The Electronic Journal of Mathematics and Technology, Volume 7, Number 2, ISSN 1933-2823

Error-Patterns within "Next-Step-Guidance" in TP-based Educational Systems

Gabriella Daróczy
a1047149@unet.univie.ac.at
MEi:CogSci
University of Vienna
1010 Austria

Walther Neuper
neuper@ist.tugraz.at
Inst.f.Software Technology
Graz University of Technology
8010 Austria

Abstract

Theorem-Prover (TP) based educational mathematics systems are able to cover all phases of stepwise problem solving, a TP checks user-input most generally and reliably; an additional feature is the ability to know the next step towards a solution. However, just presenting the next step is not motivating for the learner; so adaptive user guidance is required.

This paper describes a general approach to adaptive user guidance resulting from an interdisciplinary cooperation between Computer Mathematics and Cognitive Science. The former provides technologies for capturing a general notion of error-pattern and for general services to be used for user guidance by a dialogue component. The latter, Cognitive Science, addresses the dialogue component providing user guidance. After a brief review of guiding principles for learning an, architecture for the dialogue component in a prototype is presented. The interactions between mathematics engine, dialogue component and front-end are described as rules of a knowledge-based expert system.

The prototype is described to an extent which gives a proof of concept for both, the feasibility of general error-patterns in terms of Computer Mathematics as well as appropriateness of the respective services for adaptive user guidance in terms of Cognitive Science.

1 Introduction

This is a case study from an interdisciplinary cooperation between Cognitive Science and Computer Mathematics. The latter is involved by an experimental mathematics assistant, called $\mathcal{ISAC}$ [1], which is based on the Theorem Prover (TP) Isabelle [NPW02]. This assistant is designed as "a model of mathematics" [Neu] for experiments in interactive learning mathematics, comparable to learning chess by use of an interactive chess program — learning by trial and error augmented with the feature of a "transparent system", i.e. the possibility to "look to the justifications from TP and to the underlying mathematics knowledge" — a design which bypassed a wealth of experiences from the science of

[1]http://www.ist.tugraz.at/projects/isac/ and http://www.ist.tugraz.at/isac/

mathematics education and Cognitive Science, experiences which shall not be missed in an upcoming generation of TP-based educational mathematics assistants [Neu10].

In particular, Cognitive Science provides theories on cognitive activities, on learning and on design of educational tools. From this side, the interdisciplinary cooperation is still at the beginning and limited to a concrete goal: find ways to exploit the general and powerful TP-technology for adaptive user-guidance. This early state of cooperation is determined by a lack of systematic approach to software design in terms of Cognitive Science. For instance, the case study side steps the notion of ”misconception” — the ”error-patterns” in the paper at hand are an ad-hoc approach and are not consistently related to the large body of knowledge in the respective literature of misconception in mathematics learning, on Piaget’s or Vygotsky’s theories, etc.

The structure of the paper is as follows: §1 is followed by an introduction of notions from Computer Mathematics in §2 in order to make the paper self-contained: §2.1 briefly explains the technology which provides services for “Next-Step-Guidance”, §2.2 defines error-patterns and related notions in and concludes with a proof for a certain technological improvement triggered by the case study. §3 describes the approach from the side of Cognitive Science: after briefly laying down the framework for approaching mathematics education in §3.1, §3.2 gives the motivation for the case study and formulates the goal for adaptive user-guidance researched in this paper, §3.3 shows concrete example dialogues implemented in the prototype within the case study. How these dialogues exploit the services from Computer Mathematics is shown in §3.4 and in §3.5. The experiences gained during the case study suggest further research and development in §4, before a final conclusion is given in §5.

2 Computer Mathematics: Error-Patterns & Co.

This section presents the computer mathematics underlying next-step-guidance and error-patterns; the presentation is as brief as possible but sufficient for re-implementation. A second goal of this section is to present a proof, that error-patterns improve next-step-guidance — i.e. that the former, error-patterns, requested from the side of Cognitive Science, luckily leads to an improvement of the latter, next-step-guidance, within Computer Mathematics. For that proof some technicalities need to be introduced, where the three central definitions are given in detail and other prerequisites are introduced using a running example.

The running example is simplification of arbitrary rational terms [2], with an arbitrary number of different variables, in particular simplification of the term $\frac{5x}{4y} + \frac{3x}{4y}$. This term (and all other rational terms) are simplified by a program which might look as follows in the $\mathcal{ISAC}$ prototype.

```
Program simplify_Rational (term::rat) =
  LET
    t = TRY (Rewrite_Set prepare) term;
    t = REPEAT
        (Rewrite_Set multiply_divide) OR
        (Rewrite_Set expand_parentheses) OR
        (Rewrite_Set collect_numerals) OR
        (Rewrite_Set reduce_0_1_2) OR
        (Rewrite_Set add) OR
```

[2]In the sequel “term” and “formula” will be used interchangeably.

```
10          (Rewrite_Set cancel) t
11     IN
12       TRY (Rewrite_Set beautify) t
```

The above `Program` implements the functionality of a canonical simplifier [BN98] in Computer Algebra in full generality: The program takes a `term` as argument, `prepare`s the term by execution of the statement `Rewrite_Set` (which replaces binary minus $-$ in $2\cdot x - \frac{3\cdot x}{4\cdot x} \rightarrow_{prepare} 2\cdot x + \frac{-3\cdot x}{4\cdot x}$, for instance [3]), `REPEAT`s applying one set of rules (by `Rewrite_Set`) `OR` another set as long as one of them is applicable. Finally `beautify` is applied, which re-introduces $-$, for instance. More details about the syntax can be found in [HKN10].

Interpreting this program in a specific way called "Lucas-Interpretation", generates a calculation close to paper and pencil calculations, also provides the services for next-step-guidance and, as introduced in this paper, detects error-patterns — how this is accomplished, is concern of the following section.

2.1 "Next-Step-Guidance" from Lucas-Interpretation

Lucas-Interpretation executes a program from break-point to break-point; in the example program the break-points are `Rewrite_Set`. At the break-points specific services are available, when control is handed over to the dialogue; for instance, a next term can be added to the calculation created by interpretation of a statement in the program.

Below, both Lucas-Interpreter and a calculation, are modeled as "labeled terminal transition systems (TTS)" [Plo81]. During interpretation of a program m the transitions $\rightarrow_m$ go from break-point to break-point, here called **tactics** as usually in TP. In a **calculation** the transitions $\rightarrow$ are tactics as well, here responsible for promoting a calculation from the existing formulas to a next formula [4].

For instance, tactic `Rewrite_Set prepare` in line `03` promotes a calculation by replacing all binary minus by unary minus, while nothing is said where the tactic comes (from input by the user or from execution of the program). $\rightarrow_m$ is definitely done by execution of statement `Rewrite_Set prepare` in the program, so $\rightarrow$ and $\rightarrow_m$ are equivalent with respect to the outcome in a calculation and semantics in term of logics, but they differ in technical details (for instance, in the source formulas come from to apply tactic at: the calculation or the interpreter's environment).

Transitions $\rightarrow$ and $\rightarrow_m$ being semantically equivalent, however, does not mean that there is a one-to-one correspondence in the number of transitions in a calculation and in interpretation. If we apply the program of the running example to the term $\frac{5x}{4y} + \frac{3x}{4y}$, `Rewrite_Set prepare` is not applicable and due to `TRY` skipped such that the calculation is not promoted (without `TRY`, the not-applicable `Rewrite_Set prepare` would throw an exception). Then, interpretation of `OR` selects `Rewrite_Set add` and `REPEAT`s as long as one rule-set is applicable. Thus, given several $+$ in a term, `Rewrite_Set add` would `REPEAT`edly promote the calculation. Also `Rewrite_Set beautify` is skipped for the same reason as above.

The states of a calculation c in Def.1 below comprise formulas $\mathcal{P}(F)$ shown to the user and contexts X. The latter are usually not shown to the learner; they store logical data required by provers for checking input and for allowing to prove correctness of output, finally.

[3]Replacing binary minus avoids exponential blowup in the number of rewrite-rules for addition (and subtraction).

[4]Here the semantics of the tactics is defined such that they ensure logical consistency.

The TTS for Lucas-Interpretation operates on states comprising the states of the respective calculation plus **program states** Γ of the interpreted program; Γ comprises environments and locations in the program, as usual. Details about the terminal states, S and S_m respectively, can be found in [Neu12]. There is also a definition of **specification**, which comprises in/output items and pre/post-conditions, as usual. The predicate $initialized\_by\ s, m$ concerns initialization for steps of interpretation by a specification s and a program m (thus relating logics and computation); the respective definition (Def.5 in [Neu12]) is separated in order to shorten the central definition of Lucas-Interpretation.

Lucas-Interpretation provides three basic services for the dialogue, which are more or less directly based on TP; the first service is to "know the next step", Def.6 in [Neu12]:

Definition 1 (Lucas-Interpreter) *Given a specification s, a calculation $c = \langle X, \mathcal{P}(F), T, \rightarrow, S\rangle$ and a program m, then a labeled terminal transition system $\mathcal{L} = \langle \Xi, T_m, \rightarrow_m, S_m\rangle$ is called a Lucas-Interpreter iff*

(i) the configuration $\Xi = (\Gamma \times X \times \mathcal{P}(F))$ contains Γ the program states, X a set of ***contexts*** *and $\mathcal{P}(F)$ a set of sequences F of* ***formulas****, the latter two forming the configuration of calculation c*

(ii) the actions T_m contain (program) tactics in m and T_m is bijectively mapped with T

(iii) the transition relations $\rightarrow^{t_m}$ are steps of interpretation with $\gamma, x, F \rightarrow^{t_m} \gamma', x', F'$, functional with unique γ', x', F', the steps are $initialized\_by\ s, m$.

(vi) the terminal configurations $S_m \subset (\Gamma_m \times X \times \mathcal{P}(F))$ contain Γ_m the terminating states of m.

We say $\mathbf{c}$ ***is_generated_by*** $(\mathbf{s}, \mathbf{m}, \mathcal{L})$.

The above transition system differs from standard program interpretation in the *(i) configuration* extended by contexts X and formulas $\mathcal{P}(F)$ dedicated to the rules of logic in a separate definition (see Def.3 in [Neu12]).

In a straightforward manner the *(ii) actions* are tactics T_m in the program m, *Rewrite_Set* for the running example; these tactics are essentially the same as those available to the user, see Def.2 below. So in *(iii) transition relation* $\rightarrow^{t_m}$ is related to a tactic t_m like *Rewrite_Set*. Tactic t_m can update the context, for instance applying the rule $c \neq 0 \Rightarrow \frac{a \cdot c}{b \cdot c} = \frac{a}{b}$ updates context x with $c \neq 0$ to context x'.

(vi) terminal configurations are related to the post-condition of specification s and are not addressed in this paper.

Checking an input tactic is the second TP-based service in Lucas-Interpretation. The tactic can be input by the student directly or by the dialog in the course of user-guidance. The following definition describes how Lucas-Interpretation handles the input of a tactic t_I; the definition re-uses the predicate $c\ is\_generated\_by\ (s, m, \mathcal{L})$ from Def.1 above:

Definition 2 (Locatable tactics) *Given a specification s, a program m, a Lucas-Interpreter $\mathcal{L} = \langle \Xi, T_m, \rightarrow_m, S_m\rangle$ at configuration $\chi = (\gamma, x, F) \in \Xi$, a calculation c with $c\ is\_generated\_by\ (s, m, \mathcal{L})$*

at configuration (x, F) and finally given an (external) tactic t_I input by the learner, then we have the rule

$$\frac{t_I\ \mathit{applied\_to}\ (x,F) = (x',F') \qquad \gamma, x, F \rightarrow^{*} \gamma', x, F \rightarrow^{t_m} \gamma'', x', F' \qquad t_I \approx t_m}{\gamma, x, F \rightarrow^{t_m} \gamma'', x', F'}$$

*If the rule is applicable (in particular if for t_I an equivalent t_m in the program is found, $t_I \approx t_m$), we say '*t ***is_locatable_at*** *χ' iff applicable; otherwise we say $\mathcal{L}$* ***is_helpless_at*** *χ.*

The definition's rule has $t_I\ \mathit{applied\_to}\ (x, F)$ as premise; this premise concerns applicability of tactic t_I, which in turn relates t_I to the formulas F in calculation c and the respective logical context x; see Def.2 in [Neu12]. If the premise is given, we get a step of calculation c leading to x', F', but if no $t_I \approx t_m$ (a respective tactic t_m in program m) is found during interpretation, we don't get a γ'', a program state to resume interpretation form — $\mathcal{L}$ is *helpless*, no next step can be found anymore.

Note that the transitions $\rightarrow^{*} \cdots \rightarrow^{t}$ searching for $t_I \approx t_m$ only change the program state γ and not the configuration x, F of the calculation; so x', F' resulting from t_I is presented to the user, before control is handed over. This behavior is consistent with transitions being functions and not relations in Def.1. And the strategy implemented by Def.2 works particularly well with programs like the example on p.177.

Checking an input formula concerns the third basic TP-service offered to dialogue guidance. The following definition describes how Lucas-Interpretation handles an input formula f_I:

Definition 3 (Derivable formula) *Given a specification s, a program m, a Lucas-Interpreter $\mathcal{L} = \langle \Xi, T_m, \rightarrow_m, S_m \rangle$, a calculation c with $c\ \mathit{is\_generated\_by}\ (s, m, \mathcal{L})$ at configuration x, F as part of configuration χ of $\mathcal{L}$ and finally given a formula f_I input by the learner, then we have the rule*

$$\frac{\gamma, x, F \rightarrow^{*} \gamma', x', F' \qquad F \vdash_{x'} f_I}{\gamma, x, F \rightarrow^{t^*} \gamma', x', F_I}$$

If the rule is applicable we say '$\mathrm{f_I}$ ***is_derived_from*** *χ'; otherwise we say '*$\mathrm{f_I}$ ***is_not_derivable_from*** *χ'.*

In this case the interpreter executes the next tactics found in program m until a context x' is generated, which can justify the input f_I, i.e. $c \vdash_{x'} f_I$. This justification might involve algebraic simplification as shown in the example program on p.177. If successful, the step is presented to the user, before control is handed over; t^* is a tactic called "ad-hoc derivation"; it usually comprises a sequence of tactics.

If not successful, i.e. the interpreter executes program m until termination and no step is found with $c \vdash_{x'} f_I$, then f_I is "not derivable". This judgment can be costly in resources, since it relies on an (internal) computation of the program, including all the subprograms, until termination.

The difficulties for Lucas-Interpretation from user-input as shown in Def.2 and Def.3 are comparable to difficulties a debugger has: which kinds of user input allow to resume execution/interpretation? §2.3 will give a specific positive answer to this questions with respect to Lucas-Interpretation.

2.2 Error-Patterns and Fill-Patterns

This paper is interested in errors students make when transforming formulas in (symbolic) calculations, which is called "term- rewriting" [BN98]. Here we give a brief introduction to this technique as a prerequisite for subsequent definitions. The technique uses specific theorems: the theorems are equalities (with optional assumptions), are interpreted from left to right and called **rewrite-rules**. For instance given the theorem $c \neq 0 \Rightarrow \frac{a}{c} + \frac{b}{c} = \frac{a+b}{c}$ and the term $\frac{5\cdot x}{4\cdot y} + \frac{3\cdot x}{4\cdot y}$, rewriting the term with the theorem is

$$\frac{5\cdot x}{4\cdot y} + \frac{3\cdot x}{4\cdot y} \quad \longrightarrow_{c\neq 0 \Rightarrow \frac{a}{c}+\frac{b}{c}=\frac{a+b}{c}} \quad \frac{5\cdot x + 3\cdot x}{4\cdot y}$$

with the result under the assumption $4\cdot y \neq 0$. Rewrite-rules constitute **rule-sets**; if they have unique normal-forms, i.e. a form which cannot be simplified further, the rule-sets are called (canonical) **simplifiers**. For a simplifier s we write, for instance

$$\left(\frac{5\cdot x + 3\cdot x}{4\cdot y + 4\cdot y}\right)\downarrow_s = \frac{x}{y}$$

with $\downarrow$ denoting the **normal-form** $\frac{x}{y}$; readers might convince themselves that $\frac{x}{y}$ is equivalent to $\frac{5\cdot x+3\cdot x}{4\cdot y+4\cdot y}$ modulo the usual simplifiers for rationals.

Definition 4 (Error-Pattern) *Given a triple (id, P, R) with id a string called **identifier**, P a set of terms with equality called **patterns** and R a set of **rewrite-rules**, this triple is called an **error-pattern** iff*

(i) $\forall p \in P.\ \exists r \in R.\ match\,(lhs(p), lhs(r)) \neq \emptyset$. *This r is called the rule* **belonging_to p**

(ii) $\forall r \in R.\ \exists p \in P.\ match\,(lhs(p), lhs(r)) \neq \emptyset$

The above definition uses the function *match* as described in [BN98]: the both left-hand sides *lhs* are compared with respect to their term structure.

And this is an example for detecting an error-pattern: Given the formula on the left-hand-side below we assume the formula on the right-hand-side is input by a student

$$\frac{5\cdot x}{4\cdot y} + \frac{3\cdot x}{4\cdot y} = \frac{5\cdot x + 3\cdot x}{4\cdot y + 4\cdot y}$$

where the components of the fraction are mistaken for components of a vector. However, a student needs not input the above exact right-hand-side, but for instance

$$\frac{5\cdot x}{4\cdot y} + \frac{3\cdot x}{4\cdot y} = \frac{8\cdot x}{8\cdot y}$$

or even $= \frac{x}{y}$. With this variety of possible input in mind we use $\downarrow$ in the following definition.

Definition 5 (Error-Pattern detected) *Given a calculation $c = \langle X, \mathcal{P}(F), T, \rightarrow, S\rangle$ at configuration $(x, F) \in X \times \mathcal{P}(F)$ with formula f in F, an error-pattern $\epsilon = (id, P, R)$, a canonical simplifier $\downarrow_s$ and an "input" formula f_I then we say:*

’$\mathbf{at}\ (\mathbf{x}, \mathbf{F})\ \ \epsilon\ \mathbf{is\_detected\_for}\ \mathbf{f_I}$’ *iff* $\exists p \in P.\ (f \rightarrow_p) \downarrow_s = f_I \downarrow_s$.

If less precision is acceptable, we say "the error-pattern is detected" and omit the *at* and *for*. The definition says: An error-pattern is detected if one of the patterns simplified equals the input simplified. In the case of fractions the simplifier would create a normal form with one fraction line and fully canceled. The $\exists$ in the definition is computationally expensive: *no* error-pattern detected requires *all* patterns $p \in P$ to be checked.

So far nothing has been said about the purpose of the R in an error-pattern $\epsilon = (id, P, R)$. The rewrite-rules in R provide a service to the dialogue when helping the student out of an error-pattern: the $p \in P$ directs to a rewrite-rule applicable to the "current" formula (see Def.5), and rewrite-rules can be associated with fill-patterns:

Definition 6 (Fill-Pattern) *Given a rewrite-rule r and an ordered set (enumerable by $\mathcal{N}$, the natural numbers) of triples $\phi = \{\phi_i.\ 1 \leq i \leq n \in \mathcal{N}\ \wedge\ \phi_i = (id_i, p_i, \mathcal{P}_i(id_\epsilon))\}$, where id_i is a string called **identifier**, p_i a term with equality called **fill-in-pattern** and $\mathcal{P}_i(id_\epsilon)$ a set of error-patterns' identifiers, such a set is called a **fill-pattern of** $\mathbf{r}$ iff*

(i) $\forall i.\ 1 \leq i \leq n \Rightarrow lhs(p_i) = lhs(r)$

(ii) $\forall i.\ 1 \leq i \leq n \Rightarrow match(rhs(p_i), rhs(r)) \neq \emptyset$ *and the $lhs(p_i)$ "contain less place-holders with growing i"*

For terms f and f_I we say ’$\mathbf{f_I}\ \mathbf{is\_filled\_into}\ \phi$ ’ *iff* $f \rightarrow_r f_I$.

The third component of a fill-pattern's element, $\mathcal{P}_i(id_\epsilon)$ is planned for further services not discussed in this paper: Given fill-patterns, these can be retrieved from the error-pattern due to *(i)* in Def.4 by a dialogue-service *findFillpatterns*; from these patterns the dialogue can select an appropriate one (see *(ii)* in Def.6) and call the service *requestFillformula*; the fill-formula has some placeholders to fill-in, and the service *inputFillFormula* enforces $f_I\ is\_filled\_into\ \phi$.

In order not to loose the focus on techicalities, here are no examples given for detection of error-patterns and on fill-patterns. Such examples are better understood in the context as introduced by Cognitive Science, so instead forward references here there will be backward references from the educational context in §3.4.

2.3 Next-Step-Guidance is Sustained

Error- and fill-patterns extend the scope of reliable operation within Lucas-Interpretation: user input can make the interpreter go astray, according to Def.2 it can become "helpless" and according to Def.3 a term can be "not_derivable". In the latter case, even if an input term is "derivable", it is even possible that the last tactic in the derivation is *not* "locatable", next-step-guidance can *not* continue — while error- and fill-patterns guarantee "locatability" in this case and thus guarantee sustained next-step-guidance, as the following lemma states.

Lemma 7 (Error-pattern sustains next-step-guidance) *Given a specification s, a program m, a Lucas-Interpreter $\mathcal{L} = \langle \Xi, T_m, \to_m, S_m \rangle$ at configuration $\chi = (\gamma, x, F) \in \Xi$, a calculation c with c $is\_generated\_by$ $(s, m, \mathcal{L})$ and an error-pattern $\epsilon = (id, P, R)$. Then the statement holds:*

If all $r \in R$ are also in a rule-set in m "reachable from program state γ" [5]*, then*

at (x, F) ϵ $is\_detected\_for$ f_I $\implies$ t_r $is\_locatable\_at$ χ

where t_r is the tactic applying r.

Proof. According to Def.5 at (x, F) ϵ $is\_detected\_for$ f_I means that there is an f in F and $\exists p \in P$ which has a redex in f. Def.4*(i)* says that there exists a rewrite-rule r $belonging\_to$ p, thus r has a redex in f as well. So the first premise in Def.2 is fulfilled by the tactic t_r applying r: t_r $applied\_to$ $(x, F) = (x', F')$.
The second premise $\gamma, x, F \to^* \gamma', x, F \to^{t_m} \gamma'', x', F'$ is fulfilled by two steps. First $\to^*$ is ensured because there is still a redex in f, i.e. the normal form has not yet been reached and because the lemma's assumption states that r is in a "reachable" rule-set, say *rs*, too — so r must be applicable also within *rs*.
The second step is simply $\gamma, x, F \to^{t_r} \gamma'', x', F'$ with $t_r \approx t_m$, where t_m is the same rewrite-rule r found in program m. Since all premises in Def.2 are fulfilled, t_r $is\_locatable\_at$ χ. ■

The lemma's assumption gives an important hint on how to design the relation between error-patterns and programs.

3 Cognitive Science: Dialogue Guidance & Co.

This section is a "tour de force" in brevity for a Cognitive Science approach to educational mathematics software; after a brief overview, §3.2 gives the motivation for the case study and identifies a particular question crucial for mathematics learning, §3.3 selects an example used throughout the paper, §3.4 connects to the Computer Mathematics from §2 above focusing this example, and §3.4, §3.5 present the implementation of the example in the prototype's dialogue module.

3.1 Cognitive Science Approaching Mathematics Education

Cognitive science is relatively new in the field of mathematics education. It claims, that mathematics is a product of adaptive human activities, grounded and embodied in everyday cognitive mechanism, involving the process of abstraction, and that mathematical ideas are quite stable for about hundred years [Nun04]. There is a cognitive gap between algebra and arithmetic: the brains of humans, and also of animals, has some innate arithmetic, the so called number sense [Bob96] and [Gel78].

The following questions arise when we take human cognition into consideration in the field of computer supported learning: What is the object of our investigation (what is mathematics [Nun04]), how do people learn and understand mathematics (involving questions about mathematical abstraction [Dre90]), and how does mathematics learning and teaching work in a computer-based learning environment. It is important to think of the cognitive processes and involve them in the design for a software which fits the way human are thinking. Thus software shall adapt to the thinking processes,

[5] Formal treatment of "reachability" is out of scope, but "obvious" within a canonical simplifier, see the program on p.176 and the proof.

and not vice versa, the human would have to adapt to the software. To reach this goal we would need to identify what is important in the learning and teaching process, and how we can answer such an interdisciplinary questions, involving both computer science and human cognition.

However, answering all these questions goes beyond the scope of our paper, but as general guidelines, *ISAC*'s design addresses the following issues: individual learning and thinking at different levels of knowledge, support for stepwise calculation within a selected problem class and feedback involving the learner into interaction. Furthermore blended learning environments engage teachers and broader social environments. It is very important in the process of teaching mathematics with a computer software, that the software gives feedback, reflects on errors [Col06], and provides ”answers” in form of hints when needed.

The process of learning mathematics is often bound to emotions like anxiety and frustration [Hem90], when learners conduct error, especially when the missing knowledge in a single domain influences overall mathematical skills. According to [Bro78]”Errors are made because certain component is missing” and found to be quite pattern like, and depend on complex factors: procedural knowledge, incorrect strategies, mental representation of variables, explanations of teachers, learning environment..etc. According to [Pay90]: ”achieving a ”cognitive diagnosis” of a learners error may be an important step towards meaningful invidiualized tutoring” we focused on identifying error patterns, hints necessary to be detected in order to support learners. However in adaptive user quidance not only the detection is very important, but also the balanced individual feedback. “Balanced” means to give that amount of information learners need to accomplish a current challenge, and not more — accomplishment motivates to tackle the next step in the course of learning by doing.

3.2 Recurring Errors in Mathematics

Scanning the various aspects and challenges in mathematics learning from the side of Cognitive Science as mentioned above, and scanning the potential of “next-step-guidance” from the side of Computer Mathematics as described in §2, we identified the following issue as particularly promising to tackle from both sides:

Introduction of integration *in a calculus course, for instance, captures all attention for advanced concepts and procedures like limit and infinitesimal quantities; however, integration builds upon arithmetic and algebra like* addition of fractions*, for instance — so frustration might be caused, if errors from the basic level, e.g. in adding fractions, recur on the current level of integration and inhibit success in learning (frustration for both, teachers and students, because individual tutoring is limited in classes).*

As mentioned above, lessons cannot easily be structured alongside errors, errors occur individually for each student from a large, but finite set of error-patterns as mentioned above, while a teacher hardly can satisfy the individual requests. So the following goal has been established for the case study:

Generalise automated generation of user-guidance
with concurrent detection of errors
recurring during systematic build-up of mathematics.

Automated generation of user-guidance is successfully accomplished in several tutoring systems, in particular for tutoring symbolic computation of fractions:

Tutors like [And08, MS04] implement powerful generation of adaptive user-guidance in several aspects, in particular in their reaction to erroneous input during stepwise calculation. The power of this kind of tutors results from their conception which addresses modeling mental processes — which, however, causes multiple efforts, when dealing with basic errors recurring in advanced procedures; the respective software structure does *not* reflect the systematic buil-up of mathematics.

Also, tutors based on the concepts of Computer Algebra like [RMD$^+$05, Bee92] very efficiently create user-guidance in stepwise calculation. However, this kind of systems lack logical foundations to combine several procedures of Computer Algebra. The incapability in combining several procedures carries over to error-handling in more complex mathematics problems, for instance in advanced problems in applied mathematics.

Additional to the features of these kinds of tutors, the general technology introduced in §2 is able to model the systematic build-up of mathematics and re-uses elementary procedures for implementing advanced procedures. So, for instance, a program for tutoring integral transformations [Roč] re-uses the procedure for simplification of fractions, and thus makes all the machinery dealing with errors on fractions available also within the advanced topic.

3.3 Example Dialogues for Fractions

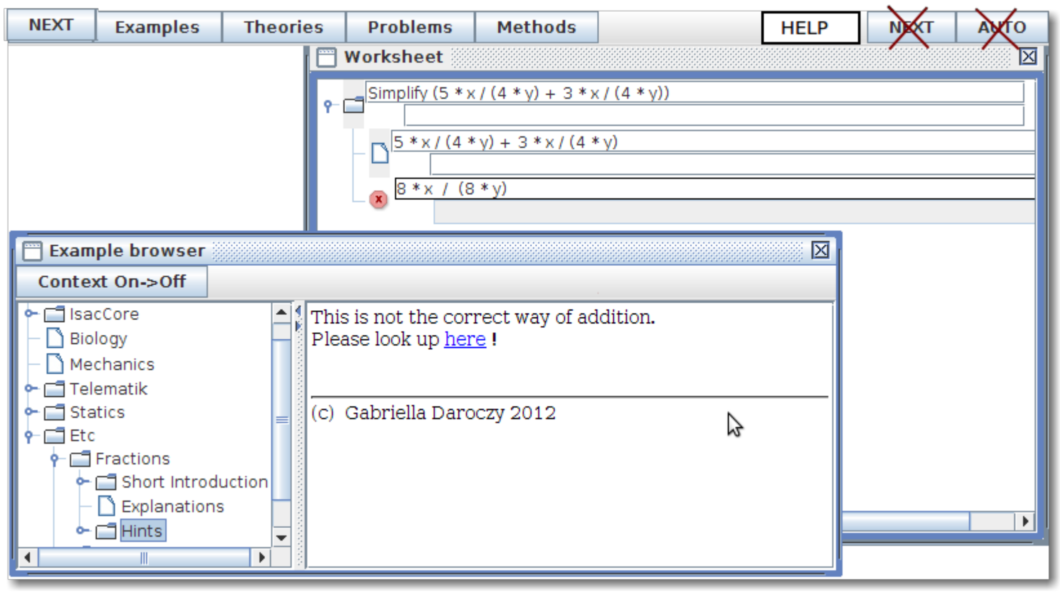
For the case study described in this paper there was, in principle, free choice for all domains of mathematics. In fact the choice was limited to the domains already implemented in the $\mathcal{ISAC}$ prototype. The choice for the domain of fractions has good reasons: Learning in this domain causes well-researched difficulties [Kie90], comprehending fractions involves complex cognitive abilities, one can vary the form of them, e.g. from a mixed form $2\frac{3}{5}$ to $\frac{13}{5}$ or 2.6, one can carry out different operations (addition, multiplication, etc.). Also, fractions establish relationship between two numbers, involve multiple representation (proportion, linear change, simple value, etc). However, it is not a question in this paper how students develop these representations, rather the goal is to develop an interactive dialogue between a learner and software [RAKC07].

In such a dialogue we should react on what the student is doing. For that purpose we need to collect information about the students action. In $\mathcal{ISAC}$ this is information the learner is looking up, the formula he/she is typing in during calculation. This formula might be correct or incorrect. In the case of a correct formula, there is not much to do, in the case of an incorrect formula we can give feedback. As mentioned in the introduction of §3 the interaction is based on reflection, and giving hints if the learner requests for.

The case of input of an incorrect formula shown in the screen-shot Fig.1 will be the running example for the sequel: In Fig.1, as a feedback for the (incorrect) input $\frac{8\cdot x}{8\cdot y}$, a "hint-page" popped up with the hint `"This is not the correct way of addition. Please look up here !"`. Since $\mathcal{ISAC}$ uses HTML technology, dialog authors will be free to exploit the ever increasing features of HTML. In the right upper corner Fig.1 indicates buttons changing during the subsequent interactions.

The running example will proceed from the situation in Fig.1 to interactions, where the learner requests more help and the software presents "fill-forms" with place-holders to be completed by the learner: $\frac{..+..}{..}$, $\frac{..+..}{4y}$, or $\frac{5x+..}{4y}$.

Tab.1 shows the situation in Fig.1 in lines `00..02`. Line `03` assumes the learner pushing the $\boxed{\text{HELP}}$ button appearing instead of $\boxed{\text{NEXT}}$ and $\boxed{\text{AUTO}}$ (which would allow to request the next step



Simplify (5 * x / (4 * y) + 3 * x / (4 * y))

5 * x / (4 * y) + 3 * x / (4 * y)

8 * x / (8 * y)

This is not the correct way of addition.
Please look up here !

(c) Gabriella Daroczy 2012

Figure 1: *ISAC*'s front-end with incorrect input (red ⊗ icon in the *Worksheet*).

or even the final result from the system). Lines 04..05 show the fill-form requested by the learner, but not filled correctly. So in line 06 the system presents the tactics applying $\frac{a}{c}+\frac{b}{c}=\frac{a+b}{c}$ which leads to a correct input by the learner finally. `rule 1-4` in the right-most column will be discussed §3.5.

3.4 Error-Patterns, Rewrite-Rules and Fill-Forms

This section describes how the prototype detects the error-pattern above and how the prototype uses fill-patterns to provide services for adaptive user-guidance.

no.	student's action	user-interface	system's action	dialogue-rule
00		$\frac{5x}{4y}+\frac{3x}{4y}=$		
01	types in $\longrightarrow$	$\frac{8x}{8y}$		
02		$\boxed{\text{hint page 1}}$	$\longleftarrow$ pops up	`rule 1`
03	hit help button $\longrightarrow$	$\frac{..+..}{..}=$	$\longleftarrow$ show fill-form	`rule 2`
04	hit help button $\longrightarrow$	$\frac{..+..}{4y}=$	$\longleftarrow$ next fill-form	`rule 3`
05	hit help button $\longrightarrow$	$\frac{5x+..}{4y}=$	$\longleftarrow$ next fill-form	`rule 3`
06	hit help button $\longrightarrow$	$\boxed{\text{Tactics}}$	$\longleftarrow$ no more fill-form	`rule 4`
07	types in $\longrightarrow$	$\frac{8x}{4y}$	correct	
...				

Table 1: Interactions for EP "add-fractions"

Error-patterns are those introduced by Def.4 in §2.2. Before the implementation is described, we consider design issues, because there is for instance an abundance of learners' errors concerning fractions described in the literature. Many of them have been experienced by the authors as well. They compiled a long list of such errors, which then were separated into groups. Assignment to respective groups was done such that one specific class of feedback can be related to each group. Such specific feedback is the hint-page in Fig.1. These are the groups identified within this case study:

1. Addition $\frac{a}{c}+\frac{b}{c}=\frac{a+b}{2\cdot c}$, $\frac{a}{b}+\frac{c}{d}=\frac{a+c}{b+d}$, $\frac{a}{b}+\frac{c}{d}=\frac{a\cdot c}{b\cdot d}$, $a+\frac{b}{c}=\frac{a+b}{c}$, $a+\frac{b}{c}=\frac{a\cdot b}{c}$
2. Multiplication: $\frac{a}{c}\cdot\frac{b}{c}=\frac{a\cdot b}{c}$, $\frac{c}{a-b}\cdot\frac{d}{c}=\frac{c\cdot d}{a-b\cdot c}$, $\frac{a}{b}\cdot\frac{c}{b}=\frac{a+c}{b}$
3. Division:$\frac{a}{b}\div\frac{c}{d}=\frac{a\cdot c}{b\cdot d}$,$\frac{a}{b}\div\frac{c}{d}=\frac{a\div c}{b\cdot d}$
4. Cancellation of Fraction: $\frac{a+b}{a+c}=\frac{b}{c}$, $\frac{a+b}{a}=b$, $\frac{a\cdot c+b}{a}=b+c$, $\frac{a+bx}{c+dx}=\frac{a+b}{c+d}$
5. Changing between forms: $a\frac{b}{c}=\frac{a+b}{c}$, $a\frac{b}{c}=\frac{a\cdot d+b}{c}$
6. Special cases with 0: $\frac{b}{c}-\frac{a}{c}=\frac{b-a}{0}$, $\frac{a}{a}=0$
7. Special cases with 1 or (-1): $\frac{a-b}{b-a}=1$, $-\frac{a}{-a}=-1$
8. Binomial forms: $\frac{a-b}{a^2-b^2}=a-b$, $(a-b)^2=a^2-b^2$, $(a+b)^2=a^2+b^2$

These groups are result of several design decisions. For instance, both groups no.1, addition, and no.2, multiplication, address multiplication *and* addition: addition can be confused with multiplication or vica versa, so a dialog author has to decide where a certain error fits better.

The decision for the case study appears in the implementation of the error-pattern in *ISAC*' mathematics engine. The latter is implemented in SML [MTHM97] (as is the underlying TP Isabelle); the implementation of group no.1, addition, in SML is as follows:

```
01 val errpats =
02   [("add-fractions",
03      [parse_patt thy "(?a / ?c + ?b / ?c) = (?a + ?b)/(?c + ?c)",
04       parse_patt thy "(?a / ?b + ?c / ?d) = (?a + ?c)/(?b + ?d)",
05       parse_patt thy "(?a / ?b + ?c / ?d) = (?a * ?c)/(?b * ?d)",

06       parse_patt thy "?a + (?b /?c)= (?a + ?b)/ ?c",
07       parse_patt thy "?a + (?b /?c)= (?a * ?b)/ ?c"],

08      [@{thm rat_add1}, @{thm rat_add2}, @{thm rat_add3}])]: errpat list;
```

According to Def.4 an error-pattern is a triple: The first element is the identifier *id*, above `"add-fractions"`; the second element are a list of *patterns p*, above `[parse_patt thy "(?a / ?c + ?b / ?c) = ..., ...]` (`[]` indicate lists); the third element is a list of rewrite-rules, above `[@thm rat_add1, ...]`. The latter is syntax of Isabelle, addressing the theorem with identifier `rat_add1`. The `?` above indicate a kind of variable which can take other values; this kind of variable allows to detect error-patterns.

In Tab.1 on p.185 the error is detected according to Def.5: the formula $\frac{5\cdot x}{4\cdot y}+\frac{3\cdot x}{4\cdot y}$ on *line 00* in Tab.1 matches the left-hand-side (*lhs*) of the pattern in line `03` above. The match is possible because of the

?variables: these are mapped to respective values $?a \to 5 \cdot x, ?c \to 4 \cdot y, ?b \to 3 \cdot x$. With this map the right-hand-side (*rhs*) of the pattern in line `03` above is instantiated to $\frac{5 \cdot x + 3 \cdot x}{4 \cdot y + 4 \cdot y}$; and this *rhs* equals the incorrect input $\frac{8 \cdot x}{8 \cdot y}$ on *line 01* in Tab.1 modulo a simplifier according to Def.5 — even for input of $\frac{x}{y}$ the error-pattern would be detected, see p.180.

The detection of error-patterns is costly in resources: the $\exists$ in Def.5 enforces to check *all* patterns for an input formula, if this formula is *not* related to any error-pattern. All eight groups of error-patterns listed on p.186 would comprise several hundred patterns.

Rewrite-rules are brought into the game as third element in an error-pattern, for instance in line `08` above: We want the student to do a correct step; however, a step is only correct if we have a rewrite-rule for it. According to Def.4 rewrite-rules in a error-pattern have a left-hand-side (*lhs*) matching the *lhs* of some patterns. Via the matching patterns an appropriate rewrite-rule is identified; for instance, among the three rewrite-rules in error-pattern “add-fractions” there are two matching the pattern in line `03` above, *rat_add1, rat_add2*:

$$\begin{aligned} rat\_add1 &:: \frac{?a}{?c} + \frac{?b}{?c} = \frac{?a + ?b}{?c} \\ rat\_add2 &:: \frac{?a}{?c} + \frac{?b}{?d} = \frac{?a \cdot ?d + ?b \cdot ?c}{?c \cdot ?d} \\ rat\_add3 &:: ?a + \frac{?b}{?c} = \frac{?a \cdot ?c + ?b}{?c} \end{aligned}$$

Fill-patterns are associated with rewrite-rules addressed in error-patterns, see Def.6. Fill-patterns contain *fill-in-patterns* with space-holders. These are left blank when instantiated: for instance, in Tab.1 line 00 the formula $\frac{5 \cdot x}{4 \cdot y} + \frac{3 \cdot x}{4 \cdot y}$ matches line `03` of the error-pattern, as already mentioned. From the error-pattern's rewrite-rules *rat_add1* is selected, and *rat_add1*'s fill-patterns are implemented in SML as follows:

```
val fillpat =
   [("fill-addition-first1",
     parse_patt @{theory Rational} "(?a / ?c + ?b / ?c) = ( _ + _ ) / (  _  )",
     ["add-fractions"]),
    ("fill-addition-first2",
     parse_patt @{theory Rational} "(?a / ?c + ?b / ?d) = ( _ + _ ) / (?c) ",
     ["add-fractions"]),
    ("fill-addition-first3",
     parse_patt @{theory Rational} "(?a / ?c + ?b / ?d) = (?a +  _) / (?c ",
     ["add-fractions"]),
    ...
   ]: fillpat;
```

Here exactly those fill-in-patterns are shown, which generated the lines `03..05` in Tab.1; the place-holders are represented as _. It is imaginable to create the place-holders automatically. It is noted, that the third element in a fill-pattern's triple, e.g. `"add-fractions"` above, are provided for future services for the dialogue not yet implemented.

3.5 Implementation of Dialogue-Rules

Error-patterns and fill-patterns are implemented in $\mathcal{ISAC}$'s *mathematics engine (ME)* which is appropriate because they involve rewriting, a technique from Computer Mathematics. Dialogue-rules are implemented in the Java-based part of $\mathcal{ISAC}$, where the dialogue component resides, mediating between the student working on the *Worksheet (WS)* and the ME, so it is called *WorksheetDialog (WD)* [6]. Fig.1 on p.185 shows a WS. The WD is, like all other dialogues, located on the same server for all students probably working remotely and connected to the server via Internet. Fig.2 shows all the relevant components.

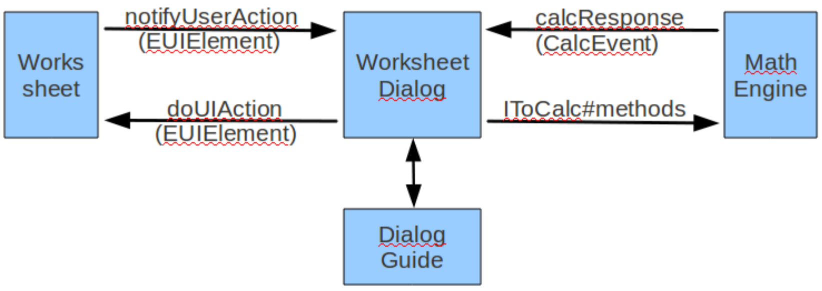


Figure 2: Architecture of $\mathcal{ISAC}$'s Dialogue

The behaviour of $\mathcal{ISAC}$'s dialog is determined by *dialogue-rules* interpreted by Drools Expert [Ama12], a knowledge-based expert system. These rules are triggered by events addressing the WD on top from left and right, and the rules in the WD can determine actions directed towards left (WS) and right (ME). The features of the arrows in Fig.2 are in more detail:

1. *notifyUserAction* notifies the WorksheetDialog about an action of the user on the Worksheet by an **EUIElement**. For instance, in `rule 2` the trigger is `EUIElement == UI_SOLVE_HELP_ENTERING_FORMULA` indicating that $\boxed{\text{HELP}}$ has been pushed.

2. *doUIAction* notifies the Worksheet about what to display. This methods uses the same **EUIElement** as (1.) compound with appropriate data (i.e. the actions between WS (learner) and WD (the "system") are symmetrical). The rules in the running example do not involve such notifications, because fill-in-forms are passed to the WS by CalcChanged without intervention of the WD.

3. *calcResponse* notifies the WorksheetDialog about the result of the MathEngine for the last request for calculation from the WorksheetDialog by a **CalcEvent**. This can be of 2 kinds:

   (a) **CalcChanged** notifies about a successful step of calculation. This case occurs when an error-situation is quit by a correct input which "changes the calculation" by adding a new line. Since the dialogue cannot interpret formulas, CalcChanged passes formulas to the

[6]The Worksheet on the front-end is accompanied by further windows accepting user-input; these windows have their own dialogues and are out of scope of this paper.

WS bypassing the dialogue. Thus the rules concerning error-patterns are not triggered by CalcChanged, but only by CalcMessage, see `rule 1`.

(b) **CalcMessage** notifies about a failure of the MathEngine trying to do the last requested step of calculation. For instance, in `rule 1` on line `03` an error-pattern is reported together with an `errorID`, in the running example *add-fractions*.

4. **IToCalc#methods** request services from the MathEngine, which is addressed via interface IToCalc, an abstraction of the calculation on the Worksheet with respective positions, formulas and tactics/rules. For instance, `rule 2` sets the action `ME#requestFillformula(err_patt, fill_patt)`.

5. Services from *DialogGuide (DG)* are access to hint-pages, update of counters etc. For instance, `rule 1` contains `DG#showHintPage (err_patt_)`.

The executable format of Drools' rules involves distracting technicalities, so the rules below are written in a pseudo-code which is both, comprehensible for a non-programmer (like a dialog-author) and succinct enough for a programmer translating them to executable code [7].

```
01 rule "1: show a hint-page if an error-pattern is detected"
02   when
03     CalcMessage == "error-pattern#errorID#"
04   then
05     error_pattern_ = CalcMessage.getErrorPattern()
06     error_patterns_.add(error_pattern_)
07     DG#showHintPage (err_patt_)
08     DG#EP_counter (err_patt_) ++
09     help_counter_ = 0
10     WS#addHelpButton()
11     WS#removeNextAndAutoButtonForWorksheet()
12   end
```

Besides giving hints it is also important to involve students actively in the learning process, and let them think. We don't let them just request the next correct step or formula (see line `11` above), but make them struggle a bit. However, line `10` adds a $\boxed{\text{HELP}}$ button (see Fig.1) for requesting fill-forms.

```
01 rule "2: from ME request fill-patterns"
02   when
03     EUIElement == UI_SOLVE_HELP_ENTERING_FORMULA
04   then
05     fill_patts_ = ME#findFillpatterns(err_patt_)
06     fill_patt = DG#selectFillPattern(fill_patts_, help_counter_)
07     ME#requestFillformula(err_patt, fill_patt)
08     help_counter_ ++
09   end
```

`rule 3` repeats the request for a fill-form. The DG knows which one to select from the `help_counter_`, see line `07` below. Finally, `rule 4` proposes the tactics, i.e. the rule, leading to the correct next formula.

[7] The $\mathcal{ISAC}$-project expects to develop a dialogue-language which copes with the need of both, authors and programmers, within one single format.

```
rule "3: request other fill-form"
  when
    CalcMessage == "fill-form incorrect"
    && help_counter < length(fill_patts_)
  then
    fill_patt =
      DG#selectFillPattern(fill_patts_,
                           help_counter_)
    ME#requestFillformula(err_patt, fill_patt)
    help_counter ++
  end
```

```
rule "4: fill-forms did not help,show rule"
 when
    ((CalcMessage)calc_event).getText()
    == "fill-form incorrect"
    && help_counter >= fill_pats_no
  then
    calc_tree.fetchProposedTactic();
  end
```

Given the 37 dialogue-rules handling interactions with $\boxed{\text{NEXT}}$, $\boxed{\text{AUTO}}$, input of a formula, etc. — the above 4 dialogue-rules are sufficient to extend ISAC's dialogue behaviour in the decisive way described in this paper.

The dynamic behaviour of the four rules is shown in Fig.3 . The reader may note, that the four rules are neither specific for addition, nor fractions — they are general such that they handle all domains of mathematics.

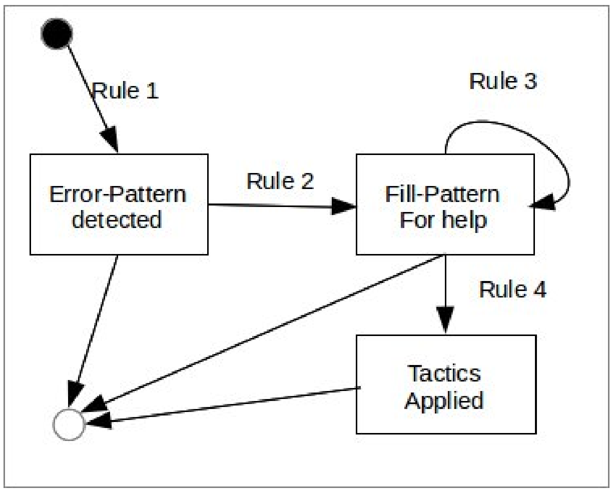


Figure 3: State-diagram of the dialogue-rules

4 Experiences from Prototyping and Future Work

The above case study extended the already general next-step-guidance with an equally general machinery handling error-patterns; this has immediate benefits and raises lots of opportunities for further research and development.

Extension of machinery for user-guidance is generalised as follows:

1. A few additional lines of code (see p.186) provide automatic detection of an error-pattern wherever a respective error occurs. Such additions are generally applicable: they refer to a particular

set of theorems, and theorems are the means to justify any step of calculation in any topic of mathematics. *So the extended machinery generalises to all kinds of mathematics.*

2. An additional error-pattern, for instance dealing with canceling fractions, carries over to any re-use of the respective theorems in any other application. For instance, error-patterns implemented for calculating with rationals, carry over to *all* calculations with rationals, because the respective program code serves both (due to Lucas-Interpretation): (1) the creation of next-step-guidance and (2) the re-use in programs on more advanced problems. *So the machinery generalises over all re-uses in the course of systematic build-up of mathematics without additional code for user-guidance.*

3. Four dialog rules extended user-guidance by three adaptive system reactions: (1) show a hint-page (and probably follow the links on this page), (2) suggest an incomplete next step (and check respective input) and (3) propose the rule to be applied. These four rules can easily be changed for other kinds of adaptive user-guidance; and, of course, the small number of rules indicates best prerequisites for well-structured extension by many, many other rules. *So the dialog machinery generalises and scales to high complexity in accordance to future requests.*

Counters are the first point raising questions for further research and development. Counters occur in all of the rules in §3.5, see `EP_counter` and `help_counter_`. However, nothing has been said about these questions:

Which value does a counter start with? Always zero? If not zero, do counters take previous sessions into account? What is the correlation between `EP_counter` and `help_counter_`? If counters are stored for many sessions, what is relevant? Their average? Do these counters provide information useful for user-guidance and for assessment?

These questions and others cannot be bypassed on the way to a **user-model**.

Recording user-actions is already implemented in *ISAC*: The `EUIElements` in Fig.2 represent steps which promote the construction of a solution within a logical context and a user not aware of the context will fail to do such a step — so these steps might be called "high-level", closely related to comprehensive mathematical activities.

So a **history** of user-actions is ready to be used for statistical analysis — but can the abundance of data be seriously related to cognitive processes involved in mathematics [Cla05], to problem-solving expertise? What structures and correlations can be expected from statistical analysis over large populations? What about comparison of different populations?

Hint-pages can pop up due to adaptive user-guidance like in Fig.1 on p.185 — but there is HTML-technology available with exciting new multimedia features (videos, pictures, diagrams, graphics...etc.). These shall connect individual thinking with support for multiple representations, e.g. for fraction as mentioned in §3.1. This aspect is important because according to [RJSA01] multiple representation is key in the abstraction process.

Blended learning environments of a new kind can be assembled from components envisaged above. This opens new possibilities for complex learning activities, for support for learning in a social environment, for involving the teacher (students are influenced of the teachers external representation in both procedural and conceptual knowledge([Bil00]), and also the individual way of thinking – people have individual strategies in solving mathematical problems and individual ways of learning [RJSA01].

5 Conclusion

This paper reports the outcomes of a first cooperation between Cognitive Science and Computer Mathematics in the $\mathcal{ISAC}$-project. The outcomes are the following:

1. The case study performed does *not* claim to have immediately improved the benefits for students from adaptive user guidance (thus the study does *not* feel obliged to confirm scientific progress by evaluation of advances in some learning scenario).
2. Rather, the goal stated in §3.2 addresses improvement of existing *machinery* which evidently advances generality of technology for user-guidance: "Generalise automated generation of user-guidance with concurrent detection of errors recurring during systematic build-up of mathematics". The accomplishment of that generalisation is described in detail on p.190 and exceeds the state-of-the-art represented by the most advanced mathematics tutors as mentioned in §3.2.
3. Cooperation between Computer Mathematics (CM) and Cognitive Science (CS) leads to mutual benefits: CM gets requirements from CS how to improve dialog machinery, and CS gets realisable ideas for improving dialog-guidance (benefits which appear particularly fruitful with respect to $\mathcal{ISAC}$'s general machinery).
4. Experience gained in the case study described in this paper suggest further ideas for research and development in interdisciplinary cooperation in §4.
5. Last not least a proof for improved reliability of a specific TP-based service in Computer Mathematics in §2.3, triggered as a requirement by Cognitive Science.

Finally, with current technological and theoretical background it would be really hard to understand and find out the thoughts of the learner. The goal is to help/guide/support in the learning process, reflect on errors, give information if requested, and encourage learners to solve problems on their own. The technology used in this case study has the potential to support teaching mathematics — also on individual level— what would be hard due to missing resources and time in a classroom.

So the authors look forward to inspire their respective communities, Cognitive Science and Computer Mathematics, about the mutual benefits from future interdisciplinary cooperation for an upcoming generation of TP-based educational mathematics assistants.

References

[Ama12] Lucas Amador. *Drools developers cookbook: over 40 recipes for creating a robust business rules implementation by using JBoss Drools rules.* Birmingham, U.K. : Packt, 2012.

[And08] John R. Anderson. Intelligent tutoring and high school mathematics. Technical Report 20, Carnegie Mellon University, Department of Psychology, 2008. http://repository.cmu.edu/psychology/20.

[Bee92] Michael J. Beeson. Mathpert: Computer support for learning algebra, trig and calculus. In A. Voronkov, editor, *Proceedings of the International Conference on Logic Programming and Automated Reasoning (LPAR'92)*, volume 624 of *LNAI*, pages 454–456, St. Petersburg, Russia, July 1992. Springer Verlag.

[Bil00] C. Bills. The influence of teachers representation on pupils mental representation. *Research in Mathematics Education*, 2, Issue 1:45–60, 2000.

[BN98] Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

[Bob96] J. Bobis. Visualisation and the development of number sense with kindergarten children. In J. Mulligan and M. Mitchelmore, editors, *Children's Number Learning*, page 1733. Adelaide: AAMT & MERGA, 1996.

[Bro78] R. R: Brown, J. S. Burton. Diagnostic models for procedural bugs in basic mathematical skills*. *Cognitive Science*, 2:155–192, 1978.

[Cla05] B. Clancey. Modeling the perceptual component of conceptual learning–a coordination perspective. In P. Grdenfors and P. Johansson, editors, *Cognition, Education and Communication Technology*, pages 109–146. Mahwah, NJ: Lawrence Erlbaum Associates, 2005.

[Col06] A. Collins. Cognitive apprenticeship. In Keith R. Sawyer, editor, *Cambridge Handbook of the Learning Sciences*. MIT Press, Washington University, St Louis, 2006. ISBN:9780521845540.

[Dre90] Tommy Dreyfus. Advanced mathematical thinking. In *Mathematics and Cognition, A Research Synthesis by the International Group for the Psychology of Mathematics Education*, 1990.

[Gel78] C. Gelman, R. & Gallistel. *The Child's Understanding of Number.* Cambridge, MA: Harvard University Press., 1978.

[Hem90] R. Hembree. The nature, effects, and relief of mathematics anxiety. *Journal for Research in Mathematics Education*, 21(1):33–46, 1990.

[HKN10] Florian Haftmann, Cezary Kaliszyk, and Walther Neuper. CTP-based programming languages ? considerations about an experimental design. *ACM Communications in Computer Algebra*, 44(1/2):27–41, March/June 2010. http://www.ist.tugraz.at/projects/isac/publ/plmms-10.pdf.

[Kie90] Carolyn Kieran. Cognitive processes involved in learning school algebra. In *Mathematics and Cognition, A Research Synthesis by the International Group for the Psychology of Mathematics Education*, 1990.

[MS04] Erica Melis and Jörg Siekmann. An intelligent tutoring system for mathematics. In L. Rutkowski, J. Siekmann, R. Tadeusiewicz, and L.A. Zadeh, editors, *Seventh International Conference Artificial Intelligence and Soft Computing (ICAISC)*, number 3070, in LNAI, page 91101. Springer-Verlag, 2004.

[MTHM97] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. The MIT Press, Cambridge, London, 1997.

[Neu] Walther Neuper. On the emergence of TP-based educational math assistants. published in this issue.

[Neu10] Walther Neuper. Common grounds for modelling mathematics in educational software. *Int. Journal for Technology in Mathematics Education*, 17(3), 2010.

[Neu12] Walther Neuper. Automated generation of user guidance by combining computation and deduction. In Pedro Quaresma, editor, *THedu'11: CTP-compontents for educational software*. EPTCS, 2012. 82-101.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[Nun04] R. Nunez. Embodied cognition and the nature of mathematics: Language, gesture, and abstraction. In *Proceeding of the 26th Annual Conference of the Cognitive Science Society (pp.36-37)*, 2004.

[Pay90] H.R. Payne, R. J. Squibb. Algebra mal-rules and cognitive accounts of error. *Cognitive Science*, 14:445–481, 1990.

[Plo81] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, CS, Aarhus University, Sep 1981.

[RAKC07] Steven Ritter, John R. Anderson, Kenneth R. Koedinger, and Albert Corbett. Cognitive tutor: Applied reserach in mathematics education. *Psychonomic Bulletin & Review*, 14 (2):249–255, 2007.

[RJSA01] B. Rittle-Johnson, R.S. Siegler, and M.W. Alibali. Developing conceptual understanding and procedural skill in mathematics: An iterative process. *Developing conceptual understanding and procedural skill in mathematics: An iterative process.*, 93:346–362, 2001.

[RMD$^+$05] R.Prank, M.Issakova, D.Lepp, E.T onisson, and V.Vaiksaar. T-algebra - interactive learning environment for expression manipulation. In *7th International Conference on Technology in Mathematics Teaching*, volume 1, pages 26–29, Bristol, UK, July 2005.

[Roč] Jan Ročnik. Interactive course material by TP-based programming. A case study. published in this issue.