# Interactive Learning and Mathematical Calculus*

Arjeh M. Cohen, Hans Cuypers, Dorina Jibetean, and Mark Spanbroek

Technische Universiteit Eindhoven,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

**Abstract.** A variety of problems in mathematical calculus can be solved by recursively applying a finite number of rules. Often, a generic solving strategy can be extracted and an interactive exercise system that emulates a tutor can be implemented.

In this paper we show how software developed by us can be used to realize this interactivity. In particular, an implementation of a generic exercise for computing the derivative of elementary functions is presented.

## 1 Introduction

This paper deals with mathematical interactive learning. In most implementations of multi-step exercises for interactive learning the exercises are completely authored for each instance of the problem. Here we discuss an implementation of *generic* exercises for specific mathematical problems. This means that the exercise system provided will depend on the type of problem and not on the particular instance of the problem. In our example, the generic problem is *computing derivatives of elementary functions* while a particular instance of the problem is *computing the derivative of* $x \mapsto x^2 + 3$. (From here on, we shall omit the binding operation '$x \mapsto$' when referring to a function in $x$.)

For the design of automated exercises we use mathematical knowledge coming from two sources. Firstly, many problems in mathematical calculus can be solved by applying recursively a finite number of rules. The rules used for solving mathematical problems in a particular domain will be called *domain rules*. In our example, these will be the differentiation rules of composed functions (sum, product, quotient, chain rule) and the formulas for derivation of elementary non-composed functions (for instance, the derivative of sin equals cos).

Secondly, we need information on the particular instance of the problem, like the type of mathematical object we deal with and its definition. This information can be immediately deduced from its OpenMath [10] representation and it is typically sufficient to infer the domain rule(s) to be used for solving the problem in this particular instance. In fact, what we need is the map assigning to a mathematical (OpenMath) object the domain rule which needs to be applied. We will call this map a *domain reasoner*.

* Work carried out within the LeActiveMath project.

, pp. 330–345, 2005.

In an interactive exercise, the student is required to solve a problem. Our strategy is to decompose the original problem into simpler sub-problems, so as to obtain a multi-step exercise. An interactive exercise can be seen as a collection of problems together with the order in which they are executed. An exercise always has a first step, corresponding to the original question. According to the student's answer and a predefined strategy, the next step is selected. In this way, the student is guided in solving the initial question. The correctness of the student's answer is evaluated by the use of a computer algebra system (CAS) connected to the exercise. There are quite a few interactive exercise systems which check automatically the correctness of the student's answer (such as [11], [7], [5]) but they are not as concerned with tutoring. These exercises consist of a single evaluation step. Some systems (like [7]) also provide automatically generated hints.

The novelty of our approach is that we produce generic exercises for certain mathematical problems by using the semantic information encoded in the OpenMath expression. Our approach can be applied to any mathematical problem for which a complete set of domain rules and a map from the OpenMath expression to the set of domain rules can be defined. We believe that this is possible for many problems in mathematical calculus.

The structure of the paper is as follows. In section 2 we present our general approach to interactive exercises, exemplified for the computation of derivatives. It shows how we use the domain rules and the OpenMath tree structure of the instance to generate the interactive exercise. Sections 3 and 4 contain details on the design of the interactive exercises, respectively on the set-up of the system running the exercise and some of the tools developed for this type of interactive exercise. In Section 5 we show our implementation for computing the derivative of an arbitrary (differentiable) univariate function. Section 6 discusses the applicability of the method to other problems in mathematical calculus. Section 7 presents ways to enhance the interactive exercise into detecting misconceptions of the student and other errors. Conclusions are presented in Section 8.

## 2 Interactive Exercises

Let us analyze the problem of computing the inverse of a given matrix using Cramer's rule. Here, a possible sub-problem is computing the determinant of the matrix. This helps for example to determine whether the inverse exists or not. Another sub-problem involves computing the elements of the inverse using Cramer's rule, as the fraction of two determinants. This exercise will consist of several steps, namely the original question, the problem of computing the determinant of a matrix, the problem of computing an element of the inverse (repeated $n^2$ times, where $n$ is the size of the matrix) and, at last, an acknowledgment of the result.

There are however more complicated examples of calculus problems in which, although there is a finite number of procedures that can be applied, the order in which they are executed is not the same for any instance of the problem. In case few alternatives are possible, an interactive exercise should allow the students to choose the course of action they want to pursue.

The choice of the rules to apply depends in general on the particular instance of the problem. For example, in differentiating $\sin(x^2 + 1)$ one applies the chain rule and the elementary differentiation rules for sin and polynomials, while differentiating $x+\log(x)$ requires applying the sum rule and the elementary differentiation rules for the logarithm and the identity.

Still, a generic exercise can be implemented in such cases. Using the OpenMath tree structure of the expression of the function, the system can recognize the rule to be applied. Here we concentrate on derivatives, although in Section 6 we discuss the applicability of the method to other fields in mathematical calculus.

Derivatives are the *case study* considered by the European project LeActiveMath [2] for interactive, user-adapted e-learning. Much of the work reported in the present paper is carried out within the context of this project.

### 2.1 Derivatives

The domain rules in the case of derivatives are easy to derive. They are the differentiation rules of composed functions and the formulas for differentiation of elementary non-composed functions.

- sum: $(f + g)' = f' + g'$.
- product: $(fg)' = f'g + g'f$.
- quotient: $(f/g)' = (f'g - g'f)/g^2$.
- chain rule: $(f \circ g)' = (f' \circ g)\, g'$.
- $c' = 0$, where $c$ is a constant.
- $id' = 1$, where $id$ denotes the identity function.
- elementary non-composed functions: $\sin' = \cos$, $\exp' = \exp$, etc.

The OpenMath expression of a function contains all necessary information for computing the derivative of the respective function according to the above rules. One can determine whether the elementary function is composed or not, by the fact that a composed function has at least two OMAs in its OpenMath expression. In case the function is composed one can easily decide which one of the rules for composed functions needs to be applied. Let us consider for example the OpenMath expression of $\sin(x^2 + 1)$.

```
<OMOBJ>
  <OMA><OMS cd='transc1' name='sin'/>
    <OMA><OMS cd='arith1' name='plus'/>
      <OMA><OMS cd='arith1' name='power'/>
        <OMV name='x'/> <OMI>2</OMI>
      </OMA>
      <OMI>1</OMI>
    </OMA>
  </OMA>
</OMOBJ>
```

By looking at the root operator (the first OMS) it is clear that we either need to apply the chain rule if the function is composed or, otherwise, a differentiation rule for elementary functions. For $x+\log(x)$ the root of the OpenMath expression is the 'plus' symbol which suggests applying the sum rule. In this way we define a map matching the root of the OpenMath tree to the particular domain rule.

In Section 5, we discuss this generic exercise in greater detail.

## 3 The Design of an Interactive Exercise

This section explains the structure of an interactive exercise. Each exercise corresponds to a particular area in mathematical calculus. Specifically, we will have one interactive exercise for computing derivatives, one for computing limits, etc. Such an exercise is applicable to any instance of the problem (under some general assumptions) and it is in this sense generic.

We decompose a given problem into several sub-problems. Then we construct a directed graph having a source (start node) and a sink (end node). Each node of the graph corresponds to a sub-problem. Each arc shows a possible succession from one sub-problem to another one. The graph is constructed in such a way that each path from the source to the sink is a possible solution of the original problem. Therefore, we call the graph a solution graph and a path from the source to the sink a solution-path.

Each sub-problem has a text field, containing for example a question, and a user-input field, in which the student types the answer. The only exception is the sink which contains only the text field, to acknowledge the result. According to the input of the user or to the state in which the system finds itself, an arc is chosen and implicitly the next step/sub-problem to be solved.

An abstract solution graph is depicted in Figure 1. Each node represents a subproblem. The source, denoted by 1 contains the original question. In case condition $c1$ is satisfied, the student is directed to a a different question 2. In case another condition is satisfied, $c1'$, the student is redirected for example to 1. In case neither $c1$ nor $c1'$ are satisfied, the student is directed to the sink, represented by $n$.

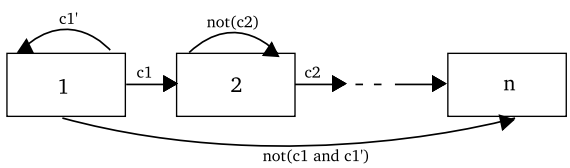


**Fig. 1** An abstract solution graph

The arcs leaving a node must represent exclusive conditions and must cover all possibilities. The conditions can be imposed for example on the student input, leading for example to a specific sub-problem if the answer to a question is correct and to another one if the answer is incorrect. The conditions can be imposed

on other parameters as well, like a maximum allowed number of trials, previous performance of the student, etc.

The solution graph is characteristic for a mathematical problem. It is designed to take into consideration the domain reasoner of the problem and aspects of a teaching methodology. There is no unique design for the solution graph of a specific mathematical problem.

## 4 Implementation of the Interactive Exercises

This section explains the implementation of the interactive exercises. More details on the general set-up and tools necessary to run the interactive exercises can be found in [6].

The exercises are implemented as a web-application. They are written in an XML based language that offers mark-up support for interactivity enriched with OpenMath for handling mathematics.

### 4.1 XML Representation of the Solution Graph

XML trees are ideal for capturing the structure of the solution graph of an exercise. Our examples use the following principal XML tags: **step**, **message**, **userinput** and **choice**. Each **step** corresponds to one sub-problem in the solution graph. The step/sub-problem has an attribute **id** by which it can be called. Each step has the following components: **message**, **userinput** and **choice**. The only exception is the last step, the sink, which has only **message**. The message contains the text or mathematical question posed to the student. That is, for example in Figure 2, *Please input a function in $x$ whose derivative you would like to compute*. In Figure 2, the user input is a text-box having the label **Function**. The tag **choice** implements the *arcs* of the solution graph, and redirects to the next step. The choice and redirect option are further explained in the following section. The choice is hidden from the user.

### 4.2 Variables and Flow Control

The interactivity is catered by a Java based server (e.g. JSP [4]) which also provides a basic programming-like language for setting and retrieving variables, performing tests, flow control, etc. The interactivity consists of walking through the solution graph and the exchange of information with the user. The path one takes is determined by various parameters which can be determined by queries but also by previously set variables.

### 4.3 Mathematical Queries

Mathematics is represented according to the OpenMath standard ([10]). OpenMath encoded objects can be displayed by browsers (through conversion to MathML) and interpreted by computer algebra systems via OpenMath phrasebooks. Moreover, we use here the OpenMath expression of a mathematical object for interactivity in order to determine the domain rule to be applied (see Sections 2.1 and 6).

Within the frame provided by the Java based server we construct custom tags. Examples are provided below.

**Query to a CAS.** It is necessary for the automatic evaluation of the student's input. A standard for mathematical queries was defined by MONET (see [8]). See also the OpenMath webpage, Software and Tools, for some implementations of query services ([10]).

**Query to a taglib.** The following queries to the OpenMath tree structure of a mathematical expression turn out to be very useful in our interactive examples: extraction of a node, extraction of the root (operator) of a node, navigation in the tree (e.g., move to parent/next sibling/first child), rewriting the tree (e.g., mathematically $x + x^2$ equals $x(x + 1)$, but the corresponding OpenMath-expressions are different), keeping track of the current node (a query of the type *What is the parent of* $x$ *in* $(1 + x)^2/x$ is ambiguous). The use of these operations will be made clear in Section 5.

The list of queries above is not exclusive and may need to be extended in case an interactive exercise for a different mathematical problem is considered.

## 5 The Example for Computing Derivatives

We present here an implementation of a generic exercise for computing the derivative of a function, see [3]. As a back-engine for checking the student's answer we use Mathematica. Expressions like $(x - 1)/(x^2 - 1)$ and $1/(x + 1)$ are therefore considered equal.

Students are allowed to choose a (univariate) function. Then they are asked to compute the derivative. In case they give a wrong answer they are guided through decomposing the function into simpler ones and then apply differentiation rules. In this implementation the next step is determined in interaction with the student. The student receives hints automatically, as shown in Step 3.

We present below snapshots taken while running the exercise. The exercise consists of a finite set of interactions, one asking the student to compute the derivative of a function, one asking the student to decompose the function, etc. This 'modularization' of the exercise allows us to reuse parts of the exercise. Note that each module is largely humanly authored but that parts of it are automatically updated by means of variables as described in Section 4. Note that the path taken for solving the exercise is decided in interaction with the student.

We describe below a possible scenario.

Step 1. The student introduces a function on which they want to practice. Alternatively, the function can be drawn from a database or randomly drawn from a particular class of functions such as polynomials, trigonometric, transcendental or any composition of functions belonging to these classes.

Step 2. The student is asked to submit the elementary function that is the derivative of the function.

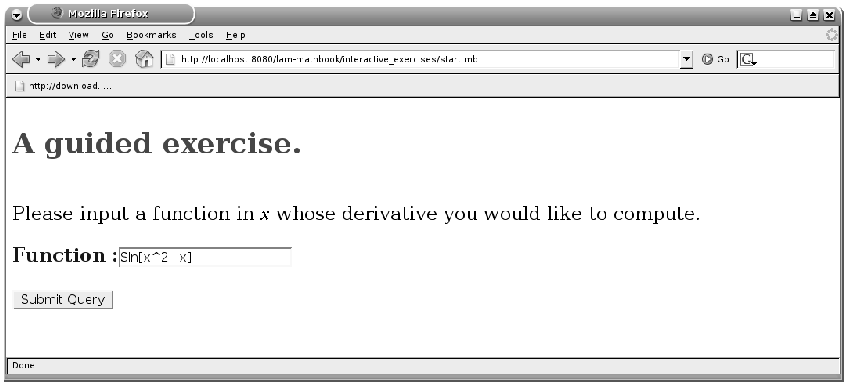


**Fig. 2**

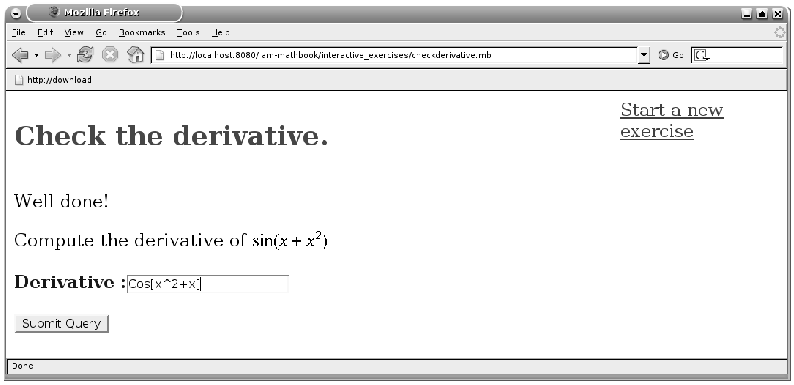


**Fig. 3**

Step 3. The student's answer is checked by Mathematica. If the student's answer is not correct, they are asked to choose a differentiation rule they want to apply. The button `Hint`, when pressed, displays the text: 'Use the chain rule'. Here the word *chain* is determined automatically by the system by analyzing the OpenMath expression of the function whose derivative we are computing at this step. The system finds the rule to be applied by using the special tools described in Section 4, namely the extraction of the root operator.

Step 4. After having typed the rule, the student is redirected accordingly.

If the rule chosen by the student matches the rule corresponding to the main operator, the system can also suggest the two functions $g$ and $h$. This is not implemented in the current version.

Step 5. The student introduces the two functions and the CAS verifies that by composing the two, one obtains indeed the original function. If that is correct, the student is directed to the first function and asked for its derivative. The expression of the first child is obtained using the tools of Section 4, namely the

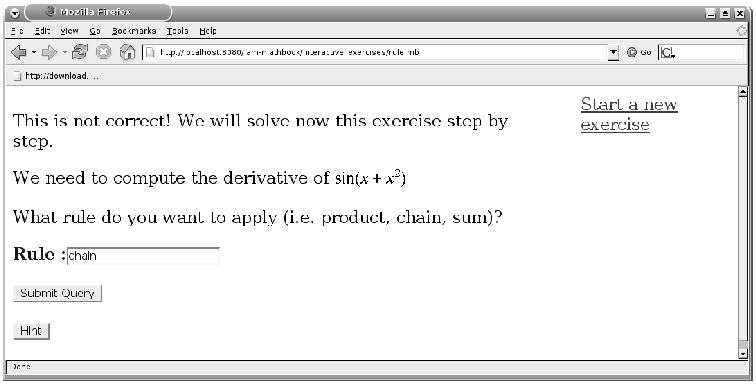


**Fig. 4**

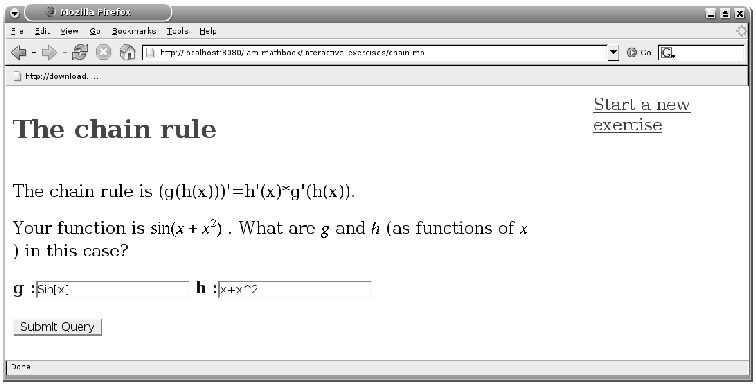


**Fig. 5**

navigation in the tree and the extraction of a node. Note that we reuse here the interaction from Step 2, for a different function.

Step 6. Since the answer at Step 5 is correct, we compute the derivative of the next sibling (reusing interaction from Steps 2 and 5). The next sibling is also found using the tools of Section 4.

Step 7. Since the second answer is correct and there are no more siblings we are redirected to compute the derivative of the parent (interaction of Steps 2, 5 and 6). If the answer were not correct at this point the student would have been asked to decompose further $x^2 + x$. (Students are allowed to decompose it both as a sum of $x$ and $x^2$ and as a product of $x$ and $x + 1$).

Note that the hint, when pressed, displays the sum rule for derivation.

Step 8. The answer is correct and this is acknowledged in the last page.

The complete solution graph of this problem is illustrated in Figure 10. In the graph, the subproblem 1 corresponds to Figure 2. From subproblem 1 we are always redirected to subproblem 2 which corresponds to Figure 3. Here de-


**Fig. 6**

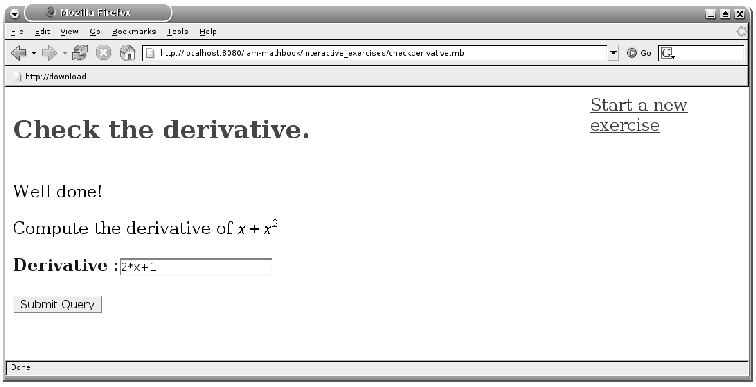


**Fig. 7**

pending on the student's answer and on previous knowledge we choose the next step. Let us explain the arcs. $C_{11}$ is taken if the answer is incorrect and the derivative for the children are known. $C_{12}$ is taken when the answer is correct and there is a next sibling, as it happens for example at Step 6. In this case, the current function is also updated. $C_{13}$ is taken if the answer is incorrect and the derivatives of the children have not yet been computed. $C_{14}$ is taken in case the answer is correct and there is no parent of the current node as it happens in Step 8. $C_{15}$ is taken if the answer is correct, the derivatives of all children have been computed and the current node has a parent. This situation occurs at Step 7.

The subproblem of 3 corresponds to Figure 4. The subproblem of 4 corresponds to Figure 5, while subproblem 5 corresponds to a different rule. The arcs $C_{21}$, $C_{22}$ correspond to the choice the student is making. Arc $C_{31}$ corresponds to an incorrect decomposition of the function into simpler ones, while the arc $C_{32}$ corresponds to a correct decomposition. Note that if the choice is correct, the

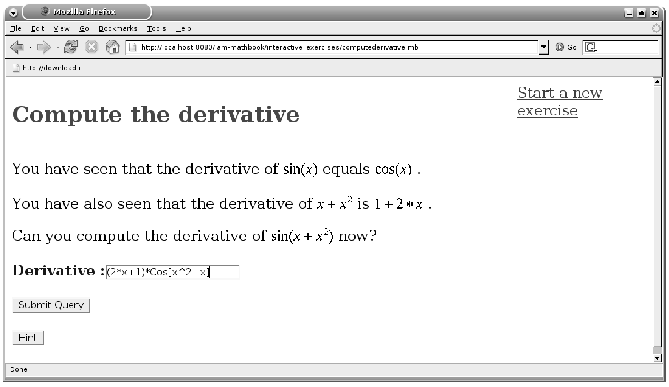


**Fig. 8**

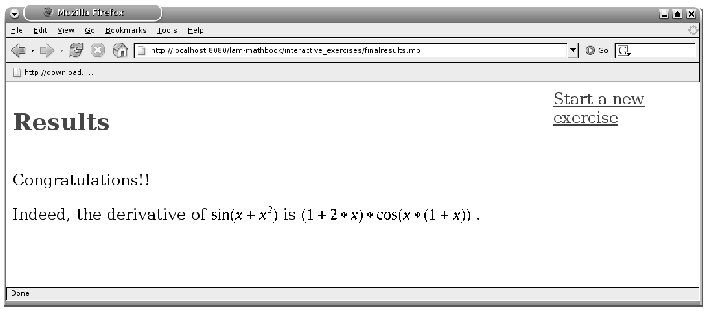


**Fig. 9**

function we work with (the current node) is updated and takes the value of the first child as it happens at Step 5. The sub-problem 7 corresponds to Figure 8. Arc $C_{51}$ corresponds to a correct answer. In this case the function we work with (current node) is updated to the parent. Arc $C_{52}$ corresponds to an incorrect answer.

## 6 Other Applications

In this section we discuss the domain reasoners for the problem of computing limits, respectively the problem of computing indefinite integrals of elementary functions. Recall that the domain reasoner is the map from the mathematical knowledge on the instance of the problem (typically available in its OpenMath expression) to the set of domain rules.

### 6.1 Limits

The limit of a univariate elementary function when its argument goes to a specified value is in general obtained by taking the limits of its parts. In other words,

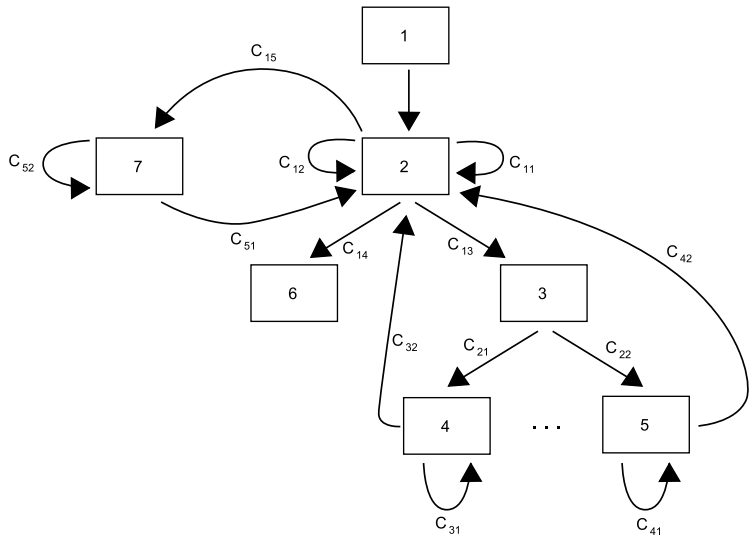


**Fig. 10.** solution graph for the derivative problem

taking the limit commutes with the composition of continuous functions. There are a few exceptions: in case the result of applying the above mentioned rule is one of the undecided cases $0^0, 1^\infty, 0^\infty, \infty^0, 0/0, 0 \times \infty, \infty - \infty$, rewriting rules are applied.

*Example 1.* Consider

$$\lim_{x \to 1} \left(\frac{1}{x}\right)^{\frac{1}{1-x}}$$

which results in the undecided situation $1^\infty$. To solve the problem we reduce it to the undetermined case $0/0$ by applying the rewriting rule

$$\left(\frac{1}{x}\right)^{\frac{1}{1-x}} = \exp \log \left(\frac{1}{x}\right)^{\frac{1}{1-x}} = \exp \frac{-\log x}{1-x},$$

and then apply l'Hôpital rule; we conclude that the original will equal

$$\exp^{\lim_{x \to 1} \frac{1}{x}} = e.$$

L'Hôpital rule says that in the undecided cases where $\lim f(x)/g(x)$ is either $0/0$ or $\pm\infty/\pm\infty$ and $\lim f'(x)/g'(x)$ exists, we have $\lim f(x)/g(x) = \lim f'(x)/g'(x)$. L'Hôpital rule can be applied repeatedly and is equivalent to the method based on Taylor series expansions. Other undecided cases can be reduced to the $0/0$, $\infty/\pm\infty$ cases using rewriting rules. For example, the 'power' cases $0^0, 0^{\pm\infty}, 1^{\pm\infty}$ are rewritten using the equivalence $f(x) = \exp(\log(f(x)))$ while for the 'difference' cases $\infty - \infty$ we can use $f(x) = \log(\exp(f(x)))$. However, when the expression of $f(x)$ contains radicals, a different rewriting rule may need to be applied.

As shown by the example, $\lim_{x\to\infty} x/\sqrt{x^2+1}$, l'Hôpital's rule occasionally fails to yield useful results. In certain cases, the limit of a 'part' of the function does not exist, although the limit of the whole function exists.

*Example 2.* Consider

$$\lim_{x\to\infty} \frac{\sin(x)}{x}.$$

In such a case the *sandwich* rule is applied:

$$-\frac{1}{x} \leq \frac{\sin(x)}{x} \leq \frac{1}{x} \quad \text{implies} \quad \lim_{x\to\infty} \frac{\sin(x)}{x} = 0.$$

The domain rules for computing limits are the rewriting rules, l'Hôpital/Taylor series rule and the sandwich rule.

- l'Hôpital rule: if $\lim_{x\to a} f(x)$ and $\lim_{x\to a} g(x)$ are both 0 or are both $\pm\infty$ and $\lim f'(x)/g'(x)$ exists, then $\lim_{x\to a} f(x)/g(x) = \lim_{x\to a} f'(x)/g'(x)$.
- sandwich rule: if $g(x) \leq f(x) \leq h(x)$, for all $x$ in $(a-\epsilon, a+\epsilon)$ ( $x$ in $(M,\infty)$, $x$ in $(-\infty,-M)$, when $a=\infty$, respectively $a=-\infty$ and $M>0$ large) and $\lim_{x\to a} g(x) = \lim_{x\to a} h(x) = L$, then $\lim_{x\to a} f(x) = L$.
- rewriting rules: $f(x) = \exp(\log(f(x)))$, $f(x) = \log(\exp(f(x)))$, rewriting rules for radicals, etc.

Computing the limit of a composed function is based on computing the limits of its sub-functions. Again, the OpenMath expression of a function is very useful for identifying the sub functions. Let us consider the OpenMath expression of $\lim_{x\to 1}(\frac{1}{x})^{\frac{1}{1-x}}$ (Example 1).

```
<OMOBJ>
  <OMA><OMS cd="limit1" name="limit"/>
    <OMI> 1 </OMI>
    <OMS cd="limit1" name="both_sides"/>
    <OMBIND><OMS cd="fns1" name="lambda"/>
      <OMBVAR><OMV name="x"/></OMBVAR>
        <OMA><OMS name='power' cd='arith1'/>
          <OMA><OMS name='divide' cd='arith1'/>
            <OMI>1</OMI><OMV name='x'/>
          </OMA>
          <OMA><OMS name='divide' cd='arith1'/>
            <OMI>1</OMI>
            <OMA><OMS name='minus' cd='arith1'/>
              <OMI>1</OMI><OMV name='x'/>
            </OMA>
         </OMA>
       </OMA>
    </OMBIND>
  </OMA>
</OMOBJ>
```

As in the case of differentiation, we can start by asking the student to compute the limit of the function. In case of a wrong answer we ask the student to decompose the function. The hint is given according to the OpenMath expression of the function. By doing this recursively, at the end of this procedure we have either computed the limit or identified the undecided case.

According to the undecided case we find, a rewriting rule (from the domain rule) is applied in order to bring the function to a case in which l'Hôpital can be applied as described in Section 6.1.

As remarked in Example 2, it is possible that evaluating a sub-function at some point we obtain a whole 'interval'. In this case, the rule (of the domain rule) to be applied is the sandwich rule.

In general the mathematical knowledge contained in the OpenMath expression is sufficient for defining the domain reasoner. An exception is the computation of limits in which knowledge on the range of the function is necessary for applying the sandwich rule.

### 6.2 Indefinite Integrals

Computing indefinite integrals (also called primitives) of elementary functions is in general much harder than computing derivatives or even limits. However a lot of progress has been made in the area of symbolic integration (see e.g. [1]) and many computer algebra systems have an integrated module for symbolic integration. Powerful algorithms have been developed which compute the indefinite integral of an elementary function in case it exists or prove that there is no elementary primitive.

Nevertheless our problem is in some sense simpler. We do not want to compute the primitive of a given function or to establish whether the primitive exists. To make sure that a primitive of $f$ exists, we will randomly generate an elementary function $F$ and compute its derivative $f$. Knowing $F$, we can guide the student through finding $F$ as shown below.

*Example 3.* Consider the function $f : \mathbf{R} \to \mathbf{R}$ given by $f(x) = 1/(1 + 2x + 2x^2)$ and a primitive $F(x) = \arctan(x/(x + 1))$ of it. From the OpenMath expression of $F(x)$ we see that $F$ equals $\arctan(y)$ for some function $y$ of $x$. Knowing that $(\arctan(y))' = y'/(y^2 + 1)$, a helpful suggestion to the student is:

> Can you write the function $f$ as $y'/(y^2+1)$ for a suitably chosen function $y$ of $x$?

Note that the answer is not unique. Besides $F$, other primitives of $f$ are

$$F_1(x) = -\arctan\left(\frac{x+1}{x}\right) \text{ obtained by rewriting } f(x) \text{ as } -\frac{(1/x)'}{1+(1/x)^2}$$

and

$$F_2(x) = -\arctan(1 + 2x) \text{ obtained by rewriting } f(x) \text{ as } \frac{(2x+1)'}{1+(2x+1)^2}.$$

All answers are correct and it can be checked that $F$, $F_1$ and $F_2$ all differ from each other by a constant.

There are several differences between our problem, which is teaching rules for computing integrals and the problem of computing integrals using state-of-the-art algorithms for symbolic integration described for example in [1]. We know the answer and want to deduce the steps that were taken in order to solve the problem. In some cases, as shown in Example 3, this is trivial. However other rules, like the integration by parts, are hard to recognize. Also, the algorithms described in [1] can be very different from the methods taught in schools, like the sum, chain, partial fractions, integration by parts rules and the elementary rules for integration.

The first implementations of algorithms for computing indefinite integrals are closer to the methods taught in school. To make our job easier, we can simply take over the domain reasoner of such a computer algebra system, for example the one described in [9]. There, the domain reasoner is based on 'pattern recognition' of a sub-function like $\sin, \exp$, etc., which is trivial from the OpenMath expression of the given function and uses heuristics for determining the rule to be applied. A multi-step interactive exercise for symbolic integration will have to implement the domain reasoner of a computer algebra system, for example of [9].

The alternative for us would be to write a domain reasoner ourselves, using the knowledge we have on the primitive $F$ and on the function $f$, as suggested in Example 3. However we do not pursue this idea here.

## 7 Enhanced Interactive Exercises

The work presented in the previous sections can be easily extended in a few directions.

### 7.1 Detecting Errors and Misconceptions

A common error the student can make is forgetting brackets. If the number of opened brackets does not equal the number of closed brackets, the error can be detected at the moment of parsing the answer to OpenMath. Otherwise, forgotten brackets can still be detected in simple expressions, by extensive trials, although that would involve a number of computations which increases exponentially with the size of the expression.

Another interesting way of using the computer power for educational purposes, namely for detecting the misconceptions of the student, is by introducing the so-called *buggy rules*. Experienced teachers know what are the most frequent errors that a student makes when solving a particular type of problem. For example, a common mistake (buggy rule) when computing derivatives is $(x^n)' = x^{n-1}$. This knowledge can be formalized and the system can compare the student's answer against a list of buggy rules, in case the answer is not correct. In this way, the computer algebra system can make a guess as to what the student did wrong.

### 7.2 Simplifications

The student may often give a correct result which is however not simplified, as in $(x-1)/(x^2-1)$. Most computer algebra systems have implemented simplification procedures to bring an expression to a normal form. At the moment we only distinguish between two cases, namely between a correct answer and a wrong answer. For pedagogical purposes, a third case would be interesting, namely 'correct but not fully simplified'. For this we would have to compare the student's answer with the normal form(s) available in the computer algebra system. However, this is a tricky issue since a normal form is not uniquely defined (e.g., both forms $x(x+1)$ and $x^2+x$ are acceptable). Also different computer algebra systems will have different implementations of simplification procedures, leading to different normal forms. Nevertheless, computer algebra systems often have a complexity measure of an expression which may be used to detect very elaborate answers.

### 7.3 Solution Generators

In this paper we have proposed implementing the domain reasoner for each particular problem in order to identify a possible next step. An alternative would be to use already existing tools, such as [12], which generate a detailed solution for a particular instance of the problem. For example [12] can display the solution of practically any instance of a variety of calculus problems among which we find computing derivatives and computing limits. This is achieved by using the domain reasoner implemented in Mathematica. The fact that [12] is an extension of Mathematica is in our view a disadvantage since this may restrict its use. At the moment the only interactivity [12] provides is in choosing the problem to be solved, hence it cannot be directly applied for interactive exercises. However, it is an innovative use of computer algebra systems and it may turn out to be very useful for interactive exercises.

## 8 Conclusion

The paper describes a method for implementing multi-step interactive exercises for certain problems in mathematical calculus. This is possible by exploiting the mathematical knowledge available in the OpenMath expression of the mathematical objects we deal with, by deriving the domain rules corresponding to the problem and constructing a domain reasoner for solving the problem.

As a first approximation, in the case of differentiation, the necessary properties of the mathematical object are deducible from its OpenMath expression. However, we have noticed already an exception in Example 2 where extra knowledge (about bounds of the function) is necessary in order to be able to compute certain limits.

Our set-up for the particular case of computing derivatives for functions is a first attempt, made possible within the LeActiveMath project [2].

## References


1. M. Bronstein, Symbolic Integration I, Transcendental Functions, Springer, 1997.
2. LeActiveMath, http://www.leactivemath.org.
3. LeActiveMath exercises, http://www.riaca.win.tue.nl:8080/leam-exercises.
4. The J2EE Trademarked 1.4 Tutorial, http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html.
5. MapleTA, http://www.maplesoft.com/products/mapleta/index.aspx.
6. A.M. Cohen, H. Cuypers, E. Reinaldo Barreiro, MathDox: Mathematical Documents on the Web, Contribution to OMDoc book, http://www.win.tue.nl/~hansc/mathdox3.pdf
7. Metr*ic*, Imperial College London, http://metric.ma.imperial.ac.uk/new/html/index.html.
8. Monet (Mathematics on net), http://monet.nag.co.uk/cocoon/monet/index.html.
9. Joel Moses, Symbolic integration: the stormy decade, http://portal.acm.org/citation.cfm?id=362651
10. OpenMath, http://www.openmath.org/cocoon/openmath/index.html.
11. Stack (System for Teaching and Assessment using a Computer algebra Kernel), http://eee595.bham.ac.uk/~stack/index.html.
12. webSolutions Detailed and Dynamic Mathematical Solutions, http://www.webmath.ch/.