

# Supporting Tutorial Feedback to Student Help Requests and Errors in Symbolic Differentiation\*

Claus Zinn<sup>1</sup>

DFKI — German Research Centre for Artificial Intelligence  
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany,  
Email: [zinn@dfki.de](mailto:zinn@dfki.de), Phone: +49 681 3025064

**Abstract.** The provision of intelligent, user-adaptive, and effective feedback requires human tutors to exploit their expert knowledge about the domain of instruction, and to diagnose students' actions through a potentially huge space of possible solutions and misconceptions. Designers and developers of intelligent tutoring systems strive to simulate good human tutors, and to replicate their reasoning and diagnosis capabilities as well as their pedagogical expertise. This is a huge undertaking because it requires an adequate acquisition, formalisation, and operationalisation of material that supports reasoning, diagnosis, and natural interaction with the learner. In this paper, we describe SLOPERT, a glass-box reasoner and diagnoser for symbolic differentiation. Its expert task model, which is enriched with buggy rules, has been informed by an analysis of human-human tutorial dialogues. SLOPERT can provide natural step-by-step solutions for any given problem as well as diagnosis support for typical student errors. SLOPERT's capabilities thus support the generation of natural problem-solving hints and scaffolding help.

## 1 Introduction

The European project “Language-enhanced, user adaptive, interactive e-learning for Mathematics (LeActiveMath)” aims at developing an innovative third generation e-learning system for high-school and university-level self-learners. One of the most prominent features of LeActiveMath is its exercise subsystem that supports students solving problems in the domain of calculus. The LeActiveMath project explores two approaches to implement this subsystem; a more traditional approach where professional math tutors *pre-author* interactive exercises; and a cutting-edge research approach that aims at *dynamically generating* exercises. The traditional approach is quite labour-intensive because exercise authors have to write-down a large number of possible dialogues for a given task. They have to anticipate students' correct and incorrect answers, and provide suitable dialogue continuations for them. The cutting-edge research approach does not rely on explicit representations of dialogue. It aims at providing an *exercise player* that can generate future interactions dynamically by exploiting two main knowledge

---

\* Funded by the European Commission's 6th Framework Programme: *IST-507826*.

sources: a reasoning and diagnosis engine for the interpretation of student input, and a response system, powered with conversational expertise and tutorial feedback strategies, for the generation of system response.

To better understand effective one-to-one human tutoring in the domain of calculus, in particular, symbolic differentiation, we collected and analysed a corpus of one-to-one dialogues between professional math teachers and students. Our data analysis aims at identifying our tutors' conversational, domain, and pedagogical expertise. In this paper, we focus on modelling observed tutors' problem solving and diagnosis expertise. We describe how we encoded this expertise in SLOPERT to provide glass-box reasoning and diagnosis services for a dialogue manager to generate human-like feedback to student input.

## 2 Background

Our research focuses on the operationalisation of human tutoring in the domain of symbolic differentiation: what types of input (and errors) do students produce, and what kind of scaffolding and feedback do tutors generate to help learning?

Sinus Rule	$\frac{d}{dx} \sin(x) = \cos(x)$
Logarithmic Rule	$\frac{d}{dx} \log(x) = \frac{1}{x}$
Power Rule	$\frac{d}{dx} [x^n] = n \cdot x^{n-1}$
Constant Multiple Rule	$\frac{d}{dx} [c \cdot f(x)] = c \cdot \frac{d}{dx} [f(x)]$
Sum Rule	$\frac{d}{dx} [f(x) \pm g(x)] = \frac{d}{dx} [f(x)] \pm \frac{d}{dx} [g(x)]$
Chain Rule For Power Functions	$\frac{d}{dx} ([f(x)]^n) = n \cdot [f(x)]^{n-1} \cdot \frac{d}{dx} [f(x)]$
General Chain Rule	$\frac{d}{dx} [f(g(x))] = \frac{d}{dg} [f(g(x))] \cdot \frac{d}{dx} [g(x)]$

**Fig. 1.** Differentiation rules;  $c$  is any real number,  $f(x)$  and  $g(x)$  are any functions.

We collected a corpus of one-to-one tutoring in this domain.<sup>1</sup> Students were given teaching material on the rules of symbolic differentiation (see Fig. 1). They were made familiar with the overall subject in their normal course of studies. During a computer-mediated session a tutor gave differentiation problems to students and then provided feedback on their problem solving steps.

Figure 2 displays three consecutive fragments of one tutor-student interaction. In the first fragment, we find the student (**S**) solving the task given by the tutor (**T**) in **T-1** without tutorial intervention. In **S-2a**, the student first rewrites the original problem statement into a form that allows him to apply the chain rule for power functions in **S-2b**. In **S2c**, the student then simplifies the resulting term by identifying the common factor 2 in the term  $(6x^5 + 2)$ , extracting it, and multiplying it with  $-\frac{5}{2}$  all in one step. Then, in **S-2d**, the student asks the tutor for feedback, which the tutor provides in **T-3a**.

The tutoring dialogue becomes more interesting with a problem that involves the  $\log$  function, introduced by the tutor in **T-3b**. The student is unsure how

<sup>1</sup> Collected by Kaska Porayska-Pomsta, Helen Pain and Manolis Mavrikis as part of the LeActiveMath project, using tools developed by Porayska-Pomsta and Mavrikis.

- T-1** Try the following one:  $\frac{5}{\sqrt{(x^6+2x)}}$
- S-2a:**  $5(x^6 + 2x)^{-\frac{1}{2}}$
- S-2b**  $= -\frac{5}{2}(x^6 + 2x)^{-\frac{3}{2}}(6x^5 + 2)$
- S-2c**  $= -5(x^6 + 2x)^{-\frac{3}{2}}(3x^5 + 1)$
- S-2d** i think thats right ut im not too sure
- T-3a** That's very good. You really have got to grips with the chain rule for algebraic expressions.
- 
- T-3b** Let's move on to other functions. Try to differentiate  $\log(x^2 + 6x - 1)$ .
- S-4** im not sure what you get when you differentiate log. It is  $e^x$ ?
- T-5** No. I'll tell you:  $\frac{d}{dx}\log(x) = \frac{1}{x}$
- S-6** im still not sure how to do this one
- T-7** First you need to identify the functions in the composition (f(g(x))). By the way, you really need to remember what the derivative of log is...
- S-8** i still dont know what to do
- T-9a** Think about the example that you read in the beginning.
- T-9b** Try to identify  $z$  again and then  $y$  as a function of  $z$ .
- S-10**  $z = x^2 + 6x - 1$  im not sure about what y is
- T-11** That's good so far. Now think where  $z$  appears in the expression
- S-12**  $y = \log z$
- T-13** Yes. That's right. Now can you put it all together?
- S-14**  $\frac{1}{x^2+6x-1}(2x + 6)$
- T-15a** Yes. That's it. We could write that as  $\frac{2(x+3)}{x^2+6x-1}$
- 
- T-15b** Now let's try one with trig functions. Try  $\frac{1}{\sin^3 x}$  Remember that the derivative of sin is cos
- S-16**  $(\sin^3 x)^{-1} - (\sin^3 x)(3\cos^2 x)$
- T-17** Think of  $\sin^3(x)$  as  $(\sin(x))^3$
- S-18**  $3(\sin(x))^2(\cos x)$
- T-19** That's much better. Now can you solve the original problem (which is a little different)?
- S-20**  $(\sin(x))^{-3} = -3(\sin(x))^{-2}(\cos x)$
- T-21** Almost. Remember that the derivative of  $x^{-n}$  is  $-nx^{-n-1}$ .
- T-22** I think you know the answer:  $-\frac{3\cos(x)}{\sin^4 x}$
- S-23** yes that is what i was thinking [...]

**Fig. 2.** Consecutive human-human tutorial dialogue fragments (not spell-corrected).

to build a derivative for terms that contain logarithmic functions, and makes an incorrect guess in **S-4**. In **T-5**, the tutor acknowledges the incorrect answer with the provision of negative feedback, and the corrected answer. However, the student is still stuck, and in **T-7** a hint is given that presents the problem in a more abstract form. Apparently, the student does not realise that the general chain rule applies and utters **S-8**. After reminding the student in **T-9a** of pre-read material, the tutor elaborates **T-7** by sketching two sub-tasks in **T-9b**. The student gets the first subtask solved in **S-10**, and tutorial help in **T-11** provides more scaffolding to get the student solve the second subtask. In the remaining sub-dialogue, the student then combines the results of the sub-tasks to obtain a solution for the overall task in **S-14**, which the tutor simplifies in **T-15a**. In the third fragment, the tutor presents a term with a trigonometric function in **T-15b**. The student, however, rewrites the problem statement into a form that is correct, but not adequate for subsequent problem solving. In **S-16**, the student also applies the general rule for power functions to the inappropriately rewritten term (with errors). In **T-17**, the tutor, recognising the student's confusion, hints toward the correct interpretation of the original term by providing the proper reading for its denominator  $\sin^3 x$ . In **S-18**, the student adopts the term  $\sin(x)^3$  as new problem and solves it. In **T-19a**, the tutor then asks the student to solve

the original term. The student’s answer in **S-20** contains a common error in applying the power rule for negative exponents. In **T-21**, the tutor then restates the power rule in terms of  $x^{-n}$ , and then, corrects the student’s answer in **T-22**. Instead of giving instructions to the student, our tutor(s) used hints to get the student to construct the knowledge and solve the problem. Whenever the student is stuck, the tutor hinted at the next possible step towards solving a given problem. In **T-7**, for instance, the tutor hinted at the form of the problem instead of providing the proper term reading; and in **T-9b** and **T11**, the tutor hinted at the substitution variables  $z$  and  $y$  and their relation, instead of just giving away this substitution step. Remedial advice depends on adequate analysis of student error. The student’s inappropriate rewriting in **S-16**, *i.e.*, is recognised and remediated with the proper reading of the problematic sub-term. The student’s error in **S-20** is addressed by pointing the student to a version of the power rule that is specialised to polynomials with negative exponents.

### 3 Formalising Expert Reasoning

The mathematical notation of differentiation rules (c.f. Fig. 1) can be easily translated into the syntax of a symbolic programming language like PROLOG. A possible encoding of the Sinus Rule is `derive(Var, sin(Var), cos(Var))`.

The predicate `derive` has three parameters: the term to differentiate (second argument) with respect to a variable (first argument), and the result (third argument). To compute the derivative of  $\sin(x)$  with respect to  $x$ , we write

`?-derive(x, sin(x), Result).` (note the query prompt “?-”)

PROLOG answers this *query* with “Result = cos(x)”; this query *unifies* with the aforementioned PROLOG rule by instantiating the variable `Var` with `x`.

Now consider the encodings of the general power rule and the sum rule:

```
derive(Var, X^N, (N*(X^N1))*XP) :- N1 is N - 1, derive(Var, X, XP).
derive(Var, A+B, A1+B1)          :- derive(Var, A, A1), derive(Var, B, B1).
```

The first rule, or *clause*, has two parts, or *subgoals*. First, `N1` is assigned the value of subtracting 1 from `N`; and second, a *recursive* call computes the derivative `XP` of some term `X` wrt. `Var`. Instantiations that result from the evaluation the subgoals are transported to the third argument of `derive`, which contains the result of the computation, namely,  $(N*(X^{N1}))*XP$ . The sum rule has two parts, too. Here, all computation happens in the recursive calls.

*Example.* The query `?- derive(x, 5*(x^6+2*x)^(-0.5), Answer)` *succeeds* and results into the following instantiation of the variable `Answer`:

```
Answer = 5*(-0.5*(x^6-2*x)^(-1.5)*(6*x^5+2*1)).
```

This answer is unsatisfying for two reasons. First it is not “tidied-up”; the student’s answer in **S-2a** and **S-2b** has a simpler and more readable expression; and second, PROLOG only returns the result of the computation, not any intermediate steps. It thus behaves similar to a black-box computer algebra system whereas a glass-box approach would be more useful for tutoring.

*Extended Representation.* There is a simple but elegant way to extend our representation to add to the result of the computation its computation process:

```

derive(X,sin(X),cos(X), [sinRule(sin(X) = cos(X))]).
derive(X,X^N,(N*(X^N1)), [powerRule(deriv(X^N)=(N*(X^N1)))] :-
    freeof(X, N), N1 is N - 1.
derive(X,A+B, A1+B1, [sumRule(deriv(A+B)=deriv(A)+deriv(B)),Rules]) :-
    derive(X, A, A1, R1), derive(X, B, B1, R2), append(R1, R2, Rules).

```

In this representation, `derive` has four arguments, where the fourth is a description of the rule (or the relevant parts thereof). The effects of this change of representation can be seen by executing the following PROLOG query:

```
?-derive(x, 5*(x^6+2*x)^(-0.5), Answer, Explain).
```

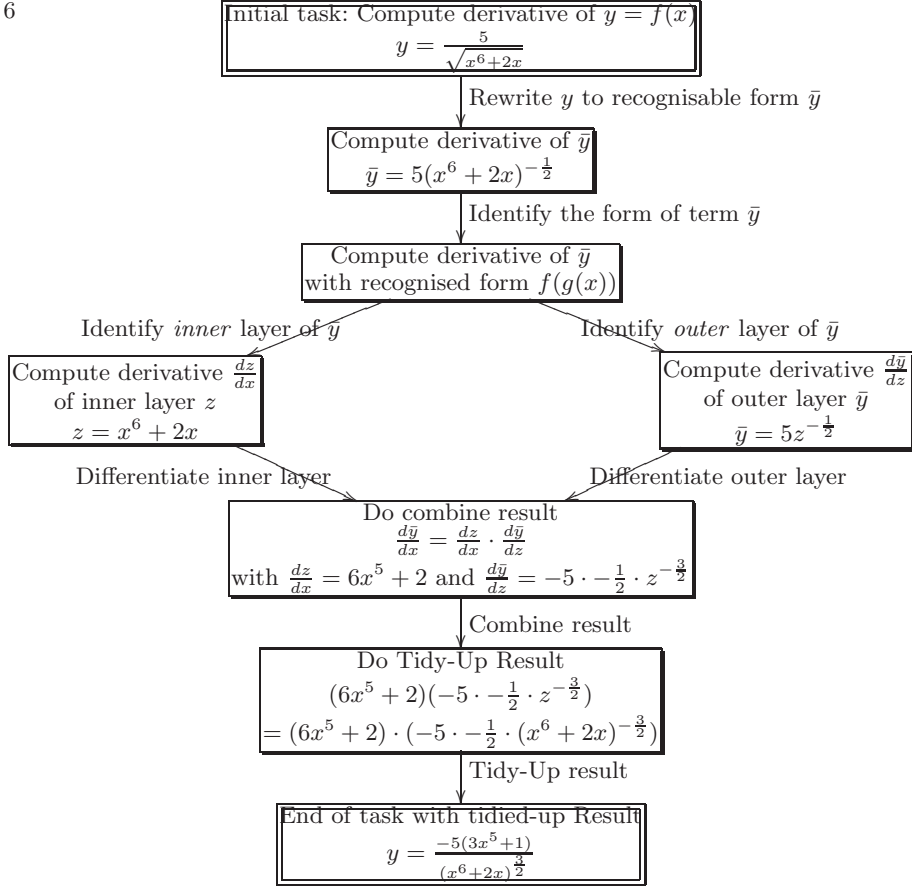
While `Answer` has still the same value, the new parameter `Explain` now contains the *solution graph* that shows each of the rules that were used to compute `Answer`:

$$\begin{array}{c}
 \frac{d}{dx} 5(x^6 + 2x)^{-\frac{1}{2}} \\
 \text{const. mult. rule} \downarrow \\
 5 * \frac{d}{dx} (x^6 + 2x)^{-\frac{1}{2}} \\
 \text{general power rule} \downarrow \\
 -\frac{1}{2} (x^6 + 2x)^{-\frac{3}{2}} * \frac{d}{dx} (x^6 + 2x) \\
 \text{sum rule} \downarrow \\
 \frac{d}{dx} x^6 + \frac{d}{dx} 2x \\
 \text{power rule} \downarrow \qquad \qquad \qquad \downarrow \text{const. mult. rule} \\
 6x^5 \qquad \qquad \qquad 2 * \frac{d}{dx} x \\
 \qquad \qquad \qquad \qquad \qquad \qquad \downarrow \text{linear rule} \\
 \qquad \qquad \qquad \qquad \qquad \qquad 1
 \end{array}$$

The richer representation enables the problem solver to communicate (parts of) its problem solving strategies. It supports, for example, the generation of feedback given in **T-5**, **T-15b**, and **T21**, where the tutor cites derivation rules. However, our representation is still not sufficient to simulate our tutor’s feedback **T-7**, **T-9b**, or **T-11**. This is because our PROLOG-based rule representation is still not explicit enough to inform the generation of such feedback. The tutor’s help “identify the form of the statement”, for instance, is implicit in PROLOG’s unification mechanism. We now extend our rule representation to give a more detailed and explicit account of the tasks involved in computing derivatives.

*Extended Representation II.* Fig. 3 describes our task model for building derivatives. Its design has been informed by an analysis of our human-human tutorial dialogues. The graph consists of framed nodes (describing tasks or goals) and labelled arcs (describing actions). We have complemented the generic task model with a specific problem instance, namely, “compute the derivative of  $y = \frac{5}{\sqrt{x^6+2x}}$ ”, which is solved within the root’s child nodes.

We have extended our representation to encode the task model for the general chain rule: the body of the `derive` clause now contains a more detailed



**Fig. 3.** Task model for building derivatives (with term rewriting).

implementation (substitutions, decompositions into separate problems for inner and outer layer, combination of results *etc.*), and the last argument of the clause head now contains the relevant explanation for it (where unification and notation is made explicit). As a result, PROLOG now returns the derivative of an input term and a solution graph that contains *all* expert problem solving steps necessary to support tutoring. Our enriched representation can now be exploited, for instance, to mimic the interesting scaffolding feedback of our human tutor in the second dialogue fragment. The respective solution graph now contains the nodes that identify the form of the statement in terms of inner and outer layer, *i.e.*, `identify_form(log(x2 + 6x - 1), [inner_layer(x2 + 6x - 1), outer_layer(log)])` that specify the necessary substitutions (*i.e.*, `z = x2 * 6x - 1`, `y = log(z)`), and that compute the sub-derivatives (*e.g.*, `log_rule(deriv(z, log(z))=z-1)`). In **T-5**, our tutor cites the general rule for differentiating *log* functions, thus abstracting from the specific rule application in the solution graph. In **T-7**, the tutor then exploits the solution graph's first action. A tutorial dialogue manager could use this node to generate hints of increasing specificity. Its generation component could exploit and verbalise this information varying lexicon, grammar, and dialogue move type, *e.g.*, "First, you have to identify the term." (declarative),

”Identify the form of the term!” (imperative), or ”How about identifying the form of the term?” (rhetorical). Also, the specificity of the hint could be varied, *e.g.*, ”Identify the functions in the compositions  $f(g(x))$ ”, ”Identify the inner and outer layers of the given term”, ”We first need to identify the form of the term. The inner layer is  $x^2 + 6x - 1$ . What’s the outer layer?”.

The generation of a more specific hint is useful when prior scaffolding fails. We observed also cases where tutors jumped to the next node of a solution graph, as for instance, our tutor in **T-9b**. There are also cases where our tutor interleaves positive feedback with scaffolding help. To support a machine tutor with the generation of such advanced feedback (*e.g.*, in **T-11** and **T-13**), further annotation of the solution graph is needed. Whenever a student answer is diagnosed as correct, the respective node is marked as `visited(diag)`; whenever a hint needs to be generated, the first non-visited node (encountered through graph traversal) is selected, exploited, and then marked as `visited(hint)`.

## 4 Formalising Buggy Reasoning

Our student in Fig. 2 makes many correct moves, but creates a few errors as well. We observe, for instance, the application of an incorrect basic rule for building the derivative of  $\log$  in **S-4**; a mis-interpretation of the form of the problem statement in **S-16**; and the application of an erroneous power rule in **S-20**. These errors are quite common among our subject population. Other typical misconceptions from our corpus are summarised in Tab. 1. In our corpus, we

Missing inner layer (chain rule)	$\frac{d}{dx}(x^3 - 3x)^5 \not\rightarrow 5(x^3 - 3x)^4$
Wrong exponent (chain/power rule)	$\frac{d}{dx}(5x^3 - 6)^{-3} \not\rightarrow -3(5x^3 - 6)^{-2}(15x^2)$
Missing exponent (chain/power rule)	$\frac{d}{dx}(x^3 - 3x)^5 \not\rightarrow 5(x^3 - 3x)(3x^2 - 3)$
Incorrect basic rule (sinus rule)	$\frac{d}{dx}\sin(x) \not\rightarrow -\cos(x)$
Missing bracketing (op. precedence)	$\frac{d}{dx}(x^3 - 3x)^5 \not\rightarrow 5(x^3 - 3x)^4 3x^2 - 3$
Erroneous transformation (rewriting)	$\frac{1}{(5x^3 - 6)^3} \neq (5x^3 - 6)^{-\frac{1}{3}}$

**Table 1.** Typical student errors in symbolic differentiation.

observed tutors to give corrective or scaffolding help based on their diagnoses. To simulate such behaviour, we complement expert reasoning with *buggy* reasoning.

Fig. 4 depicts our enriched representation. To distinguish expert from buggy

```

derive(expert,X,X^N,N*(X^N1),[powRule(deriv(X^N)=N*(X^N1))]) :-
    freeof(X, N), N1 is N - 1.
derive(buggy,X,X^N,N*(X^N1), [powRule(deriv(X^N)=N*(X^N1)),wrongPow(N1)]) :-
    N < 0, freeof(X, N), N1 is N + 1.
derive(buggy,X,X^N,N*X, [powRule(deriv(X^N)=N*X), missingExpTerm(N-1)]) :-
    freeof(X, N).
derive(buggy,X,X^N,X^N1,[powRule(deriv(X^N)=X^N1),missingFactor(N)]) :-
    freeof(X, N), N1 is N - 1.

```

**Fig. 4.** Buggy rules for symbolic differentiation in PROLOG.

rules, we add a new first argument to `derive`. Also, a buggy rule may have additional conditions, and always has a description of its “bugginess”. The first buggy rule is only applicable for negative exponents. The fact that students add 1 instead of subtracting it is encoded in the rule’s fifth argument `wrongPow(N1)`.

We can exploit our enriched rule representation to diagnose student error. When we ask PROLOG to construct a (potentially buggy) solution graph with the provision of both input and output term, we target PROLOG’s search:

```
?- derive(RuleType, x, x^(-4), (-4*(x^(-5))), Explain)
=> RuleType = expert
   Explain = [powerRule(deriv(x^(-4))=(-4*(x^(-5))))]
?- derive(RuleType, x, x^(-4), (-4*(x^(-3))), Explain)
=> RuleType = buggy
   Explain = [powerRule(deriv(x^(-4))=(-4*(x^(-3))), wrongPow(-3))]
```

The symbolic differentiation of  $x^{-4}$  only requires the use of the power rule. When the student answers  $-4 \cdot x^{-5}$ , only the expert rule for power functions *fires*, and PROLOG returns its description. When the student answers  $-4 \cdot x^{-3}$ , the buggy rule that covers wrong powers for negative exponents is applicable, and its argument `Explain` is instantiated appropriately. A machine tutor could then use this information to produce remedial feedback of various specificity, for instance, “There is something wrong.”, or “Check the power!”, or **T-21**.

## 5 Evaluation

We encoded all the expert rules of differentiation, including the detailed task model presented in Fig. 3, and many buggy rules, including those of Tab. 1. The resulting PROLOG program, called SLOPERT, can automatically compute a wide range of differentiation problems. It can generate an expert graph that provides natural step-by-step solutions. It can use buggy rules to diagnose student answers, with potentially multiple errors. We have also implemented a graph traversal that can trace through the space of possible solutions. Annotation has been added to mark nodes of the graph as visited, either because they were used for the generation of hints or for the diagnosis of student answers.

Our tutors generated hints at any stage during the student’s problem solving. Hints varied from vague (“there is a common factor”) to specific (“The expression  $5(x^3 - 3x)^4(3x^2 - 3)$  has a common factor of 3”). Sometimes, more specific hints followed vague hints, and sometimes, the first hint given was very specific. SLOPERT supports the generation of such hints. It can exploit its expert model to generate solution graphs that contains all necessary intermediate steps. How much of the nodes’ information should be given to the student is a different matter, and should be answered by the pedagogical module of the ITS, which may take into account other information than problem state and error diagnosis.

Our students seem to be quite familiar with the differentiation rules. In most cases, the chain rule was applied correctly, and at once. When errors were made, they were usually of the type shown in Tab. 1. Students who applied the chain

rule step by step (*i.e.*, similar to our task model in Fig. 3), usually made *notational* errors. They were confused with the use of the  $x$ 's,  $y$ 's, and  $z$ 's in the  $dx$  notation. Sometimes,  $dz/dx$  and  $d\bar{y}/dz$  were computed correctly, but the result was incorrectly combined (wrong bracketing, oversights with “back-substitutions”). Most turns occurred when students performed algebraic transformations, either to rewrite terms to forms that allow for the application of derivation rules, or simplifying terms that resulted from the application of such rules.

Given a student's attempt to solve a differentiation problem, SLOPERT can search the solution graph for a node that matches the student's answer. A diagnosis of student input is successful if the input term *matches* the contents of the graph's root node, or the content of any node in the root's subgraph. The notion of *matching* is complex, however. Reconsider the first dialogue fragment in Fig. 2. In **S-2a**, the student succeeds in rewriting the original statement. SLOPERT cannot confirm the correctness of the step by itself, since it only has rules for symbolic differentiation. Therefore, we have interfaced SLOPERT to the PRESS algebra system [3]. Fortunately, PRESS can transform the tutor's initial term to the one given by the student, so that the student's contribution can be judged as correct. The interpretation of **S-2b** is trickier as the student correctly applies the general power rule and does *some* tidying-up, simplifying  $5 \cdot -\frac{1}{2} \cdot (x^6 + 2x)^{-\frac{3}{2}} (6x^5 + 2)$  to  $-\frac{5}{2}(x^6 + 2x)^{-\frac{3}{2}} (6x^5 + 2)$ . SLOPERT's solution graph only contains the first expression, which is semantically equal to the student term, but syntactically different. Asking PRESS to simplify it is of only limited help, since it results into a term that is closer but syntactically different to **S-2c**. It will be thus necessary to extend the PRESS simplification rules in a manner similar to our approach. This will allow PRESS to generate step-by-step solutions for algebraic transformations, and to diagnose errors in this process.

## 6 Related Work and Conclusion

Instead of giving instructions to the student, (our) tutors use hints to get the student to construct the knowledge. Research shows that hints facilitate active construction of knowledge [4, 7]. *Problem solving hints* can help students retrieve information and then to use it to make an inference and solve the problem [6].

We found off-the-shelves computer algebra systems of only limited use to inform hinting and the provision of other effective remedial feedback in intelligent tutoring systems. While these expert reasoning engines can solve a vast number of math problems, their algorithms are optimised for generality and efficiency, and thus, rarely mirror or mechanise human/student problem solving behaviour.

SLOPERT follows the footsteps of the BUGGY project [2], which started with studying students doing simple algebra, e.g., subtraction. *Procedural networks* were devised to represent each mislearned procedural skill independently. Resulting diagnostic models reflected students' understanding of the skills and sub-skills involved in a task. To reproduce erroneous student behavior, buggy variants of individual sub-skills were added to the procedural network. BUGGY then correctly diagnosed student errors like “the student subtracted the smaller digit in each column from the larger digit regardless of which is on top”.

The Cognitive Tutors of Carnegie Learning are based on cognitive models of student problem solving following the ACT-R theory of learning [1]. Production rules capture students multiple strategies and their common error. They describe how a problem solving goal is rewritten to another goal, in a correct or buggy fashion. A model tracer exploits a system of production rules to follow students through their individual approaches to solve a problem. It uses and updates estimates of how well the student knows each (correct and buggy) production rule. This allows for the generation of feedback that is sensitive to the problem solving context. The authoring of production rules usually involves a huge investment from math tutors, as we can also confirm. Moreover, huge rule sets have a detrimental effect on the performance of the model tracer algorithm. Tutoring systems based on model tracing thus tend to provide *immediate* feedback to keep the student on track of recognised problem solving paths.

BUGFIX proves that remedial feedback does not have to be immediate feedback. Henneke's work provides an efficient implementation of a diagnosis algorithm that can consider several billion different student calculations for a given task, without keeping the student waiting [5]. BUGFIX also separates domain from pedagogical expertise. A *separate* tutoring module could exploit its diagnostic capabilities to generate well-informed remedial feedback.

*Conclusion.* The feedback of our tutors often depended on the problem solving state and the diagnosis of student actions with respect to correctness, and possibly, goal-direction. To mimic this behaviour, an *intelligent* tutoring system needs access to deep reasoning and diagnosis facilities. AI researchers can contribute to the design and implementation of algorithms that search the space of possible solutions efficiently. Pedagogical experts can inform the design of expert and buggy rules, and provide feedback strategies. The separation of domain expertise from pedagogical expertise can help studying, formalising, and operationalising effective human tutorial feedback. This research reported in this paper contributes to this approach of designing intelligent tutoring systems.

## References

1. J. R. Anderson, C. F. Boyle, A. T. Corbett, and M. W. Lewis. Cognitive modeling and intelligent tutoring. *Artificial Intelligence*, 42:7–49, 1990.
2. J. S. Brown and R. R. Burton. Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, 2:155–192, 1978.
3. A. Bundy and B. Welham. Using meta-level inference for selective application of multiple rewrite rule sets in algebraic manipulation. *Artificial Intelligence*, 16(2):189–212, 1981.
4. A. C. Graesser, N. K. Person, and J. P. Magliano. Collaborative dialogue patterns in naturalistic one-to-one tutoring. *Applied Cognitive Psychology*, 9:495–522, 1995.
5. M. Henneke. *Online Diagnose in intelligenten mathematischen Lehr-Lern-Systemen*. PhD thesis, Universität Hildesheim, 1999.
6. G. Hume, J. Michael, A. Rovick, and M. Evens. The use of hints as a tutorial tactic. 15th Cognitive Science Conf., pages 563–568. Lawrence Erlbaum Associates, 1993.
7. D. C. Merrill, B. J. Reiser, M. Ranney, and J. G. Trafton. Effective tutoring techniques: Comparison of human tutors and intelligent tutoring systems. *Journal of the Learning Sciences*, 2(3):277–305, 1992.