In D. Reitter & F. E. Ritter (Eds.), Proceedings of the 14th International Conference on Cognitive Modeling (ICCM 2016). University Park, PA: Penn State.

Automatic Generation of Analogous Problems for Written Subtraction

Christina Zeller (christina.zeller@uni-bamberg.de)
Ute Schmid (ute.schmid@uni-bamberg.de)
Cognitive Systems Group, University of Bamberg
An der Weberei 5, 96045 Bamberg, Germany

Introduction

Learning in domains such as mathematics or programming, involves the acquisition of procedural knowledge (Young & O'Shea, 1981). For example, when learning written subtraction, students need to understand and apply an algorithm for calculation of differences column by column. Erroneous solutions most often are the result of procedural bugs (Brown & Burton, 1978) such as missing or faulty rules or the application of a rule in the wrong context. If such a procedural bug is diagnosed, a strategy is needed to support the student resolving this bug. Such strategies can be: written explanations, presenting additional problems, or giving bug-related feedback such as an explanation together with a worked-out example (Narciss & Huth, 2006).

A worked-out example can be considered as an analogy to the given problem which a student could not solve correctly (Gick & Holyoak, 1983). That is, for the current (target) problem a structurally isomorphic base problem is provided where the correct solution can be demonstrated step by step. While Narciss and Huth (2006) make use of this feedback approach, they rely on predefined analogies stored together with an—also predefined—set of student problems. However, the automatic generation of such analogous problems for written subtraction can improve and facilitate feedback generation.

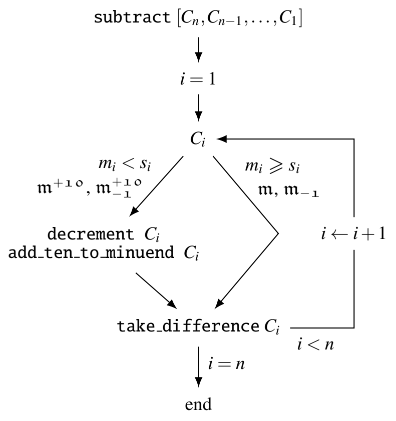
Written Subtraction

In Figure 1 the visualization of the subtraction algorithm using the *decomposition method*, which is implemented in Prolog and described in Zinn (2014), is shown.[1]

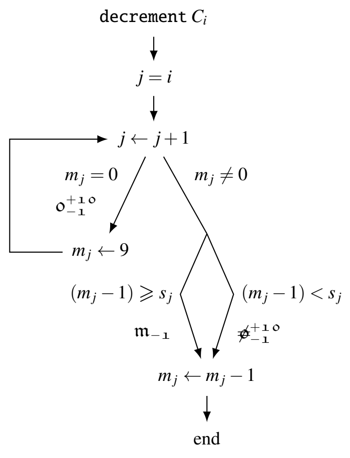
Subtraction is realized by five production rules:

- `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 $m_i \leqslant s_i$ 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).

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



(a) Subtraction algorithm



(b) `decrement` procedure

Figure 1: Schema of the written subtraction algorithm (a) and a closer look on the `decrement` rule (b) (Zinn, 2014), enriched with the column cases ($\mathfrak{m}$, $\mathfrak{m}_{-1}$, ...; Zeller, 2015).

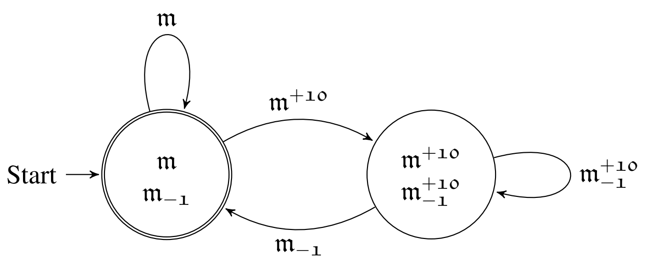


Figure 2: Automaton to describe the generation of a written subtraction problem, starting with the rightmost column using the column cases annotated in Figure 1.

- `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.

Consider the following subtraction problem:

$$\begin{array}{rrrr} & C_3 & C_2 & C_1 \\ & 3 & 0 & 5 \\ - & 2 & 0 & 6 \\ \hline \end{array}$$

The algorithm starts with the rightmost column (C_1). Because of $m_1 = 5$ and $s_1 = 6$ `process_column` calls the borrowing procedure. The minuend of C_2 is 0 and therefore borrowing is needed in C_3. Afterwards, `take_difference` is applied. The application of `decrement` (left), `add_ten_to_minuend` (middle), and `take_difference` (right) results in:

$$\begin{array}{rrrr} & \mathbf{2} & \mathbf{9} & 5 \\ - & 2 & 0 & 6 \\ \hline \end{array} \qquad \begin{array}{rrrr} & 2 & 9 & \mathbf{15} \\ - & 2 & 0 & 6 \\ \hline \end{array} \qquad \begin{array}{rrrr} & 2 & 9 & 15 \\ - & 2 & 0 & 6 \\ \hline & & & \mathbf{9} \end{array}$$

Next C_2 is processed (`take_difference` with $m_2 = 9$ and $s_2 = 0$). After that `take_difference` is applied to the last column (C_3). The correct difference is 99.

Analogies for Written Subtraction

The algorithm in Figure 1 induces an automaton for the generation of arbitrary subtraction problems given in Figure 2. A subtraction problem starts with the rightmost column. A column either needs borrowing (arrow $\mathtt{m}^{+10}$) or not (arrow $\mathtt{m}$). From the second column onward a column can be borrowed from (arrow $\mathtt{m}_{-1}$) or be borrowed from and need borrowing simultaneously (arrow $\mathtt{m}^{+10}_{-1}$). For this most complex case, it can be discriminated whether the value of the minuend is 0 ($\mathtt{o}^{+10}_{-1}$) or not ($\not{\mathtt{o}}^{+10}_{-1}$, 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 subtraction problems generated with this automaton can be solved by the subtraction algorithm given in Figure 1 with the restriction that only such problems are allowed where the result is greater or equal to zero. This automaton was implemented as Prolog program (Zeller, 2015) and generated for instance the following analogous examples:

Problem 1

$$\begin{array}{rccc} & C_3 & C_2 & C_1 \\ & \mathtt{m}_{-1} & \mathtt{o}^{+10}_{-1} & \mathtt{m}^{+10} \\ & 3 & 0 & 5 \\ - & 2 & 0 & 6 \\ \hline \end{array}$$

Problem 2

$$\begin{array}{rccc} & C_3 & C_2 & C_1 \\ & \mathtt{m}_{-1} & \mathtt{m}^{+10} & \mathtt{m} \\ & 4 & 3 & 7 \\ - & 3 & 7 & 4 \\ \hline \end{array}$$

Analogy 1

$$\begin{array}{rccc} & 1 & 0 & 6 \\ - & 0 & 3 & 8 \\ \hline \end{array}$$

Analogy 2

$$\begin{array}{rccc} & 3 & 1 & 0 \\ - & 1 & 8 & 0 \\ \hline \end{array}$$

The column cases of the problem define the structure of the analogy. For example, if C_1 of the problem is of case $\mathtt{m}^{+10}$ then this holds also for the analogy.

Conclusion

The proposed approach was integrated in an intelligent tutor system. There analogous problems were created to specifically address students' errors. That is, the analogous example preserved that characteristics of the given problem where the error occurred.

As a next step we plan an empirical study, where we want to compare automatic generated analogies with analogies created by human tutors. Here, we will start with a set of generated erroneous student solutions. These solutions will be presented to teachers in elementary schools who are experienced in teaching written subtraction. The teachers are instructed (a) to identify the error in the solution, and (b) to propose an analogues problem for which they assume that it helps the student to understand the error. Teacher solutions are analyzed with respect to the constraints of our automatic generation approach.

Furthermore, we plan to transfer the concepts to other domains. On the one hand, we are interested in transfer to related domains, such as teaching other mathematical operations (written addition, multiplication, and division). On the other hand, we are interested in transfer to other domains strongly depending on procedural skills such as teaching computer programming.

References

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

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

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.

Young, R. M., & O'Shea, T. (1981). Errors in children's subtraction. *Cognitive Science*, *5*(2), 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`

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