Towards Automatic Generation of Domain-Specific Mathematical Input Support

Eric Andrès

Bastiaan Heeren

Johan Jeuring

Technical Report UU-CS-2013-012
July 2013

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Towards Automatic Generation of Domain-Specific Mathematical Input Support

Eric Andrès*
CeLTech
Saarbrücken
eric.andres@celtech.de

Bastiaan Heeren
Open Universiteit
Heerlen
bastiaan.heeren@ou.nl

Johan Jeuring
Open Universiteit and
Universiteit Utrecht
J.T.Jeuring@uu.nl

July 10, 2013

Abstract

Providing input when solving a mathematical problem in a technology-enhanced learning system is often a challenging task for a learner. Input editors either provide clickable palettes to construct terms, or require knowledge of some linear syntax. To alleviate this problem, the learning environment ActiveMath was extended with a new interface supporting learners with providing a stepwise solution in the fraction domain. The interface allows learners to insert intermediate steps using pre-defined templates such as *"The least common multiple of* □ *and* □ *is* □*"*, where a blank can be filled in using a dedicated simple input field. Developing similar interfaces for other mathematical domains is labor intensive and error prone. In this article, we investigate how the Ideas domain reasoners can be used to derive the necessary information for the automatic generation of such templates, by making the structure of domain rules explicit using OpenMath expressions.

1 Introduction

The question of how to design a user interface for interactive problem-solving is central to technology-enhanced learning environments. Depending on the level of sophistication of the system, the input mechanism for a learner ranges from a single blank field to input a solution to carefully designed forms that can be considered to provide support by their structure. Building such elaborate input templates is a tedious task, and requires a thorough analysis of the domain and expertise in interface design. In the case of Angle, a cognitive tutor for geometry [5], the interface is based on a specific cognitive model for Euclidean geometry, which tightly binds it to this domain.

For mathematical domains involving formulæ, designing this interface becomes even more delicate given the complexity of formula input per se. Although mathematical input editors such as MathDox and Wiris are widely used, their application in learning

*Partially supported by the DFG - Deutsche Forschungsgemeinschaft, project ATuF ME 1136/8-1 and NA 738/10-1

environments still raises usability issues, as they typically require typing some linear syntax, offer palettes for the construction of formulæ, or allow a combination of these approaches. This places extraneous cognitive load [9] on the learner and can interfere with the learning process. Alternative approaches, such as allowing drag-and-drop to combine formulæ, were tried [6], but so far none were adopted. Formula input is not only tedious for learners, but it also complicates evaluation for the underlying intelligent tutoring system, which may have to deal with erroneous and incomplete input in addition to the evaluation of the (intended) mathematical expression provided by the learner.

The user interface may also have a more general impact on the potential effectiveness of a tutoring system. VanLehn [10] contrasted the impact of eight hypotheses that potentially explain why human tutoring is more effective than tutoring by systems. According to his article, the two most promising hypotheses are that humans are superior in scaffolding and giving feedback. He postulates that a major reason for the more effective use of feedback and scaffolding by human tutors is their management of interaction granularity, which VanLehn defines as *"the amount of reasoning required of participants between opportunities to interact"*. This leads to the *Interaction Granularity Hypothesis*: the smaller the tutoring system's interaction granularity, the higher its potential effectiveness. This has an impact on the interface design of systems supporting problem-solving. In order to support small interaction granularity, the level of detail at which the learner inputs a solution also needs to be refined.

We have developed an interface for stepwise input of problem solutions based on templates that embody general methods used in that domain (e.g. expansion of fractions for fraction problems), which we describe in more detail in Section 2. While the templates are domain-specific, the approach of decomposing a problem into sub-steps is more general and should at least transfer to domains with an algorithmic character. In Section 3, we characterize the essence of sub-steps in our interface. After introducing the Ideas domain reasoners in Section 4, we present an example of automatic sub-step extraction. Finally, we discuss problems we encountered and raise questions for future work.

2 Steps interface

The Steps interface is part of the web-based intelligent learning environment ActiveMath [7]. It is designed in such a way that a learner can freely construct her own solution path for a problem [1]. The Steps interface has been implemented for the domain of fraction addition exercises. Figure 1 shows a screenshot of the Steps interface during the problem-solving process. In the remainder of this section, we will describe features as implemented in the fraction domain.

When an exercise is started, a learner is presented with a single input blank that can be used to supply the overall solution. The basic idea is that the learner inserts as many steps as she needs using an interactive element referred to as a *line*, which allows the manipulation of intermediate steps. A line consists of a drop-down menu from which the learner selects a description of the intention of the step. The menu-items are the

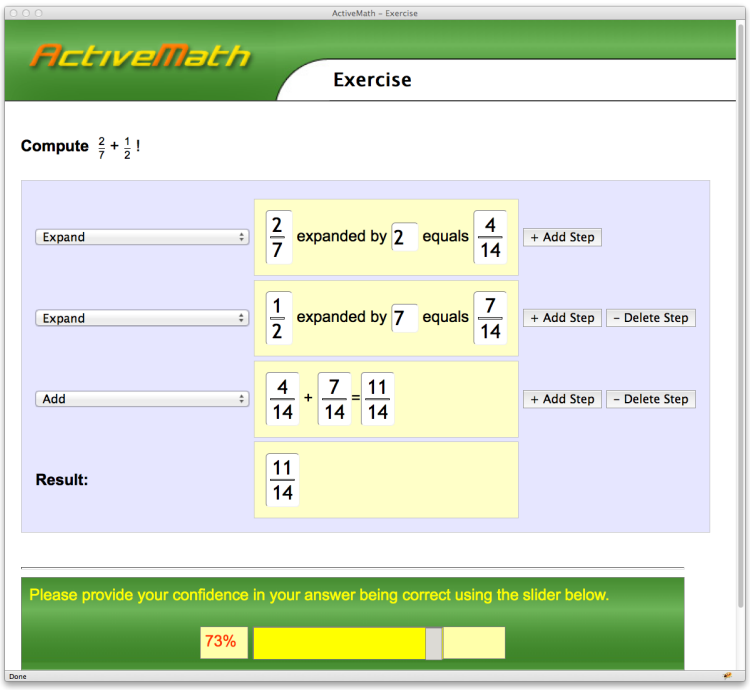

Figure 1: A STEPS exercise in progress

operations that are relevant in fraction-addition problems: expand, reduce, find prime factors, find common denominator, compute inverse, transform, add, subtract, multiply, divide, find least common multiple, and find greatest common divisor. After selecting an operation, a corresponding template to be filled in by the learner is inserted in the *line*. For instance, if the learner chooses `expand fraction`, she will get the template "$\square$ expanded by $\square$ is $\square$", as shown in Figure 1. Steps are added or removed by clicking on the corresponding buttons on the right.

This approach decouples the learner's *intention* (e.g., expanding a fraction) from the actual *execution* (e.g., computing the expansion step) and eliminates the need for heuristics to guess what the learner is trying to achieve. The separation of intention and execution is useful for the system to interpret the learner's input. In addition, by the introduction of sub-steps, the complexity of input to be entered in one single blank can be reduced (e.g., $\boxed{2+3=5}$ vs. $\boxed{2} + \boxed{3} = \boxed{5}$). We replaced the complex input editor usually available in ACTIVEMATH with a dedicated light-weight textfield replacement supporting input of numbers, basic arithmetic operations as well as fraction input.

3 What's in a template?

A template represents the application of a domain method. It has a descriptive name (such as `add`) which is displayed in the drop-down menu. The representation of the template consists of a formula or text interspersed with blanks to be filled in by the

learner. This simple format supports the learner at multiple levels by pre-structuring the input format.

A template contains structural information about the represented method. The `add` template, for instance, is $\square + \square = \square$, which makes it easy to identify summands and the result. Making the + explicit in the template structure gives away that an addition term should contain a + symbol, which in this case is a trivial fact. Distributing the input of the addition term over two blanks reduces the chances of learners struggling with typing mistakes. Still, one has to be careful with the amount of guidance offered by structural help. It can be tempting to create templates that do more than just making input more comfortable or interpretable. The following template for one of de Morgan's laws can certainly be considered as a form of scaffolding for a learner who is not familiar with it, but might not be appropriate in all situations: $\neg(\square \wedge \square) = \square \vee \square$.

Another more subtle form of structural support is obtained by offering a particular input editor to the learner to fill in a blank. A template, in addition to making structure explicit by means of showing blanks, also contains implicit knowledge about the type of input that is expected. In the template "$\square$ expanded by $\square$ is $\square$", it is relatively clear that the first and last blanks should be filled in with fractions, while the second blank should be filled in with a number. This type information can guide instantiation of the blanks: blanks expecting numeric input could be rendered with a very simple input editor, while a blank in a derivative exercise to be filled in with an elaborate function expression could be rendered using a palette-based editor such as WIRIS OM[1].

A template also embodies a certain step granularity. Consider the task of solving a quadratic equation. We could have a single template for the application of the quadratic formula, representing an equivalence with the equation on its left-hand side and the solution set on the right-hand side. An alternative to such a template for a big step could be to define additional templates for the sub-steps of applying the quadratic formula: (i) identification of a, b, and c, (ii) computation of the discriminant Δ, and (iii) specification of solutions. In this way, sets of templates can be used to allow a learner to work on different levels of granularity. Adding templates for algebraic manipulations to the example above would further expand a learner's freedom by allowing the usage of various strategies (e.g., clever factorization).

Templates do not only support learners, they also enable fine-grained diagnostics in the system. A template filled by the learner carries valuable information, such as the intention of the learner, derived from the selection of the template, and the instantiation of the domain method, derived by the filled in blanks. More elaborate information such as correctness, applicability conditions, and strategic cleverness could be computed from the template by domain reasoners.

4 Rewrite strategies, rules, and canonical forms

The Ideas domain reasoners are constructed around rewrite strategies [4] that specify how an exercise can be solved incrementally. A strategy calculates which rule should be

[1] http://www.wiris.com

applied (and how) to solve an exercise, and traces steps submitted by learners. Feedback is calculated automatically from a rewrite strategy and the set of rules that are supported by the domain.

The granularity of steps is an important design consideration in a domain reasoner. The compositional specification of strategies allows for coarse-grained steps that combine several rules. For fine-grained steps, views [2] are used in the rules. A view defines a canonical form and can be used for matching and normalizing expressions. Consider for example the expression $2(x + 3)$ and the rewrite rule $a(b + c) \rightarrow ab + ac$. A view for calculating with natural numbers could take care of the implicit calculation $2 \cdot 3 = 6$, and turn $2(x + 3)$ into $2x + 6$ after distribution. Another view based on the algebraic law $a - b = a + (-b)$ would help to also let the distribution rule work on $2(x - 3)$ by matching the rewrite rule's + symbol in a more liberal way.

Consider the exercise to compute $\frac{2}{7} + \frac{1}{2}$ from Figure 1. The worked-out solution generated by the fractions domain reasoner for this exercise is:

$$\begin{array}{ll} & \frac{2}{7} + \frac{1}{2} \\ = & \quad \{\texttt{expand by 2}\} \\ & \frac{4}{14} + \frac{1}{2} \\ = & \quad \{\texttt{expand by 7}\} \\ & \frac{4}{14} + \frac{7}{14} \\ = & \quad \{\texttt{add fractions}\} \\ & \frac{11}{14} \end{array}$$

The derivation does not show that the domain reasoner starts with an implicit step that computes the least common multiple of the denominators. This multiple is then used for determining how to expand both fractions. The rewrite rule for expansion, $\frac{a}{b} \rightarrow \frac{ac}{bc}$ with $c \neq 0$, is parameterized over the expansion factor (meta-variable c) since it does not appear on the rule's left-hand side. The rewrite rule for adding two fractions, $\frac{a}{c} + \frac{b}{c} = \frac{a+b}{c}$, also performs the addition of two numbers in the numerator.

5 Generation of templates based on Ideas

The derivations generated by Ideas and the structured solutions that can be produced using Steps share similar characteristics. We postulate that this similarity can be exploited to generate domain-specific templates for the Steps interface from Ideas semi-automatically, and even fully automatically for certain domains. In the remainder of this section, we propose an approach to extract templates from a simple domain reasoner for fractions. We first reconsider a part of the fraction example from Figure 1 and make the ingredients of templates more explicit. The template used in the first line contains information about the semantics of the step. It represents the application of *expansion of fractions* to two arguments, and establishes an equality statement between this and the result (the third argument). In addition, it constrains the input by using our simple input editor for numerical input to fill in the blanks.

We represent a template as an OpenMath object. The OpenMath standard is a suitable choice for exchanging information about templates, as it allows us to express

```
<template name="Find LCM">
  <typemap>
    <typeof varname="a" typename="Integer">
    <typeof varname="b" typename="Integer">
    <typeof varname="c" typename="Integer">
  </typemap>
  <OMOBJ>
    <OMA>
      <OMS cd="relation1" name="eq" />
      <OMA>
        <OMS cd="arith1" name="lcm" />
        <OMV name="a" />
        <OMV name="b" />
      </OMA>
      <OMV name="c" />
    </OMA>
  </OMOBJ>
</template>
```

Figure 2: Representation of the Find LCM template with OpenMath

the semantics of the template, and, if necessary, to define new symbols that can be used to convey the intended meaning of a template. Figure 2 shows the OpenMath representation of a template for finding the least common multiple.

The OpenMath object in Figure 2 represents the equation $lcm(a, b) = c$. To obtain a template, the variables in this equation are replaced by blanks. The variable names can also be used to identify the blanks in further communication between the learning environment and the domain reasoner, for instance to provide feedback about the correctness of the blanks filled in by the learner. Information about the types of the variables is stored in a type map, which is used for selecting the input editors for the blanks.

The rewrite rules of Ideas can already be represented as Formal Mathematical Properties (FMPs) in OpenMath [3], and thus it is relatively easy to also let the domain reasoners support the automatic generation of templates. For example, the rewrite rules for finding the least common multiple and expansion are defined as follows:

```
lcmRule :: RewriteRule Expr
lcmRule = describe "Find LCM" $ makeRewriteRule "findlcm" $
   \a b -> lcm a b :~> lcm' a b

expandRule :: RewriteRule Expr
expandRule = describe "Expand by" $ makeRewriteRule "expand" $
   \a b c -> expand (a / b) c :~> (a *! c) / (b *! c)
```

These definitions make use of a Haskell library for datatype-generic rewriting in Haskell [8]. The meta-variables of the rewrite rules are represented as lambda-bound variables in the host language. The rewrite rules contain OpenMath symbols for the operations in the

fraction domain (e.g., function `lcm` for the application of the symbol `arith1.lcm`), but also symbols representing calculations (`lcm'` and operator `*!` for multiplication). In fact, the XML template shown in Figure 2 can be generated from the definition of `lcmRule`, with 1 and 0 as the default depths for replacing subexpressions by blanks in the left-hand side and right-hand side, respectively. The type map is computed from the type signatures of the operations.

Because precise control over the granularity in the generated templates is needed, we expect that adding annotations cannot be fully avoided. Hence, template generation will be semi-automatic. The domain reasoners are extended with the new *ruletemplate* service that given a rule returns a template.

To render the OpenMath-based template representation as Steps templates in the User Interface, we replace the previously hard-coded templates by a call to a new service providing a list of available templates. This list is obtained by parsing template descriptions as returned by the *ruletemplate* service. Labels are extracted from the `name` attribute of the top-level `template` tag.

The current implementation assumes to find the application of `relation1/eq` at the top level of the OpenMath expression to identify left- and right-hand sides. For each of these expressions, OpenMath symbols are presented using pre-authored presentations encoded in the system. `OMV`s are transformed to blank placeholders containing type information extracted from the `typemap`. When the template is finally requested for presentation, these placeholders for blanks are replaced by the actual HTML/JavaScript code for the required mathematical input editor.

6 Conclusion

In this paper we have discussed domain-specific templates for supporting the input of intermediate steps in mathematical exercises, and how these templates can be generated semi-automatically from rule specifications in the Ideas domain reasoners. The generation of templates is based on the close correspondence between the explicit rewrite rules and the implicit calculations in the domain reasoner on the one hand, and the small interaction granularity in the Steps interface on the other hand. The canonical forms that are used in the domain reasoner's rewrite rules can provide enough information about the structure of the templates, as well as the types of the blanks. We have proposed a representation for templates based on the OpenMath standard.

In the future we want to explore whether automatically generated templates providing various levels of structural support can be used for adaptive scaffolding purposes. This could be particularly useful for domains involving term rewriting such as algebraic manipulation of mathematical expressions. We also want to investigate how we can make the tree structure underlying many problem-solving processes more explicit. Another line of research we will follow will focus on using domain reasoners to interpret learner input. Templates extracted from the domain reasoners contain enough information to run fine-grained diagnostics which will allow precise identification of the learner's problems.

Finally, we will investigate how blank locations and types at suitable granularity

levels can be automatically derived from rewrite rules and rewrite strategies in existing domain reasoners.

Acknowledgements

We thank the reviewers for their useful comments and suggestions. Eric Andrès' visit to Utrecht was partially funded by the Utrecht University Research Impulse Educational and Learning Sciences.

References

[1] A. Eichelmann, E. Andrès, L. Schnaubert, S. Narciss, and S. Sosnovsky. Interaktive Fehler-Finde- und Korrektur-Aufgaben. Eine Akzeptanz und Usability-Studie bei Sechst- und Siebtklässlern. In F. Reichl G. Csanyi and A. Steiner, editors, *Digitale Medien - Werkzeuge für Forschung und Lehre*, pages 401–412, 2012.

[2] B. Heeren and J. Jeuring. Canonical forms in interactive exercise assistants. In *MKM'09*, volume 5625 of *LNAI*, pages 325–340. Springer, 2009.

[3] B. Heeren and J. Jeuring. Adapting mathematical domain reasoners. In *MKM'10*, volume 6167 of *LNAI*, pages 315–330. Springer, 2010.

[4] B. Heeren, J. Jeuring, and A. Gerdes. Specifying rewrite strategies for interactive exercises. *Mathematics in Computer Science*, 3(3):349–370, 2010.

[5] K.R. Koedinger and J.R. Anderson. Reifying implicit Planning in Geometry: Guidelines for Model-Based intelligent tutoring system design. *Computers as cognitive tools*, pages 15–46, 1993.

[6] P. Libbrecht and D. Jednoralski. Drag and drop of formulae from a browser. *Proceedings of the MathUI 2006 Workshop*, 2006.

[7] E. Melis, G. Goguadze, M. Homik, P. Libbrecht, C. Ullrich, and S. Winterstein. Semantic-aware components and services of ActiveMath. *British Journal of Educational Technology*, 37(3):405–423, 2006.

[8] T. van Noort, A. Rodriguez Yakushev, S. Holdermans, J. Jeuring, B. Heeren, and J.P. Magalhães. A lightweight approach to datatype-generic rewriting. *Journal of Functional Programming*, 20:375–413, June 2010.

[9] J.G. Van Merriënboer and J. Sweller. Cognitive Load Theory and Complex Learning: Recent Developments and Future Directions. *Educational Psychology Review*, 17(2):147–177, 2005.

[10] K. VanLehn. The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems. *Educational Psychologist*, 46(4):197–221, October 2011.