2)=0);
(%o147) (c+1<=0) %or (5 c-27>=0)
```

Figure 1: Performing quantifier elimination in Maxima

As most notions of calculus like limit, continuity, convexity, differentiability are defined in terms of quantifiers, one can use quantifier elimination to make these notions computable at least for a restricted set of functions defined by algebraic expressions.

This paper will investigate the scope of an approach to concepts of calculus based on this method.

3 APPLICATIONS IN CALCULUS

The need to work with quantifiers has been identified as one of the key obstacles in learning calculus (e.g. Roh 2009 and Durand-Guerrier et al. 2012). Working with a computer algebra system that supports quantifier elimination allows students to formalize notions and then test if the formal expression they gave really gives the behaviour they intended. The quantifier elimination methods thus gives them quick response that is guaranteed to be formally correct.

We view this whole process as a special case of modelling (Niss et al. 2007). Students may have developed e.g. from looking at examples an intuitive notion, a concept image (Tall&Vinner 1981). Now, this informal notion is formalized to become a sound logical formula. Finally, this is translated into the syntax of the computer algebra system and can then be applied. Interpretations of results may make it necessary to refine the definition so that it meets the concept image that should be modelled – or it may lead to the insight that the informal concept image is not well formed and should be altered. The following diagram (Fig. 2) shows a modelling circle (Niss et al. 2007) applied to the concept of global minimum. In this example the informal concept image might first have the property the minimum is unique which is not encoded in the formal definition as the example shows. This might either lead to modify the informal or the formal definition or it might lead to introduce two notions.

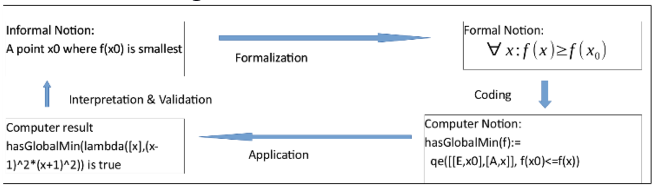

Figure 2: Modelling circle for modelling concepts within a formal system

```
(%i2) p1(x):=(x-5)^2+x+1; p2(x):=x^3-x;
(%o2) p1(x):=(x-5)^2+x+1
(%o3) p2(x):=x^3-x

(%i4) r1(x):=x/(x-3); r2(x):=1/((x-2)*(x-1)); r3(x):=-1/(1+x^2);
(%o4) r1(x):=x/(x-3)
(%o5) r2(x):=1/((x-2)(x-1))
(%o6) r3(x):=-1/(1+x^2)
```

Figure 3. Some test functions

```
(%i7) hasGlobalMin(f):= qe([[E,x0],[A,x]], f(x0)<=f(x)) ;
(%o7) hasGlobalMin(f):=qe([[E,x0],[A,x]],f(x0)<=f(x))

(%i8) [hasGlobalMin(p1),hasGlobalMin(p2)];
(%o8) [true,false]

(%i9) [hasGlobalMin(r1),hasGlobalMin(r2),hasGlobalMin(r3)];
(%o9) [false,false,true]

(%i10) hasGlobalMin(lambda([x],(x-1)^2*(x+1)^2));
(%o10) true

(%i11) hasUniqueGlobalMin(f):= qe([[E,x0],[A,x]], x#x0 %implies f(x0)<f(x));
(%o11) hasUniqueGlobalMin(f):=qe([[E,x0],[A,x]],x#x0 %implies f(x0)<f(x))

(%i12) hasUniqueGlobalMin(lambda([x],(x-1)^2*(x+1)^2));
(%o12) false
```

Figure 4: Global minima (# in Maxima means 'not equal'. Anonymous functions are defined in Maxima using the lambda function constructor)

Local minima are an even richer field for exploration. The absolute value function that is needed to describe distances is not available in the current implementation of qepcad so that one needs to express $|x| < y$as $x^2 < y^2 \wedge y \geq 0$and similar for other uses (probably this may require to introduce new variables).

```
(%i17) hasLocalMin(f):= qe([[E,x0],[E,delta],[A,x]],
          delta>0 %and (((x-x0)^2<delta^2) %implies (f(x0)<=f(x))  )) ;
(%o17) hasLocalMin(f):=qe([[E,x0],[E,δ],[A,x]],δ>0 %and ((x-x0)^2<δ^2 %implies f(x0)<=

(%i18) hasLocalMin(lambda([x], x^2*(x-1)  ));
(%o18) true

(%i19) hasLocalMin(lambda([x], x^3  ));
(%o19) false

(%i20) hasLocalMin(lambda([x], -x^2  ));
(%o20) false
```

Figure 5: Formalizing local minima.

Up to now, all variables in the formulae where bound by quantifiers. Thus eliminating them yielded either false or true as result. Variables that are not bound by quantifiers are in a sense much more interesting, as they allow the method to calculate conditions on them to make the statement true as the next example shows:

```
(%i21) hasLocalMinIn(f,x0):= qe([[E,delta],[A,x]],
          delta>0 %and (((x-x0)^2<delta^2) %implies (f(x0)<=f(x))  )) ;
(%o21) hasLocalMinIn(f,x0):=qe([[E,δ],[A,x]],δ>0 %and ((x-x0)^2<δ^2 %implies f(x0)<=f(x)))

(%i22) hasLocalMinIn(lambda([x], x^2*(x-1)  ),X);
(%o22) 3X-2=0
```

Figure 6: Leaving variables free often produces interesting results!

Next, we consider the notion of continuity. The techniques to translate the standard formalization are similar to the examples above so we hope that the screen shots speak for themselves.

```
(%i23) continous(f,x0):= qe([[A,eps],[E,delta],[A,x]],
          (delta>0) %and ((eps>0) %implies   (( ((x-x0)^2<delta^2) ) %implies
                                  ((f(x)-f(x0))^2<eps^2) )));
(%o23) continous(f,x0):=qe([[A,eps],[E,δ],[A,x]],δ>0 %and (eps>0 %implies ((x-x0)^2<δ^2 %implies (f(x)-f

(%i24) p1(x):=(x-5)^2+x+1$ p2(x):=x^3-x$ r1(x):=x/(x-3)$
       r2(x):=1/((x-2)*(x-1))$ r3(x):=-1/(1+x^2)$

(%i29) continous(p1,2);
(%o29) true

(%i30) continous(r1,2);
(%o30) true

(%i31) continous(r1,a);
(%o31) a-3#0
```

Figure 7: Translating absolute value conditions to squares is a bit clumsy but works as expected. Especially, the method can detect where rational functions are not continuous.

Besides rational function there are no examples where the question of continuity is interesting in the range of functions that can be applied. However, using a simple trick one can encode piecewise defined functions by logical propositions by means of the equivalence:

$$f(x) = \begin{Bmatrix} f_1(x) if x < x_0 \\ f_2(x) if x \geq x_0 \end{Bmatrix} \wedge y = f(x) \Leftrightarrow$$

$$(x < x_0 \Rightarrow y = f_1(x)) \wedge (x \geq x_0 \Rightarrow y = f_2(x))$$

This requires rewriting functions in a relational way. By convention, we use the name for functions represented in that way, the square function e.g. is encoded as. With this bag of tricks piecewise defined functions can be handled.

```
(%i44) makePWfunglg(f1,x0,f2):=  apply(lambda,[[x,y],  ((x<x0) %implies (y=f1(x))) %and ((x>=x0) %implies (y=f2(x)))]);
(%o44) makePWfunglg(f1,x0,f2):=apply(lambda,[[x,y],(x<x0 %implies y=f1(x)) %and (x>=x0 %implies y=f2(x))])

(%i45) makePWfunglg(lambda([x],x),0,lambda([x],x^2))(5,y);
(%o45) (5>=0 %implies y=25) %and (5<0 %implies y=5)

(%i46) continousFG(fg,x0):= qe([[E,y0],[A,eps],[E,delta],[A,x],[E,y]],
          (delta>0) %and ((eps>0) %implies   (( ((x-x0)^2<delta^2) ) %implies
                         ( fg(x,y)  %and fg(x0,y0) %and  (y-y0)^2<eps^2) )));
(%o46) continousFG(fg,x0):=
qe([[E,y0],[A,eps],[E,δ],[A,x],[E,y]],δ>0 %and (eps>0 %implies ((x-x0)^2<δ^2 %implies fg(x,y) %and fg(x0,y0) %and (y-y0)^2<eps^2)))

(%i47) continousFG(makePWfunglg(lambda([x],x),0,lambda([x],x^2+x)),0);
(%o47) true

(%i48) continousFG(makePWfunglg(lambda([x],x+1),0,lambda([x],x^2+x)),0);
(%o48) false
```

Figure 8: Piecewise defined functions and the test for continuity

The same considerations apply for testing differentiability. However, it turns out that proving the differentiability of piecewise defined functions would take too much memory to be handled by Maxima on standard computers and maybe out of the reach as the complexity of the quantifier elimination algorithm can be very time consuming (e.g. double exponential in the number of variables). However, the implementation of quantifier elimination in Mathematica can handle this example.

```
(%i52) DiffGlg(fg,x0,m):=qe([[E,y0],[A,eps],[E,delta],[A,x],[E,y]],
         (eps>0 %and x#x0) %implies (delta>0 %and (((x-x0)^2<delta^2) %implies
           ( fg(x,y) %and fg(x0,y0) %and   (m-(y-y0)/(x-x0))^2<eps^2)            )) );
(%o52) DiffGlg(fg,x0,m):=qe([[E,y0],[A,eps],[E,δ],[A,x],[E,y]],eps>0 %and x#x0 %implies
((x-x0)^2<δ^2 %implies fg(x,y) %and fg(x0,y0) %and (m-(y-y0)/(x-x0))^2<eps^2))

(%i53) DiffGlg(lambda([x,y],y=x^2),2,d);
(%o53) d-4=0

(%i54) DiffGlg(makePWfunglg(lambda([x],x),0,lambda([x],x^2)),2,d);
bash: Zeile 1:  9130 Abgebrochen          (Speicherabzug geschrieben) $qe/bin/qepcad +N20
(%o54) FAIL
```

Figure 9: Maxima proving that the derivative of $f(x) = x^2$at 2 is 4. It fails however, to establish differentiability of a piecewise defined function.

```
In[1]:= implies[a_, b_] := Not[a] || b

In[2]:= diff[fg_, x0_, d_] :=
         Exists[y0,
          fg[x0, y0] &&
           ForAll[eps,
            implies[eps > 0,
             Exists[delta,
              delta > 0 &&
               ForAll[x,
                Exists[y, implies[x ≠ x0, fg[x, y] && implies[Abs[x - x0] < delta,
                    Abs[(y - y0) / (x - x0) - d] < eps]]]]]]]]

In[3]:= quadrat[x_, y_] := y == x^2

In[4]:= diff[quadrat, 3, d]

Out[4]= ∃y0 (y0 == 9 && ∀eps (eps ≤ 0 ||
            ∃delta (delta > 0 && ∀x ∃y (x == 3 || (y == x² && (Abs[-3 + x] ≥ delta || Abs

In[5]:= Resolve[diff[quadrat, 3, d], Reals]

Out[5]= d == 6

In[6]:= makePW[f1_, x0_, f2_] := Function[{x, y}, implies[x < x0, y == f1[x]] && imp

In[7]:= Resolve[diff[makePW[Function[{x}, x], 0, Function[{x}, x^2 + x]], 0, d],

Out[7]= d == 1

In[8]:= Resolve[diff[makePW[Function[{x}, x], 0, Function[{x}, x^2 + x]], 1, d],

Out[8]= d == 3

In[9]:= Resolve[diff[makePW[Function[{x}, 2 * x], 0, Function[{x}, x^2 + x]], 0,

Out[9]= False
```

Figure 10: Mathematica can determine the derivative (see cell In[8]) for which Maxima (Fig. 9) failed

Next, we consider two notions that are linked to calculus without involving limits: monotone and convex functions.

```
(%i32) monotoneDe(f,a,b):= qe([[A,x1],[A,x2]],(a<x1 %and x1<x2 %and x2<b) %implies f(x2)<f(x1));
(%o32) monotoneDe(f,a,b):=qe([[A,x1],[A,x2]],a<x1 %and x1<x2 %and x2<b %implies f(x2)<f(x1))

(%i33) monotoneIn(f,a,b):= qe([[A,x1],[A,x2]],(a<x1 %and x1<x2 %and x2<b) %implies f(x2)>f(x1));
(%o33) monotoneIn(f,a,b):=qe([[A,x1],[A,x2]],a<x1 %and x1<x2 %and x2<b %implies f(x2)>f(x1))

(%i34) monotoneIn(lambda([x],x*(x-1)*(x-2)),a,b);
(%o34) (a-1>0) %and (3a^2-6a+2>=0) %or (b-a<=0) %or ((b-1<0) %and (3b^2-6b+2>=0))
```

Figure 11: Functions that are strictly monotone in an interval. Obvious variations are non-strict versions.

```
(%i36) convexity(f,a,b):= qe([[A,x1],[A,x2],[A,x3]],
          a<b %and ((a<x1 %and x1<x2 %and x2<x3 %and x3<b) %implies f(x2)<f(x
(%o36) convexity(f,a,b):=qe([[A,x1],[A,x2],[A,x3]],a<b %and (a<x1 %and x1<x2 %and

(%i37) convexity(lambda([x],x^2),1,5);
(%o37) true

(%i38) convexity(lambda([x],x*(x-1)*(x-2)),1,a);
(%o38) a-1>0
```

Figure 12: Convexity detection without using second derivative

A question that pushes the complexity of logical combinations to the limit of the system (and maybe the user) is to express that a point is an inflection point of a function in the sense that the function is convex on one side and concave on the other.

```
(%i56) inflectionpoint(f,x3,a,b):= qe([[A,x1],[A,x2],[A,x4],[A,x5]],
        a<x3 %and x3<b %and (a<x1 %and x1<x2 %and x2<x3 %and x2<x3 %and x3<x4 %and x4<x5 %and x5<b)
         %implies ((f(x2)<f(x1)+(x2-x1)*(f(x3)-f(x1))/(x3-x1)) %and (f(x4)>f(x3)+(x4-x3)*(f(x5)-f(x3))/(x5-x3))
        %or (f(x2)>f(x1)+(x2-x1)*(f(x3)-f(x1))/(x3-x1)) %and (f(x4)<f(x3)+(x4-x3)*(f(x5)-f(x3))/(x5-x3)))
        );
(%o56) inflectionpoint(f,x3,a,b):=qe([[A,x1],[A,x2],[A,x4],[A,x5]],a<x3 %and x3<b %and
(a<x1 %and x1<x2 %and x2<x3 %and x2<x3 %and x3<x4 %and x4<x5 %and x5<b) %implies (f(x2)<f(x1)+(x2-x1)(f(x3)-f(x1))/(x3-x1)
%and f(x4)>f(x3)+(x4-x3)(f(x5)-f(x3))/(x5-x3) %or f(x2)>f(x1)+(x2-x1)(f(x3)-f(x1))/(x3-x1) %and f(x4)<f(x3)+(x4-x3)(f(x5)-f(x3))/(x5-x3)))

(%i57) inflectionpoint(lambda([x],x*(x-1)*(x-2)),x,-5,5);
(%o57) (x-5>=0) %or (x-1=0) %or (x+5<=0)
```

Figure 13: Checking if is an inflection point of$f(x) = x(x-1)(x-2)$ in the interval [-5,5]. Note that if is not from this interval no statement is made, so it is not false

4 THE PITFALLS

The last section has shown some glimpses of the power of the method of quantifier elimination. The most important is that it is a correct method so that students get results they can trust it (unless the computing time or memory usage becomes to high to be acceptable). This is a crucial point. In Oldenburg&Weygandt (2015) and we showed how much wrong or incomplete answers from computing limits within a computer algebra system can irritate students. However, this correctness comes at a price: The set of test functions is restricted to the algebraic class described above, so that all transcendental functions like exponentials, logarithms and trigonometric functions as well as special functions that are of interests e.g. for physicists (Airy, Bessel, ...) are out of the reach of the method. Piecewise defined functions have to be handled in way that is not very user friendly (although this drawback might be overcome by smoother interfaces to the core method). In its direct incarnation of the method not even rational functions can be handled which leads the students with a set of functions that includes only continuous and differentiable ones. The Maxima implementation is thus an important step from a didactical perspective.

Another drawback is the vast computing power to carry out the methods – and that is increases very fast with the complexity of the problem. One may say that from the perspective of a trained mathematician the method is only up to toy problems.

5 BALANCING GAINS AND PITFALLS

Often the claim is made that the most advanced mathematics and most sophisticated implementation is not needed for education as students deal with elementary concepts. Quantifier elimination is certainly an exception to this. Only now the implementations get so fast and reliable that the method can provide a safe and reliable 'playground' for working with quantifiers. We expect the method to become in more widespread use in the next years as it gains momentum from several directions including exact optimization and automated theorem proving in geometry. This will hopefully increase the use in education as well and will provide empirical evidence to give a judgement if he gains outweigh the pitfalls.

REFERENCES

Artigue, M., Oldenburg, R. (2014): How to get rid of quantifiers. Klein Vignette. In electronic 'press'

Durand-Guerrier, V. Et al.: An insight on university mathematics teaching practices about proofs involving multiple quantifiers. ICME (2012)

Niss, M., Blum, W. & Galbraith, P. (2007). Introduction. In W. Blum, P. Galbraith, P., H.-W.Henn & M. Niss (Eds.) Modelling and applications in mathematics education – The 14th ICMI study (pp. 4-32). New-York, USA: Springer.

Oldenburg, R., Weygandt, B. (2015): Einsatzmöglichkeiten und Grenzen von Computeralgebrasystemen zur Förderung des Begriffsverständnisses. Proceedings of khdm 2013, in press.

Qepmax (2014): https://github.com/YasuhaikHonda/qepmax: Maxima-qepcad-interface by Y. Honda with contributions by R. Oldenburg

W. Stein et al (2013) : SAGE. www.sagemath.org

A. Tarski (1948): A decision method for elementary algebra and geometry, Rand Corporation Publication.

F. Winkler (1996): Algorithms for Polynomials. Springer.

Roh, K. H.: An empirical study of students' understanding of a logical structure in the definition of limit via the ε-strip activity. Educ Stud Math (2010) 73:263–279

Vinner, S. & Tall, D. (1981). Concept image and concept definition in mathematics with particular reference to limits and continuity. Educational Studies in Mathematics 12 (1981) 151-169.

St. Wolfram : Mathematica – A system for doing mathematics by Computer.