Automatic Generation of Analogous Problems to Help Resolving Misconceptions in an Intelligent Tutor System for Written Subtraction

Christina Zeller and Ute Schmid

Cognitive Systems Group, University of Bamberg
An der Weberei 5, 96045 Bamberg, Germany
{Christina.Zeller,Ute.Schmid}@uni-bamberg.de

Abstract. In domains involving procedural skills such as mathematics or programming, students often are prone to misconceptions which result in erroneous solutions. We present the ASG algorithm for generation of analogous problems of written subtraction as an extension of an intelligent tutor system (ITS) proposed by Zinn (2014). The student module of this ITS does not rely on an error library but uses algorithmic debugging where an erroneous solution is recognized by identifying which expert rules fail when trying to reproduce the student solution. Since the ITS allows students to create their own subtraction problems, feedback generation must be online and automatic. ASG is a constraint-based algorithm for constructing problems which are structurally isomorphic to the current, erroneously solved student problem.

1 Introduction

Learning to perform written subtraction mainly relies on the acquisition of procedural knowledge as is characteristic for mathematics or programming (Anderson & Reiser, 1985; Young & O'Shea, 1981). Students errors mostly can be attributed to procedural bugs (Brown & Burton, 1978) such as missing or faulty rules or the use of rules in wrong contexts. That is, incorrect solutions can be regarded as manifestation of fundamental misconceptions based on erroneous rules. Consequently, to support students while learning a procedural skill, it is crucial to identify these misconceptions and provide suitable feedback to overcome them. We can assume that an experienced human tutor is able to diagnose misconceptions and provide helpful feedback. An intelligent tutor system (ITS) should provide similar helpful support.

An ITS is a computer-based system, that monitors the problem-solving of a student, coaches, instructs, and counsels the student (Sleeman & Brown, 1982). It is composed of four modules: (a) expert knowledge module, (b) student model module, (c) tutoring module, and (d) user interface module (Nwana, 1990). The *expert knowledge module* contains the knowledge of the tutored domain. The *student model module* is used to dynamically capture the current knowledge and misconceptions of the student. The *tutoring module* realizes principles of teaching, and the *user interface module* addresses the interaction with the student.

In Proceedings of the ICCBR 2016 Workshops. Atlanta, Georgia, United States of America

In the context of learning procedural skills, typical error libraries are used which consist of a set of faulty rules explaining observed errors in student solutions (Anderson & Reiser, 1985; Brown & Burton, 1978; Burton, 1982; Narciss & Huth, 2006; Young & O'Shea, 1981). The diagnostic program DEBUGGY, for example, contains 130 primitive and compound bugs (Burton, 1982). The advantage of error libraries is that misconceptions are represented explicitly and thereby can support the construction of specific feedback. However, the set of stored faulty rules might be incomplete. In this case, it is not possible to identify the underlying misconception in a student solution and to give specific feedback.

An alternative to error libraries is algorithmic debugging (AD; Murray, 1988) which originally was introduced to identify errors in programs (Shapiro, 1983): Given a set of input/output examples, a program is automatically tested and the program part which is assumed to be responsible for the erroneous output is identified. Zinn (2014) uses this idea in the context of ITSs: The rules representing the expert knowledge for written subtraction are applied to the current problem given to the student. If the expert and student solution differ, the difference is described by the failing rule in the expert module. In this case the tutor model has to generate a strategy to support the student to resolve this misconception. One approach is to present the same or additional problems which a student has to solve until a correct solution is provided; a more elaborate approach is to give bug-related feedback (Narciss & Huth, 2006). In the domain of written subtraction, early systems focused on the identification of misconceptions and did not provide a tutoring strategy (Brown & Burton, 1978; Burton, 1982; Young & O'Shea, 1981). The focus of Zinn's system is also on diagnosis, in addition, feedback is given in form of a written statement describing the identified erroneous rule (Zinn, 2014) while Narciss and Huth (2006) provide an explanation together with a worked-out example.

Such a worked-out example for an incorrectly solved subtraction problem can be viewed as an analogy (Gick & Holyoak, 1983). That is, the example serves as a base for the current (target) problem. For the base problem the correct solution can be demonstrated step by step. Given structural isomorphy between the worked-out example and the current problem, the student should be able to transfer the solution steps (Gick & Holyoak, 1983; VanLehn, 1998). Narciss and Huth (2006) rely on predefined analogies stored together with predefined student problems. However, a more natural learning environment should allow students to input own problems and let the system provide feedback for these—for the ITS unknown—problems. In this paper, the ASG algorithm for automatic generation of analogous subtraction problems is presented. The analogies are constructed such that they capture these characteristics responsible for the diagnosed error and consequently provide support for the supposed underlying misconception.

In the following, we first introduce the expert knowledge for written subtraction. Then, the AD approach used to generate the student model is described. Afterwards, the use of analogies as tutoring principle is introduced and details are given about how a helpful analogy can be constructed with the ASG algorithm. Finally, we conclude with a short review and pointers to further work.

2 Expert Knowledge of Written Subtraction

Knowledge concerning the process of solving a problem is stored in the expert knowledge module of the ITS. Zinn (2014) implemented the widely used *decomposition method* in Prolog. An illustration of the algorithm which only works correct when the result is greater or equal to zero together with an example is shown in Figure 1.[1] Subtraction is defined by five production rules:

- `subtract` $[C_n, C_{n-1}, ..., C_1]$: subtracts a subtrahend from a minuend. The procedure gets as input a non empty list of columns with $C_i = (m_i, s_i, d_i)$ where m_i stands for minuend, s_i for subtrahend and d_i for the difference of the column i. C_1 belongs to the rightmost and C_n to the leftmost column. If the subtrahend has fewer positions than the minuend, leading zeros are added.
- `process_column` C_i: starts with the rightmost column and compares m_i and s_i. If $m_i \geqslant s_i$ the production rule `take_difference` is applied immediately. Otherwise, a borrowing procedure is needed previously, which is the application of `decrement` and `add_ten_to_minuend`. After processing column i the next column $(i + 1)$ is inspected. The `process_column` rule ends the subtraction algorithm after processing the last column ($i = n$, cf. Fig. 1a).
- `decrement` C_i: borrows *ten* from the minuend m_{i+1}. If $m_{i+1} = 0$, further borrowing is needed in C_{i+2} (cf. Fig. 1b).
- `add_ten_to_minuend` C_i: adds the borrowed *ten* to m_i.
- `take_difference` C_i: takes the difference $m_i - s_i$ and stores the difference in d_i.

The algorithm in Figure 1 induces an automaton for the generation of arbitrary subtraction problems (see Fig. 2) which is the backbone of the ASG algorithm.[2] A subtraction problem starts with the rightmost column which needs borrowing (arrow $\mathfrak{m}^{+1\mathfrak{o}}$) or not (arrow $\mathfrak{m}$). From the second column onward a column can be borrowed from (arrow $\mathfrak{m}_{-1}$) or be borrowed from and needs borrowing simultaneously (arrow $\mathfrak{m}_{-1}^{+1\mathfrak{o}}$). In this case, it can be discriminated whether the value of the minuend is 0 ($\mathfrak{o}_{-1}^{+1\mathfrak{o}}$) or not ($\not\mathfrak{o}_{-1}^{+1\mathfrak{o}}$; cf. Fig. 1b). The states of the automaton constitute column cases, that is, they characterize the structural relation between minuend and subtrahend in each column. All problems generated with this automaton can be solved by the provided subtraction algorithm.

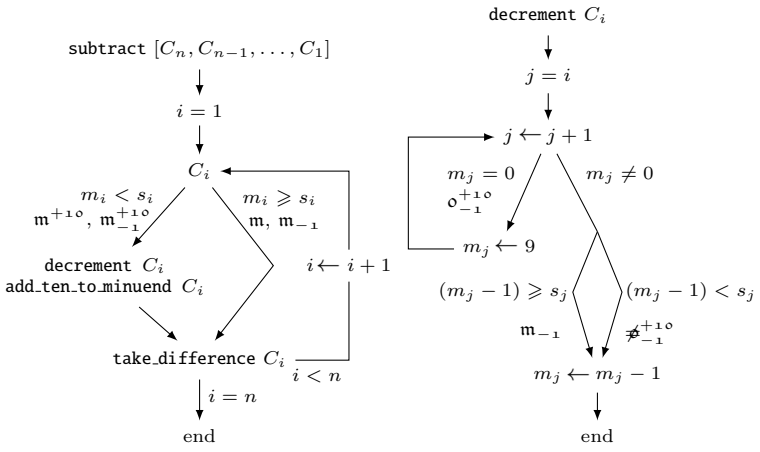
3 Diagnosing Student Solutions with AD

Students' current knowledge is represented in the student model of an ITS. Once a student submits a (partial) solution of a problem, it is analyzed and possible misconceptions are diagnosed. In case of written subtraction this could, for example, be a wrong conception of when or how to perform a borrow procedure.

Based on the algorithmic representation of expert knowledge for written subtraction, AD is used to analyze a student solution (Zinn, 2014): The student solution is—counter-intuitively—considered as correct output of written subtraction. That is, while processing the subtraction problem with the expert algorithm, the partial results are compared with the student solution. In case of differences the

[1] In contrast to Zinn (2014) we label columns from right to left.

[2] The algorithm and automaton are also described in Zeller and Schmid (2016).



(a) Subtraction algorithm (b) `decrement` procedure

C_3	C_2	C_1
3	0	5
− 2	0	6

(c) Problem

2	**9**	5
− 2	0	6

(d) `decrement`

2	9	**15**
− 2	0	6

(e) `add_ten_to_minuend`

2	9	15
− 2	0	6
		9

(f) `take_difference`

Fig. 1: Schema of the written subtraction algorithm (a) with a focus on the `decrement` rule (b) (Zinn, 2014), enriched with the column cases ($\mathtt{m}$, $\mathtt{m}_{-1}$, ...) (Zeller, 2015). In (c) a written subtraction problem is given, which is partially solved step by step in (d), (e), and (f).

production rule which failed is considered as buggy. Consequently, this rule is supposed to be faultily represented by the student and causes the misconception.

There are three production rules assumed to be responsible for misconceptions (Zinn, 2013): `take_difference`, `decrement`, and `add_ten_to_minuend`. The student solution is checked for omission or incorrect application of the respective production rule. Further, superfluous applications of rules are assessed. An example of a typical erroneous student solution is given in the upper part of Figure 3. The student correctly processed C_1 and C_3, but in C_2 something went wrong. The student borrowed correctly from C_3, but the `add_ten_to_minuend` in C_2 is missing. Furthermore, the student calculated $s_2 - m_2 = d_2$ instead of $m_2 - s_2 = d_2$. The execution of the expert algorithm for this subtraction problem together with the annotation of the student solution based on AD is given in Figure 4.

First the rightmost column is processed. C_1 is a $\mathtt{m}$ column and therefore only a `take_difference` is needed. The comparison of the expert and student result of `process_column` C_1 shows no difference, thus no further inspection of C_1 is needed. Next C_2 is inspected. Here the results of expert and student solution for `process_column` differ and therefore a closer look on the sub-procedures is

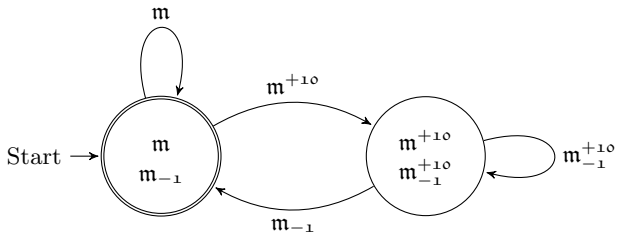


Fig. 2: Automaton to describe the generation of a written subtraction problem, starting with the rightmost column using the column cases annotated in Figure 1.

subtraction problem

	C_3	C_2	C_1
	4	3	7
−	3	7	4

typical student solution

	m_{-1}	m^{+10}	m
	3	3	7
−	3	7	4
	0	4	3

analogous problem

	C_3	C_2	C_1
	4	1	0
−	1	8	0

analogous problem solution

	m_{-1}	m^{+10}	m
	3	11	0
−	1	8	0
	2	3	0

Fig. 3: First row: subtraction problem (with column numbers) and a student solution (with column cases). Second row: analogous problem and its solution.

needed. Whereas the result of `decrement` for the expert and student solution is the same, the difference occurs in `add_ten_to_minuend`. Consequently, AD stops and declares `add_ten_to_minuend` as the student's misconception (cf. Fig. 4).

In contrast to error libraries, where it can be necessary to introduce additional rules to represent compound bugs (Burton, 1982), AD allows to collect all failed applications of the expert production rules (Zinn, 2014). However, when generating feedback for the student, it might be more helpful to deal separately with different misconceptions. Since written subtraction proceeds step-wise from the rightmost column onward, a plausible strategy is to focus feedback on the rightmost error which was detected.

4 Analogies as Tutoring Strategy

The selection of adequate problems as well as the individual feedback for student solutions is realized within the tutoring module (Nwana, 1990). The ITS of Zinn (2014) is designed such that students are free to submit their own subtraction problems. That is, selection of problems is not a component of the tutor module. Therefore, the focus of tutoring is to provide helpful feedback based on the diagnosis of the student model. For example, if `add_ten_to_minuend` is missing in C_2, one can assume that the student struggles with this production rule.

Empirical findings indicate that worked-out examples which are analogous to an erroneously solved problem provide helpful feedback: Problem solving is better facilitated by analogous examples than by written explanations (Reed &

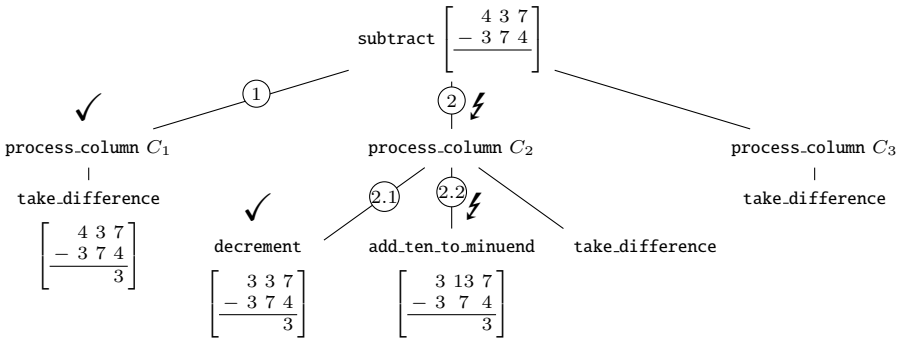


Fig. 4: Execution of the expert algorithm for the problem $437 - 374$ with the annotation of AD. The digits on the edges state the order of production rules application, the ✓ states that expert solution and student solution agree, the ↯ states that expert and student solution differ (cf. Zeller, 2015).

Bolstad, 1991); students are more motivated to engage with a problem domain when examples are given (Narciss & Huth, 2006); the presence of worked-out examples can shorten practice time and reduce errors in a post test (Sweller & Cooper, 1985). Furthermore, providing base and analogous problems allows for structure generalization and thereby induces learning of more general solution routines (Gick & Holyoak, 1983; Goldwater & Gentner, 2015). Accordingly, we extended Zinn's ITS by the ASG algorithm for automatic generation of analogous problems which captures the structural characteristics responsible for the diagnosed error in a student solution. In particular, we assume that the comparison of the own solution with a correct solution of an analogy supports the student to find and correct the own misconception and therefore enhance learning.

In the context of tutoring with analogies, the crucial aspects of analogical problem solving are (Gentner, 1983; Gick & Holyoak, 1983; VanLehn, 1998; Wiese, Konerding, & Schmid, 2008): mapping of the provided example (base) with the student problem (target) and transfer of the rules applied in the base to the target problem to correct the erroneous solution. For a worked-out example, mapping should highlight the student's misconception and consequently induce a revision of the student's knowledge (Renkl, 2014). However, it is not plausible to assume that a student identifies the misconception from the analogy alone, but needs additional hints (Novick, 1988). Our system supports the student by highlighting the relevant column(s) in the graphical user interface (Zeller, 2015).

For successful analogical problem solving it is crucial that the *structural* similarity between base and target is recognized (Novick, 1988). Therefore, when constructing analogous examples, they should have isomorphic structure but vary in superficial features. With respect to written subtraction, problems are structural isomorphic if the same path in the subtraction algorithm (cf. Fig. 1) is used to generate the correct solution. That is, the only difference between base and target problems is the instantiation with different values (Zeller, 2015).

Let be
- $i \in \{1, \dots n\}$ be the column number of a natural number with 1 as the rightmost position,
- m_{Si} the minuend, s_{Si} the subtrahend, and d_{Si} the correct difference of the target problem (for which the student produced an erroneous solution) in column i,
- m_i the minuend, s_i the subtrahend, and d_i the difference of the to be constructed analogous problem at column i.

Start with the rightmost column $i = 1$ and proceed column-wise until the column with the first identified error in the student solution is reached and assign values to the analogous example obeying the following constraints:

- Initialize $m_i, s_i, d_i = \{0, \dots 9\}$ for $i \in \{1, \dots n\}$
- $m_i = m_i \setminus m_{S_i}$, $s_i = s_i \setminus s_{S_i}$

ASG-C – Generation of correct columns:

- If case $\mathfrak{m}$ (no borrowing from column i, no borrowing in column $i+1$): $m_i \geqslant s_i$
- If case $\mathfrak{m}_{-1}$ (borrowing from column i, no borrowing in column $i+1$): $m_i = m_i \setminus 0$, $s_i = s_i \setminus 9$, $d_i = d_i \setminus 9$, $(m_i - 1) \geqslant s_i$, $d_i = (m_i - 1) - s_i$
- If case $\mathfrak{m}^{+1\circ}$ (no borrowing from column i, borrowing in column $i+1$): $m_i = m_i \setminus 9$, $s_i = s_i \setminus 0$, $d_i = d_i \setminus 0$, $m < s$, $d_i = (m_i + 10) - s_i$
- If case $\mathfrak{m}_{-1}^{+1\circ}$ (borrowing from column i, borrowing in column $i+1$): $d_i = ((m_i - 1) + 10) - s_i$
  - $\not{\mathfrak{m}}_{-1}^{+1\circ}$ ($m_{S_i} \neq 0$): $m_i = m_i \setminus 0$, $s_i = s_i \setminus 0$, $d_i = d_i \setminus 0$, $(m_i - 1) < s_i$
  - $\mathfrak{o}_{-1}^{+1\circ}$ ($m_{S_i} = 0$): $m_i = \{0\}$

ASG-E – Treatment of special restrictions for error types:

- If `take_difference` error: $d_{S_i} = d_i$ except
  - If case $\mathfrak{m}$ and $d_{S_i} = 9$: $m_i = \{9\}$, $s_i = \{0\}$ and final column is i
  - If case $\mathfrak{m}_{-1}$ and $d_{S_i} = 8$: $m_i = \{9\}$, $s_i = \{0\}$ and final column is i
  - If case $\mathfrak{m}^{+1\circ}$ and $d_{S_i} = 1$: $m_i = \{0\}$, $s_i = \{9\}$ and final column is $i+1$ with case $\mathfrak{m}_{-1}$
  - If case $\not{\mathfrak{m}}_{-1}^{+1\circ}$ and $d_{S_i} = 1$: $m_i = \{1\}$, $s_i = \{9\}$ and final column is $i+1$ with case $\mathfrak{m}_{-1}$
  - If case $\mathfrak{o}_{-1}^{+1\circ}$: $s_i = 9 - d_{S_i}$ and final column is $i+1$ with case $\mathfrak{m}_{-1}$
- If `add_ten_to_minuend` error: current column case is $\mathfrak{m}^{+1\circ}$ or $\not{\mathfrak{m}}_{-1}^{+1\circ}$ and final column is $i+1$ with case $\mathfrak{m}_{-1}$
- If `decrement` error: current column case is $\mathfrak{m}^{+1\circ}$ or $\not{\mathfrak{m}}_{-1}^{+1\circ}$ and for column $i+1$
  - If case $\mathfrak{m}^{+1\circ}$ or $\not{\mathfrak{m}}_{-1}^{+1\circ}$ ($m_{S_{i+1}} \neq 0$) then final column is $i+1$ with case $\mathfrak{m}_{-1}$
  - If case $\mathfrak{o}_{-1}^{+1\circ}$ ($m_{S_{i+1}} = 0$) then find the last successive column $i+n$ in m_S which contains value 0 and for $j = i+1 \dots i+n$ column case is $\mathfrak{o}_{-1}^{+1\circ}$, final column is $i+n+1$ with case $\mathfrak{m}_{-1}$

Fig. 5: The ASG algorithm for generation of analogous subtraction problems for an erroneous student solution.

To help students to focus on one misconception at a time, in our ASG algorithm, the first identified error is considered while generating the analogy. That is, to keep a balance between overall similarity of base and target and highlighting a specific misconception, we create the analogy by constructing structurally isomorphic columns from the rightmost column until the column with this error is reached. This column is created using constraints helping to identify the error.

The ASG algorithm in Figure 5 is a realization of the automaton given in Figure 2 with the target problem and the diagnosed error as additional constraints. Details of the Prolog implementation can be found in Zeller (2015). Input to ASG are the current (target) problem and the diagnosed error (`take_difference`, `decrement`, `add_ten_to_minuend`). To generate an analogous problem whose solution path is identical for all columns of the target problem up to the diagnosed error column, the column cases ($\mathfrak{m}$, $\mathfrak{m}_{-1}$, $\mathfrak{m}^{+1\circ}$, $\mathfrak{m}_{-1}^{+1\circ}$) have to be considered.

In general, each column i of the minuend, subtrahend and difference of a problem can be a single digit natural number. Accordingly, the solution set for each m_i, s_i, and d_i is initialized with $\{0, \ldots 9\}$. The constraints $m_i = m_i \setminus m_{S_i}$ and $s_i = s_i \setminus s_{S_i}$ exclude the use of the same values in a column for base and target. To generate a column isomorphic to the respective column in the target, the relation given between minuend and subtrahend needs to be preserved in the analogy. This is reflected by the specific constraints induced by each column case. For the example given in Figure 3, in the student solution (target) the rightmost column has the column case $\mathfrak{m}$. Therefore, the constraint in this case is $m_i \geqslant s_i$. In the generated problem, both m_i and s_i were instantiated with 0. There are many further legal instantiations, such as $m_i = 1$ and $s_i = 0$ or $m_i = 9$ and $s_i = 2$. The second column is case $\mathfrak{m}^{+1\circ}$. Because $m_{S_2} < s_{S_2}$, it is necessary to borrow from m_{S_3}. Consequently, m_2 cannot be instantiated with 9 and s_2 cannot be instantiated with 0 and the relation $m_2 < s_2$ needs to hold. Case $\mathfrak{m}^{+1\circ}$ always induces case $\mathfrak{m}_{-1}$ or $\mathfrak{m}_{-1}^{+1\circ}$ for the next column (cf. Fig. 2). In the target problem of Figure 3, Column 3 is a $\mathfrak{m}_{-1}$ case. That is, m_3 cannot be instantiated with 0 and s_3 cannot be instantiated with 9 and the relation $(m_3 - 1) \geqslant s_3$ needs to hold. For case $\mathfrak{m}_{-1}^{+1\circ}$, two special cases have to be considered depending whether the minuend of the column in the target problem is 0 or not.

With the ASG-C part of the algorithm, the set of all structural isomorphic problems to a given problem can be generated. However, most of these problems might not highlight the misconception. Therefore, for the first column from the right for which an error was diagnosed with AD, additional constraints are introduced. In the example in Figure 3, the student made an `add_ten_to_minuend` error. ASG-E restricts the solution set for Column 2 and 3 according to $\mathfrak{m}^{+1\circ}$ and $\mathfrak{m}_{-1}$, respectively. ASG always terminates after the column(s) highlighting the first error with a $\mathfrak{m}$ or $\mathfrak{m}_{-1}$ column (cf. Fig. 2).

If a student makes an `take_difference` error, there exist additional special cases. For example, if a student cannot solve $7-4$, a structural analogous problem (case $\mathfrak{m}$), such as $9 - 1$ seems not to be helpful. To create problems which cover the `take_difference` error, one can make use of arithmetic analogies (Gentner, 1983, p. 156). One arithmetic analogy might be such that the difference of m_i and s_i is identical for both problems (i.e., $d_{S_i} = d_i$). However, this structural restriction cannot be obeyed in all cases. The exceptions are given in ASG-E.[3]

ASG generates structural isomorphic problems by construction (the definitions given in Fig. 5) and it is guaranteed that ASG never returns an empty solution set (proof in Zeller, 2015). However, the generated minuend could include leading 0s and the subtrahend can be 0. Both cases are not plausible analogies. These cases can be excluded by the simple inclusion of additional constraints.

5 Conclusion

We presented the ASG algorithm for online automatic generation of analogous subtraction problems. The constructed analogies specifically address the diag-

[3] Details for `decrement` and `add_ten_to_minuend` error are given in Zeller (2015).

nosed student error, that is, they capture the structural relevant relations between minuend and subtrahend of a student solution and highlight the student's misconceptions. Our work builds on an ITS of Zinn (2014) which does not use an error library (Brown & Burton, 1978; Young & O'Shea, 1981; Narciss & Huth, 2006) but performs diagnosis with AD. This results in a more flexible system, because every possible error in arbitrary subtraction problems can be identified based on the expert production rules. In consequence, the system is not restricted to a predefined set of problems. Instead, students can input their own problems, or problems from text books or assignments.

Our ASG approach allows the generation of structural isomorphic problems for all possible subtraction problems. The solution path of the generated analogy is guaranteed to be identical to the student problem until the erroneous column or columns. Then, the student error is highlighted. Since these characteristics of ASG are guaranteed by construction (proofs see Zeller, 2015) an empirical evaluation of the algorithm is not necessary. However, empirical studies about the impact of ASG generated examples on learning success of elementary school students should be conducted to evaluate the usefulness of our approach.

To investigate whether analogies generated by ASG facilitate the identification of misconceptions and thereby support learning of written subtraction, we plan to conduct several empirical studies. In a first study, we want to compare analogies created by ASG with analogies created by a human tutor. We will construct a set of hypothetical erroneous student solutions which we will present to experienced elementary school teachers. The teachers have to identify the error and are instructed to create an analogous problem which they assume to be helpful to the student. Teacher provided analogies will be compared with the analogies generated by ASG with respect to path congruence of the written subtraction algorithm (cf. Fig. 1).

The AD approach and ASG can easily be adapted to other written subtraction methods. We plan to transfer the concepts underlying AD and ASG to teaching other mathematical operations (written addition, multiplication, and division) and to other domains strongly depending on procedural skills such as teaching computer programming.

References

Anderson, J. R., & Reiser, B. J. (1985). The LISP tutor. *Byte*, *10*, 159–175.

Brown, J. S., & Burton, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, *2*, 155–192.

Burton, R. R. (1982). Diagnosing bugs in a simple procedural skill. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 157–183). Boston, NY: Academic Press.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, *7*, 155–170.

Gick, M. L., & Holyoak, K. J. (1983). Schema induction and analogical transfer. *Cognitive Psychology*, *15*, 1–38.

Goldwater, M. B., & Gentner, D. (2015). On the acquisition of abstract knowledge: Structural alignment and explication in learning causal system categories. *Cognition*, *137*, 137–153.

Murray, W. R. (1988). *Automatic program debugging for intelligent tutoring systems*. San Francisco, CA: Morgan Kaufmann.

Narciss, S., & Huth, K. (2006). Fostering achievement and motivation with bug-related tutoring feedback in a computer based training for written subtraction. *Learning and Instruction*, *16*, 310–322.

Novick, L. R. (1988). Analogical transfer, problem similarity, and expertise. *J. of Exp. Psy.: Learning, Memory, and Cognition*, *14*, 510–520.

Nwana, H. S. (1990). Intelligent tutoring systems: An overview. *Artificial Intelligence Review*, *4*, 251–277.

Reed, S. K., & Bolstad, C. A. (1991). Use of examples and procedures in problem solving. *J. of Exp. Psy.: Learning, Memory, and Cognition*, *17*, 753–766.

Renkl, A. (2014). The worked examples principle in multimedia learning. In R. E. Mayer (Ed.), *The Cambridge handbook of multimedia learning* (Second ed., pp. 391–412). Cambridge University Press.

Shapiro, E. Y. (1983). *Algorithmic program debugging*. Cambridge: MIT Press.

Sleeman, D., & Brown, J. S. (1982). Introduction: Intelligent tutoring system. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 1–11). Boston, NY: Academic Press.

Sweller, J., & Cooper, G. A. (1985). The use of worked examples as a substitute for problem solving in learning algebra. *Cognition & Instruction*, *2*, 59–89.

VanLehn, K. (1998). Analogy events: How examples are used during problem solving. *Cognitive Science*, *22*, 347–388.

Wiese, E., Konerding, U., & Schmid, U. (2008). Mapping and inference in analogical problem solving: As much as needed or as much as possible? In *Proceedings of the 30th CogSci* (pp. 927–932). Mahwah, NJ: Lawrence Erlbaum.

Young, R. M., & O'Shea, T. (1981). Errors in children's subtraction. *Cognitive Science*, *5*, 153–177.

Zeller, C. (2015). *Automatische Erzeugung analoger Beispiele aus Debugging-Traces [Automatic generation of analogue examples from debugging-traces]* (Master's thesis, University of Bamberg, Germany). Retrieved from `http://www.cogsys.wiai.uni-bamberg.de/theses/zeller/ma_zeller-christina_online.pdf`

Zeller, C., & Schmid, U. (2016). Automatic generation of analogous problems for written subtraction. In D. Reitter & F. E. Ritter (Eds.), *Proceedings of the 14th International Conference on Cognitive Modeling (ICCM 2016)* (pp. 241–242). University Park, PA: Penn State.

Zinn, C. (2013). Program analysis and manipulation to reproduce learners' erroneous reasoning. In *LOPSTR 2012* (pp. 228–243). Springer.

Zinn, C. (2014). Algorithmic debugging and literate programming to generate feedback in intelligent tutoring systems. In *KI 2014, LNCS 8736* (pp. 37–48). Springer.