Zuerst ersch. in : KI 2013: Advances in Artificial Intelligence : 36th Annual German Conference on AI, Koblenz, Germany, September 16-20, 2013 ; Proceedings / edited by Ingo J. Timm, Matthias Thimm. - Berlin : Springer, 2013. - S. 272-283. - (Lecture notes in computer science ; 8077). - ISBN 978-364-24094-2-4
http://dx.doi.org/10.1007/978-3-642-40942-4_24

Algorithmic Debugging for Intelligent Tutoring: How to Use Multiple Models *and* Improve Diagnosis

Claus Zinn

Department of Computer Science, University of Konstanz
Funded by the DFG (ZI 1322/2/1)
claus.zinn@uni-konstanz.de

Abstract. Intelligent tutoring systems (ITSs) are capable to intelligently diagnose learners' problem solving behaviour only in limited and well-defined contexts. Learners are expected to solve problems by closely following a *single* prescribed problem solving strategy, usually in a fixed-order, step by step manner. Learners failing to match expectations are often met with incorrect diagnoses even when human teachers would judge their actions admissible. To address the issue, we extend our previous work on cognitive diagnosis, which is based on logic programming and meta-level techniques. Our novel use of Shapiro's algorithmic debugging now analyses learner input independently against *multiple* models. Learners can now follow one of many possible algorithms to solve a given problem, *and* they can expect the tutoring system to respond with improved diagnostic quality, at negligible computational costs.

1 Introduction

The intelligent tutoring community aims at building computer systems that simulate effective human tutoring. A key building block is the diagnoser that analyses learner input for correctness with regard to the current problem solving context. Much of the intelligent behavior of state-of-the-art tutoring systems is due to learning interactions that highly constrain learners' scope of action. Usually, learners are expected to solve a given problem by executing the steps of a *single* prescribed procedure. User interfaces ask learners to enter their answers in a structured and often piece-meal fashion, and systems intervene after each and every problem-solving step, preventing learners to pursue their own (correct or potentially erroneous) problem-solving paths. The tight leash between tutoring system and learners has more practical than pedagogical reasons. While human tutors are capable of dealing with free discovery interactions and recognizing and accommodating alternative problem-solving strategies, most machine tutors only possess a fixed *single* problem solving strategy to diagnose anticipated input.

In [9], we propose a novel method to trace learners' problem solving behaviour using the programming technique *algorithmic debugging*. In [10], we interleave our method with program transformations to perform *deep* cognitive analyses. While our method has many benefits (*e.g.*, low authoring costs, any-time feedback), so far, it only uses a single frame of reference. In this paper, we extend

our previous work by comparing learner behaviour independently against *multiple* models. Our new contribution permits learners to exhibit a wider range of problem solving strategies, while at the same time *improving* diagnostic quality, with little additional computational cost.

The paper is structured as follows. Sect. 2 introduces the cognitive task multi-column subtraction as tutoring domain. It presents the technique of algorithmic debugging, and our adaptation of the method to support reasoning about learner input. Sect. 3 is the main part of the paper, and shows how our variant of algorithmic debugging can be improved to track learner behaviour across multiple models, while at the same time improving the quality of the diagnosis. Sect. 4 discusses related work, while Sect. 5 concludes.

2 Background

2.1 Encoding Cognitive Task Models in Prolog

Cognitive task analysis (CTA) aims at giving a qualitative account of the processing steps an individual problem solver takes to solve a given task. Our research uses logic programming as vehicle for encoding cognitive task models and for tracking and diagnosing learner behaviour. Our domain of instruction is multi-column subtraction, a well-studied domain in the ITS community. Its expert model(s), resulting from CTA, can be concisely encoded in Prolog. In the *Austrian method* (AM), see Fig. 1, sums are processed column by column, from right to left. The predicate `subtract/2` determines the number of columns, and calls `mc_subtract/3`, which implements the recursion (decrementing the column counter `CC` at each recursive call). The clause `process_column/3` gets a partial sum, processes its right-most column and takes care of borrowing (`add_ten_to_minuend/3`) and payback (`increment/3`) actions. A column is represented as a term `(M, S, R)` representing minuend, subtrahend and result cell. If the subtrahend `S` is greater than the minuend `M`, then `M` is increased by 10 (borrowing) before the difference `M-S` is taken. To compensate, the `S` in the column left to the current one is increased by one (payback). – The introduction of the column counter `CC` is not an essential part of the subtraction method, but a technical requirement for mechanising the Oracle part of our diagnosis engine (see below).

We have also been implementing three other subtraction algorithms that reuse code fragments from Fig. 1. The *trade-first* variant of the Austrian method (TF) performs the same steps as the Austrian method, but in a different order; first, all payback and borrowing operations are executed, then all differences are taken. The *decomposition* method (DC) realizes the payback operation by decrementing minuends rather than incrementing subtrahends. The *left-to-right* method (LR) processes sums in the opposite direction; also, payback operations are performed on result rather than minuend or subtrahend cells.

2.2 Algorithmic Debugging for Tutoring

Shapiro's algorithmic debugging technique defines a systematic manner to identify bugs in programs [6]. In the top-down variant, the program is traversed from

```
subtract(PartialSum, Sum) :- length(PartialSum, LSum),
  mc_subtract(LSum, PartialSum, Sum).
mc_subtract(_,  [], []).
mc_subtract(CC, Sum, NewSum) :-
  process_column(CC, Sum, Sum1),
  shift_left(Sum1, Sum2, ProcessedColumn), CC1 is CC - 1,
  mc_subtract(CC1, Sum2, SumFinal),
  append(SumFinal, [ProcessedColumn], NewSum).

process_column(CC, Sum, NewSum) :-
  butlast(Sum, LastColumn),    allbutlast(Sum,RestSum),
  subtrahend(LastColumn, Sub), minuend(LastColumn, Min),
  ( Sub > Min
  -> ( add_ten_to_minuend(CC, LastColumn, LastColumn1),
       take_difference(CC, LastColumn1, LastColumn2),
       butlast(RestSum, LastColumnRestSum), allbutlast(RestSum, RestSum1),
       increment(CC, LastColumnRestSum, LastColumnRestSum1),
       append(RestSum1,[LastColumnRestSum1,LastColumn2],NewSum) )
  ;  ( take_difference(CC, LastColumn, LastColumn1),
       append(RestSum,[LastColumn1], NewSum) ) ).

shift_left( SumList, RestSumList, Item ) :-
  allbutlast(SumList, RestSumList), butlast(SumList, Item).

add_ten_to_minuend(CC, (M,S,R),  (M10,S, R) ) :- irreducible, M10 is M+10.
increment(         CC, (M,S,R),  (M,  S1,R) ) :- irreducible, S1 is S+1.
take_difference(   CC, (M,S,_R), (M,  S, R1)) :- irreducible, R1 is M-S.

minuend( (M,_S,_R), M).                 subtrahend( (_M,S,_R), S).
```

Fig. 1. Multi-column subtraction (Austrian method)

the goal clause downwards. At each step during the traversal of the program's AND/OR tree, the programmer is taking the role of the *oracle*, and answers whether the currently processed goal holds or not. If the oracle and the buggy program agree on the result of a goal G, then algorithmic debugging passes to the next goal on the goal stack. Otherwise, the goal G is inspected further. Eventually an *irreducible agreement* will be encountered, hence locating the program's clause where the buggy behaviour is originating from. In [9], we turn Shapiro's algorithm on its head: instead of having the oracle specifying how the assumed incorrect program should behave, we take the expert program to take the role of the buggy program, and the role of the oracle is filled by a student's potentially erroneous answers. An irreducible disagreement between program behaviour and given answer then indicates a student's potential misconception. In [9], we have also described how to mechanise the Oracle by reconstructing learners' answers from their submitted solution. In [10], we refine the Oracle to complement irreducible disagreements with the attributes *incorrect*, *missing*, or *superfluous*. We then interleave algorithmic debugging with program transformation to

incrementally reconstruct, from the expert program, an erroneous procedure that the learner is following, allowing deep diagnoses of learners with multiple bugs.

Example. Our algorithmic debugger, given the learner's answer to $401 - 199$:

$$\begin{array}{lccc} & 4 & {}^{1}0 & {}^{1}1 \\ - & 1 & 9 & \boxed{10} \\ \hline = & 3 & 1 & 2 \end{array}$$

and the Austrian method (see Fig. 1), yields the following (abbreviated) dialogue:

```
do you agree that the following goal holds:

  mc_subtract( 3, [(4, 1, R1), ( 0, 9, R2), ( 1, 9, R3)],
                  [(4, 2, 2),  (10, 10, 0), (11, 9, 2)])    |: no.

  process_column(3, [(4,1,R1), (0,9,R2),  (1, 9,R3)],
                    [(4,1,R1), (0,10,R2), (11,9,2)])        |: no.

  add_ten_to_minuend(3, (1,9,R3), (11,9,R3 )                |: yes.

  increment(2, (0, 9, R2), (0, 10, R2))                     |: no.

=> irreducible disagreement: ID = increment(2, (0,9,R2), (0,10,R2))
```

Whenever the learner submits a solution, such a dialogue can be automatically generated, and hence, the irreducible disagreement deduced. Compared with existing methods for cognitive diagnosis, our method has a number of advantages. To locate learners' errors, it requires only an expert model, which is an executable Prolog program; no representation of buggy knowledge is required. Moreover, learners are no longer limited to providing their solution in a piecemeal fashion; algorithmic debugging easily copes with input that spans multiple, potentially erroneous, problem-solving steps. Like many other approaches, however, our method only supported a single expert model to solve a given task; it thus fails to recognise learners using algorithms different than the prescribed one. The application of our variant of Shapiro's algorithm on the Austrian method will return irreducible disagreements ("errors") for learners who *correctly* follow one of the other three subtraction algorithms. Moreover, when learners follow one of the other algorithms *incorrectly*, error diagnosis will return incorrect analyses as they are based on the assumption of the Austrian method. Clearly, tutoring systems shall be less prescriptive when asking learners to tackle problems.

3 Input Analysis across Models

The diagnostic engine of our tutoring system shall be able, *e.g.*, to cope with learners following any of the four subtraction methods, or erroneous variants thereof. The method reported in [9] must be generalized.

Step	Austrian (AM)	Trade-first (TF)	Decomposition (DC)	Left-to-right (LR)
1	$\begin{array}{r} 4\,0\,{}^{1}1 \\ -\ 1\,9\,9 \\ \hline = \end{array}$	$\begin{array}{r} 4\,0\,{}^{1}1 \\ -\ 1\,9\,9 \\ \hline = \end{array}$	$\begin{array}{r} 4\,0\,{}^{1}1 \\ -\ 1\,9\,9 \\ \hline = \end{array}$	$\begin{array}{r} 4\,0\,1 \\ -\ 1\,9\,9 \\ \hline = 3 \end{array}$
2	$\begin{array}{r} 4\,0\ \ {}^{1}1 \\ -\ 1\,9_{1}\,9 \\ \hline = \end{array}$	$\begin{array}{r} 4\,0\ \ {}^{1}1 \\ -\ 1\,9_{1}\,9 \\ \hline = \end{array}$	$\begin{array}{r} 4\,{}^{1}0\,{}^{1}1 \\ -\ 1\,9\ \ 9 \\ \hline = \end{array}$	$\begin{array}{r} 4\,{}^{1}0\,1 \\ -\ 1\,9\ \ 9 \\ \hline = 3 \end{array}$
3	$\begin{array}{r} 4\,0\ \ {}^{1}1 \\ -\ 1\,9_{1}\,9 \\ \hline = \quad 2 \end{array}$	$\boxed{\begin{array}{r} 4\,{}^{1}0\,{}^{1}1 \\ -\ 1\,9_{1}\,9 \\ \hline = \end{array}}$	$\begin{array}{r} 4\,9\,\cancel{{}^{1}0}\,{}^{1}1 \\ -\ 1\,9\quad 9 \\ \hline = \end{array}$	$\begin{array}{r} 4\quad {}^{1}0\,1 \\ -\ 1\quad 9\ \ 9 \\ \hline = 2\,\cancel{3} \end{array}$
4	$\begin{array}{r} 4\,{}^{1}0\,{}^{1}1 \\ -\ 1\,9_{1}\,9 \\ \hline = \quad 2 \end{array}$	$\begin{array}{r} 4\quad {}^{1}0\,{}^{1}1 \\ -\ 1_{1}\,9_{1}\,9 \\ \hline = \end{array}$	$\begin{array}{r} 3\,\cancel{4}\,9\,\cancel{{}^{1}0}\,{}^{1}1 \\ -\ 1\quad 9\quad 9 \\ \hline = \end{array}$	$\begin{array}{r} 4\quad {}^{1}0\,1 \\ -\ 1\quad 9\ \ 9 \\ \hline = 2\,\cancel{3}\,1 \end{array}$
5	$\begin{array}{r} 4\quad {}^{1}0\,{}^{1}1 \\ -\ 1_{1}\,9_{1}\,9 \\ \hline = \quad 2 \end{array}$	$\begin{array}{r} 4\quad {}^{1}0\,{}^{1}1 \\ -\ 1_{1}\,9_{1}\,9 \\ \hline = \quad 2 \end{array}$	$\begin{array}{r} 3\,\cancel{4}\,9\,\cancel{{}^{1}0}\,{}^{1}1 \\ -\ 1\quad 9\quad 9 \\ \hline = \quad 2 \end{array}$	$\begin{array}{r} 4\quad {}^{1}0\,{}^{1}1 \\ -\ 1\quad 9\ \ 9 \\ \hline = 2\,\cancel{3}\,1 \end{array}$
6	$\begin{array}{r} 4\quad {}^{1}0\,{}^{1}1 \\ -\ 1_{1}\,9_{1}\,9 \\ \hline = \quad 0\ \ 2 \end{array}$	$\begin{array}{r} 4\quad {}^{1}0\,{}^{1}1 \\ -\ 1_{1}\,9_{1}\,9 \\ \hline = \quad 0\ \ 2 \end{array}$	$\begin{array}{r} 3\,\cancel{4}\,9\,\cancel{{}^{1}0}\,{}^{1}1 \\ -\ 1\quad 9\quad 9 \\ \hline = \quad 0\quad 2 \end{array}$	$\begin{array}{r} 4\quad {}^{1}0\,{}^{1}1 \\ -\ 1\quad 9\ \ 9 \\ \hline = 2\,\cancel{3}\,0\,\cancel{1} \end{array}$
7	$\begin{array}{r} 4\quad {}^{1}0\,{}^{1}1 \\ -\ 1_{1}\,9_{1}\,9 \\ \hline = 2\ \ 0\ \ 2 \end{array}$	$\begin{array}{r} 4\quad {}^{1}0\,{}^{1}1 \\ -\ 1_{1}\,9_{1}\,9 \\ \hline = 2\ \ 0\ \ 2 \end{array}$	$\begin{array}{r} 3\,\cancel{4}\,9\,\cancel{{}^{1}0}\,{}^{1}1 \\ -\ 1\quad 9\quad 9 \\ \hline = 2\quad 0\quad 2 \end{array}$	$\begin{array}{r} 4\quad {}^{1}0\,{}^{1}1 \\ -\ 1\quad 9\ \ 9 \\ \hline = 2\,\cancel{3}\,0\,\cancel{1}\,2 \end{array}$

Fig. 2. Problem Solving States in Four Algorithms for Multi-Column Subtraction

3.1 Correct Learner Behaviour

For the time being, consider learners being perfect problem solvers. They will solve any given subtraction problem by consistently following one of the four aforementioned subtraction algorithms. At any problem solving stage, we would like to identify the algorithm that they are most likely following. Fig. 2 depicts all correct states for solving the subtraction problem $401 - 199$ in each of the four algorithms. Fig. 3 illustrates the basic idea of our approach, which we will later refine. Given some partial learner input at the root (the learner is correctly executing the trade-first variant, see boxed third step in Fig. 2), we run algorithmic debugging on each of the four subtraction methods. As a result, we obtain four different diagnoses. With regard to AM, the learner failed to take the difference in the ones column; for TF, he failed to increment the subtrahend in the hundreds column; for DC, he failed to decrement the minuend in the tens; and for LR, he failed to take the difference in the hundreds column.

This result is unsatisfactory as we cannot derive which of the algorithms the learner is following. For this, let us consider the number of *agreements before*

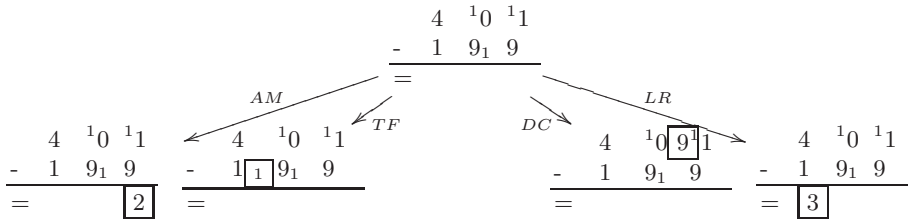

Fig. 3. Algorithmic Debugging (AD) for Four Algorithms on Identical Input

an *irreducible* disagreement between program and learner behaviour is identified. While all methods disagree on their respective top goal `subtract/2`, there are varying *reducible* disagreements that contain partial *agreements*. For AM, while disagreeing with the goal `process_column/3` (ones column), we have **two** agreements with regard to its observable actions in the subgoals `add_ten_to_minuend/3` (ones column) and `increment/3` (tens column). For TF, we have an agreement with regard to `process_column/3` in the ones column, but a disagreement for this clause in the tens column. However, we find an agreement with one of its subgoals, namely `add_ten_to_minuend/3`, yielding a total of **two** agreements. For DC, while disagreeing with the goal `process_column/3` in the ones column, we find **two** agreements, one with its subgoal `add_ten_to_minuend/3`, and one with the partial execution of the decrement operation. And for LR, there are **zero** agreements before the irreducible disagreement is found. In summary, the number of agreements only indicates that the learner is most likely *not* following the LR method; all other methods receive two agreements.

First Refinement. To better rank the methods, we now take into account the size of the code pieces that are being agreed upon. For this, we count the number of *irreducible agreements before* the first irreducible *disagreement*. Fig. 4 depicts the relevant execution trace of AM for the task $401 - 199$. All leafs that are marked "irreducible" (see Fig. 1) have weight 1; the weights of nodes are accumulated upwards. For brevity, `borrow` represents `add_ten_to_minuend/3`, and `payback` represents `increment/3`. With this refinement, there is no change in the agreement score for AM, DC and LR – as all agreements are on leafs nodes. For TF, however, the score increases by one; our agreement on `process_column/3` in the ones column now contributes a value of 2 rather than 1 (as the TF implementation of this predicate has the two leaf nodes `add_ten_to_minuend/3` and `increment/3`). For the given example, the refinement thus yields the intended diagnosis; in fact, this holds for most problem solving steps given in Fig. 2.

Evaluation. Each of the four matrices in Fig. 5 shall be read as overlay to Fig. 2. Fig. 5(a), *e.g.*, gives the results of analysing each problem solving step – performed in each of the four subtraction methods – in the context of AM. Learners perfectly executing AM receive "full marks" when their actions are evaluated against AM; when their actions are evaluated against the other three

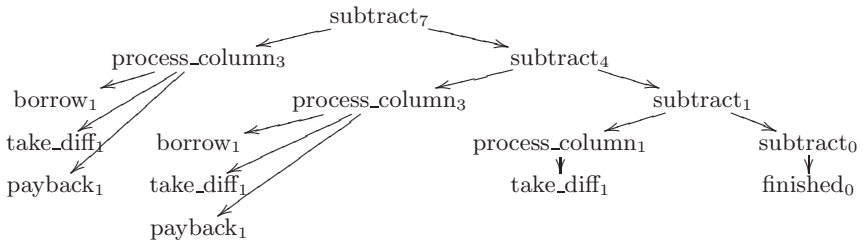

Fig. 4. Execution Trace for the Austrian Method on $401 - 199$

methods, they receive a lesser score. It shows that our first refinement weighting yields the correct diagnoses in all cases.

AM	TF	DC	LR
1	1	1	0
2	2	1	0
3	2	1	0
4	2	2	0
5	5	2	0
6	6	2	0
7	7	2	0

(a) Ref: AM

AM	TF	DC	LR
1	**1**	1	0
2	**2**	1	0
2	**3**	2	0
3	**4**	2	0
5	**5**	2	0
6	**6**	2	0
7	**7**	2	0

(b) Ref: TF

AM	TF	DC	LR
1	1	**1**	0
1	1	**2**	0
1	2	**3**	0
2	2	**4**	0
2	2	**5**	0
2	2	**6**	0
2	2	**7**	0

(c) Ref: DC

AM	TF	DC	LR
0	0	0	**1**
0	0	0	**2**
0	0	0	**3**
0	0	0	**4**
1	1	2	**5**
1	1	2	**6**
1	1	2	**7**

(d) Ref: LR

Fig. 5. Evaluation Matrix (Correct Learner Behaviour)

3.2 Buggy Learner Behaviour

Learner errors are rarely random but result from correctly executing a procedure that has been acquired incorrectly [7,8]. This makes it possible to analyse learner input in terms of expert models or buggy deviations thereof. We distinguish *problematic cases* that cannot be reliably associated with expert models, *standard cases* that can be associated with expert models, and *complex error patterns* that can only be matched to carefully designed buggy models, see Fig. 6.

Problematic Cases. Consider Fig. 6(a), where the student always subtracted the smaller from the larger digit. With no payback and borrowing operation observed, there cannot be sufficient evidence to link learner behaviour to any of the four algorithms. Nevertheless, algorithmic debugging returns useful diagnoses: the irreducible disagreement indicates that the learner failed to borrow in the ones (AM, TF, DC) or tens (LR) column. In Fig. 6(b), the learner forgets to payback after borrowing. Since the main differentiator between the four methods is the location of the payback operation. we cannot, in principle, differentiate between the methods. Again, our algorithmic debugger yields the same type of diagnosis: the learner forgot to payback, either by failing to increment the subtrahend in the tens column (AM, TF), or by failing to decrement the minuends in the tens (and hundreds) column (DC), or by failing to decrement the result cell

	4	0	1
-	1	9	9
=	3	9	8

(a) Smaller from larger digit

	4	$^{1}0$	$^{1}1$
-	1	9	9
=	3	1	2

(b) Forgot to payback

	4	$^{1}0$	$^{1}1$
-	1_{1}	9_{1}	9
=	2	0	3

(c) Wrong subtraction fact

	~~3~~4	$9^{\cancel{1}}\cancel{0}$	$^{1}1$	0
-	1	9	9	9
=	2	0	2	0

(d) Zero digit confusion

	1	$^{1}1$	$^{1}2$	3
-		4	9_{1}	0_{1}
=	1	7	2	2

(e) Combining LR with AM

Fig. 6. Three Groups of Frequent Errors

in the hundreds column (LR). It shows that the comparison of learner behaviour against a set of models strengthens the credibility of the diagnosis.

Standard Cases. Erroneous behaviour that is not directly related to borrowing and payback can be better associated with one of the four subtraction methods. In Fig. 6(c), the learner has followed AM or TF, but got a basic subtraction fact wrong in the ones column. In Fig. 6(d), the student has followed DC, but showed a misconception wrt. columns where the minuend is zero (ones column); in this case, the learner takes the result cell to be zero as well. Note that both errors occur – with respect to AM and DC – early in the problem solving process, and both solutions have no other errors. When we run algorithmic debugging on Fig. 6(c), we obtain wrt. AM and TF an incorrect difference in the ones column (**2** agreements). For DC, the learner failed to decrement minuends in the tens and hundreds (**2** agreements); and for LR, there is a superfluous increment of the subtrahend in the ones (**0** agreement). With each of AM, TF and DC sharing the same number of irreducible agreements, we cannot select one diagnosis over the other. Running the diagnoser on Fig. 6(d), we obtain no agreement for AM, TF, and DC (failed to borrow in the ones column), and also no agreement for LR (superfluous decrement of minuend in the thousands). Given that Fig. 6(d) is almost correct wrt. DC, we need to further refine our algorithm to better recognise the method learners are following.

Second Refinement. When errors occur early in the problem solving process, our simple algorithm for method recognition must perform poorly. Now, instead of only counting the number of irreducible agreements *before* the first irreducible disagreement, we also take into consideration irreducible agreements *after* the first and any subsequent irreducible disagreements. *I.e.*, once an irreducible disagreement between expert model and learner behaviour has been identified, our algorithmic debugger now continues to trace through and analyse the execution tree until all agreements and disagreements have been counted, see Fig. 7.

With this refinement, we now get these (dis-)agreement scores for Fig. 6(c): For AM and TF, we obtain **6** irreducible agreements (*i.e.*, correct cell modifications), and **1** irreducible disagreement (erroneous difference in the ones column). For DC, we have **4** irreducible agreements (correct borrow in the ones; initiated

```
1: NumberAgreements ← 0, NumberDisagreements ← 0
2: Goal ← top-clause of subtraction routine
3: Problem ← current task to be solved, Solution ← learner input to task
4: procedure ALGORITHMICDEBUGGING(Goal)
5:     if Goal is conjunction of goals (Goal1, Goal2) then
6:         ← algorithmicDebugging(Goal1)
7:         ← algorithmicDebugging(Goal2)
8:     end if
9:     if Goal is system predicate then
10:        ← call(Goal)
11:    end if
12:    if Goal is not on the list of goals to be discussed with learners then
13:        Body ← getClauseSubgoals(Goal)
14:        ← algorithmicDebugging(Body)
15:    end if
16:    if Goal is on the list of goals to be discussed with learners then
17:        SystemResult ← call(Goal), given Problem
18:        OracleResult ← call(Goal), given Problem and Solution
19:        if results agree on Goal then
20:            Weight ← computeWeight(Goal)
21:            NumberAgreements ← NumberAgreements + Weight
22:        else
23:            if Goal is leaf node (or marked as irreducible) then
24:                NumberDisagreements ← NumberDisagreements + 1
25:            else
26:                Body ← getClauseSubgoals(Goal)
27:                ← algorithmicDebugging(Body)
28:            end if
29:        end if
30:    end if
31: end procedure
32: Score ← NumberAgreements − NumberDisagreements
```

Fig. 7. Pseudo-code: Top-Down traversal of model, keeping track of (dis-)agreements

payback in the tens; correct differences in the tens and hundreds), and **5** irreducible disagreements (wrong difference in the ones; two superfluous increments of the subtrahend in the tens and hundreds; incorrect minuends in the tens and hundreds as payback is not fully carried out). For LR, we yield **4** irreducible agreements (correct borrowing in the ones and tens and correct differences in the tens and hundreds), and **3** irreducible disagreements (incorrect difference in the ones, and two superfluous increments of the subtrahends in the tens and hundreds). Combining the (dis-)agreements, we get for AM/TF the highest score $(6-1=5)$, and hence correctly recognize that the learner followed this method. Our scoring for Fig. 6(d) also correctly determines that the learner followed DC.

Complex Error Patterns. Some learner input is too erroneous to be associated with any of the available expert models. Consider Fig. 6(e), where the learner is mixing-up two expert algorithms, applying the Austrian method from left to

right. Running our algorithmic debugger against all four expert models will yield the following irreducible (dis-)agreements. For AM and TF, $2-8=-6$; for DC, $2-8=-6$; and for LR, $4-6=-2$. All diagnoses acknowledge that learners performed two correct borrow operations, but missed that corresponding paybacks were performed, albeit at wrong positions. While the LR method is identified as the most likely candidate, the diagnoses are unsatisfactory; they are not sufficiently close to the compound diagnosis "combines two algorithms". Here, it is advisable to complement expert with buggy models to capture such complex erroneous behaviour. If we add the respective buggy model to the existing expert models, a run of algorithmic debugging against the resulting five models clearly associates the learners' solutions in Fig. 6(e) with the buggy model.

4 Related Work

Logic Programming Techniques in Tutoring. There is only little recent research in the ITS community that builds upon logic programming and meta-level reasoning techniques. In [1], Beller & Hoppe also encode expert knowledge for doing subtraction in Prolog. To identify student error, a fail-safe meta-interpreter executes the Prolog code by instantiating its output parameter with the student answer. While standard Prolog interpretation would fail on erroneous outputs, a fail-safe meta-interpreter can recover from execution failure, and can also return an execution trace. Beller & Hoppe then formulate *error patterns*, which they match against the execution trace; with each match indicating a plausible student bug. It is unclear, however, how Beller & Hoppe deal with learner input that cannot be properly diagnosed against some given model, as the chosen model sets the stage for all possible execution traces and the patterns that can be defined on them. Their approach would need to be extended to multiple models, including a method to rank matches of error patterns to execution traces.

In Looi's tutoring system [4], Prolog itself is the domain of instruction, and diagnosing learner input is naturally defined in terms of analysing Prolog code. Learners' programs are debugged with the help of different LP techniques such as the automatic derivation of mode specifications, dataflow and type analysis, and heuristic code matching between expert and student code. Moreover, Looi employs Shapiro's algorithmic debugging techniques [6] in a standard way to test student code with regard to termination, correctness and completeness. It is interesting that Looi also mechanised the Oracle. Expert code that most likely corresponds to given learner code is identified and executed to obtain Oracle answers. *Given* the variety and quality of the expert code, Looi's approach should be able to track learners following multiple solution paths.

In [3], Kawai et al. also represent expert knowledge as a set of Prolog clauses; Shapiro's Model Inference System (MIS) [6], following an inductive logic programming approach, is used to synthesize learners' (potentially erroneous) procedure from expert knowledge and student answers. Once the procedure to fully capture learner behaviour is constructed, Shapiro's Program Diagnosis System, based upon standard algorithmic debugging, is used to identify students' misconceptions, *i.e.*, the bugs in the MIS-constructed program. The inductive

approach helps addressing the issue that learners may follow one of many possible solution paths, *given* that the expert knowledge used for synthesis is carefully designed.

Both Kawai et al. and Looi's work use algorithmic debugging in the traditional sense, thus requiring an erroneous procedure. By turning Shapiro's algorithm on its head, we are able to identify simple and common learner errors by only using expert models. For the diagnosis of more complex error patterns, our approach naturally admits the use of additional, buggy, models. Our taking into account of multiple models adds to the robustness and the quality of the diagnosis, esp. given our well-defined criteria for differentiating between models.

Model Tracing Tutors. Most intelligent tutoring systems are based on production rule systems [2]. Here, declarative knowledge is encoded as working memory elements (WMEs) of the production system, and procedural knowledge is encoded as IF-THEN rules. *Model tracing* allows the recognition of learner behaviour: whenever the learner performs an action, the rule engine tries to find a sequence of rules that can reproduce the learner's action – and update the working memory correspondingly. With the successor state identified, the system can then provide adaptive feedback. Model tracing tutors, however, have two major drawbacks; high authoring costs, and the need to keep learners close to the correct solution path to tame the combinatorial explosion in the forward-reasoning rule engine.

We focus on the authoring cost. As production rule systems are forward-chaining, goal-directness must be induced by preconditions that check whether goal-encoding WMEs hold, or postconditions that maintain goal stacks or perform sub-goaling. Moreover, as conditions are framed in terms of WMEs, there is little if any abstraction. The programmer thus has the tedious burden to give a correct and complete specification of a rule's pre- and postconditions, glueing-together declarative and procedural knowledge. This makes rules verbose, and hence less readable and maintainable. When each expert rule is associated with buggy variants, a cognitive model of multi-column subtraction can grow quickly to more than fifty rules. Their authoring becomes increasingly complicated, costly, and a process prone to error. It is thus not surprising that there is no tutoring system based on production rules that supports more than a single algorithm for solving a given task. This is in line with O'Shea et al, who argues that model tracing systems have had only limited success in modelling arithmetic skills. They only "build single-level representations, with no support for modelling multiple algorithms" [5, p. 265].

5 Conclusion

In our previous work, we presented a variant of algorithmic debugging that compares learner action against a single expert model. We have extended our approach to multiple models. Our refined method now continues past the first and any subsequent irreducible disagreements until the entire execution tree of a Prolog program has been traversed. In the process, agreements and disagreements are being counted, and the code size being agreed upon taken into account.

These numbers are used to identify, among all available models, the algorithm the learner is most likely following. For the domain of multi-column subtraction, we have illustrated the effectiveness of our approach. Perfect learners get correct diagnoses at any stage of their problem solving process. Our approach is also robust for erroneous problem solving. For this, we have distinguished three cases. For problematic cases, where the lack of a central skill prevents any discrimination between given expert models, our multi-model analysis yields, nevertheless, a consistent set of irreducible disagreements that clearly indicates the missing, central skill in question. We have also illustrated the effectiveness of our method for standard cases where the central skills are observable, but other errors are made. For complex error pattern, where learners exhibit central skills, but perform them in a seemingly untimely or chaotic manner, our method is equally applicable. By complementing expert models with buggy models, and by subsequently analysing learner input in the context of both expert and buggy models, we yield diagnoses of high accuracy. Our extension has three benefits. First, the tutoring system can be less prescriptive as learners can now follow one of many predefined algorithms to tackle a given problem. Second, the quality of the diagnosis improves despite of the wider range of input that is taken into account. Third, the improvement comes with little computational costs.

References

1. Beller, S., Hoppe, U.: Deductive error reconstruction and classification in a logic programming framework. In: Brna, P., Ohlsson, S., Pain, H. (eds.) Proc. of the World Conference on Artificial Intelligence in Education, pp. 433–440 (1993)
2. Corbett, A.T., Anderson, J.R.: Knowledge tracing: Modeling the acquisition of proc. knowledge. User Modeling and User-Adapted Interaction 4, 253–278 (1995)
3. Kawai, K., Mizoguchi, R., Kakusho, O., Toyoda, J.: A framework for ICAI systems based on inductive inference and logic programming. New Generation Computing 5, 115–129 (1987)
4. Looi, C.-K.: Automatic debugging of Prolog programs in a Prolog Intelligent Tutoring System. Instructional Science 20, 215–263 (1991)
5. O'Shea, T., Evertsz, R., Hennessy, S., Floyd, A., Fox, M., Elson-Cook, M.: Design choices for an intelligent arithmetic tutor. In: Self, J. (ed.) Artificial Intelligence and Human Learning: Intelligent Computer-Aided Instruction, pp. 257–275. Chapman and Hall Computing (1988)
6. Shapiro, E.Y.: Algorithmic Program Debugging. ACM Distinguished Dissertations. MIT Press (1983); Thesis (Ph.D.) – Yale University (1982)
7. VanLehn, K.: Mind Bugs: the origins of proc. misconceptions. MIT Press (1990)
8. Young, R.M., O'Shea, T.: Errors in children's subtraction. Cognitive Science 5(2), 153–177 (1981)
9. Zinn, C.: Algorithmic debugging to support cognitive diagnosis in tutoring systems. In: Bach, J., Edelkamp, S. (eds.) KI 2011. LNCS, vol. 7006, pp. 357–368. Springer, Heidelberg (2011)
10. Zinn, C.: Program analysis and manipulation to reproduce learner's erroneous reasoning. In: Albert, E. (ed.) LOPSTR 2012. LNCS, vol. 7844, pp. 228–243. Springer, Heidelberg (2013)