Automated feedback for intelligent tutoring systems

Bastiaan Heeren
Informatica studiedag, 6-9-2019, Utrecht
Template for the hypothesis testing strategy

hypothesisStrategy :: LabeledStrategy ComponentSet

hypothesisStrategy = label "Hypothesis testing" $
  [ addHypothesesRule, addH0FromHARule, addH0FromHAEqualSignRule, addHARule,
    addHypothesesChiSquaredRule, addAlphaRule, determineSided, chooseTTestRule,
    chooseTTestTwoRule, chooseTTestPairedRule, chooseZTestRule,
    chooseRPearsonRule, chooseAnovaRule, chooseChiSquaredRule
  ] ++ sampleStatistics$

check (\cs -> all (derived cs `contains` [NullHypothesis, AlternativeHypothesis]))

label "Computation" (whileNotReady $ $
  (check (\cs -> derived cs `doesNotContain` TestValue)).$
  (addTestFormulaRule .|. choice sampleStatistics))$

  (check allowCriticalRoute .*. choice
   [ addTestValueRule, addRejectionRule, lookupZValueRule, lookupTValueRule, lookupRValueRule, lookupFValueRule, lookupChiValueRule ])

  (check allowPValueRoute .*. choice
   [ computePValueZTest, computePValueTTest ])

).$

check (\cs -> derived cs `contains` TestValue &&
  derived cs `contains` Critical || derived cs `contains` PValue)$

label "Conclusion" ($
  whileNotReady (criticalConclusionRule .|. addConclusionPValueRule)$
).$

(hypothesesConclusionCriticalRule .|. hypothesesConclusionPValueRule))$

where

sampleStatistics =
  [ addNRule, addAverageRule, addVarianceRule, addStandardDeviationRule
\[4(x - 4) = 5(2x + 1)\]
\[4(x - 4) = 5(2x + 1)\]

\[4x - 16 = 5(2x + 1)\]
\[ 4(x - 4) = 5(2x + 1) \]
\[ 4x - 16 = 5(2x + 1) \]
\[ 4x - 16 = 10x + 1 \]

fout bij haakjes
wegwerken, vermenigvuldig
beide termen tussen de
haakjes
\[
4(x - 4) = 5(2x + 1) \\
4x - 16 = 5(2x + 1) \\
4x - 16 = 10x + 1 \\
\]

Tip:
haakjes uitwerken
\[ 4(x - 4) = 5(2x + 1) \]
\[ 4x - 16 = 5(2x + 1) \]
\[ 4x - 16 = 10x + 1 \]

Tip:
haakjes uitwerken

\[ 4x - 16 = 5(2x + 1) \]
wordt dan:
\[ 4x - 16 = 10x + 5 \]
\[4(x - 4) = 5(2x + 1)\]
\[4x - 16 = 5(2x + 1)\]
\[4x - 16 = 10x + 5\]

haakjes uitwerken

correct (staandaard strategie)
\[ 4x - 16 = 10x + 5 \]

\[ -16 = 6x + 5 \]

\[ -21 = 6x \]

\[ -\frac{7}{2} = x \]

\[ x = -\frac{7}{2} \]

Van beide kanten 4 \cdot x aftrekken

Van beide kanten 5 aftrekken

Beide kanten delen door 6

De vergelijking omdraaien

Correct opgelost
Overview of this talk

1. Intelligent Tutoring Systems (ITS)
   - Domain reasoners
   - Feedback services
2. Expert domain knowledge
   - Problem-solving procedures
   - Granularity (step-size)
3. Examples of domain reasoners

Motivation:
1. Simplify construction of ITSs (which are complex software systems)
2. Represent expert domain knowledge explicitly (for better feedback)
3. Apply approach to a wide range of problem domains
Research team

Many more scientists collaborate

Started around 2006

>20 BSc students

>20 MSc students

13,269 SVN commits, by 52 authors

Bastiaan Heeren
Open University of the Netherlands & Utrecht University
Bastiaan is the core designer and developer of the ideas software.

Johan Jeuring
Utrecht University & Open University of the Netherlands
Johan started with the ideas project more than a decade ago. He is involved in many of the subprojects.

Josje Lodder
Open University of the Netherlands
For her PhD, Josje works on several tutors related to logic.

Hieke Kouning
Windesheim University of Applied Sciences & Open University of the Netherlands
For her PhD, Hieke works on tutors for (imperative) programming.

Alex Gerdes
Gothenburg University
Alex is the main architect of the functional programming tutor Ask-Ella.

Alejandro Serrano Menas
Utrecht University
Alejandro works on Ask-Ella.
Part 1:

Intelligent Tutoring Systems (ITS)
Inner and outer loops (VanLehn 2006)

The Behavior of Tutoring Systems

Kurt VanLehn, LRDC, University of Pittsburgh, Pittsburgh, PA, USA
VanLehn@pitt.edu

Abstract. Tutoring systems are described as having two loops. The outer loop executes once for each task, where a task usually consists of solving a complex, multi-step problem. The inner loop executes once for each step taken by the student in the solution of a task. The inner loop can give feedback and hints on each step. The inner loop can also assess the student’s evolving competence and update a student model, which is used by the outer loop to select a next task that is appropriate for the student. For those who know little about tutoring systems, this description is meant as a demystifying introduction. For tutoring system experts, this description illustrates that although tutoring systems differ widely in their task domains, user interfaces, software structures, knowledge bases, etc., their behaviors are in fact quite similar.

Keywords. Intelligent tutoring systems, knowledge components, learning events, tutoring

- **Outer loop**: solving one task after another
- **Inner loop**: the steps for solving one complex, multi-step problem
Four component ITS architecture

- Classical structure of an ITS (with four components)
- In practice, often one monolithic system
Domain reasoner

A domain reasoner is the part of the system that can ‘reason about the problems’:
- the objects in a domain (e.g. expressions, equations)
- how these objects can be manipulated
- how to guide manipulation to reach a certain goal

- For math, computer algebra systems (CAS) can do part of the job:
  - they are great in evaluating expressions, but
  - built-in equality can be very subtle
  - not designed for providing feedback
Providing feedback

Narciss (2008) distinguishes the following feedback types:

- Knowledge of performance
  - E.g. percentage of correctly solved tasks
- Knowledge of result/response (KR)
  - Correct/incorrect
- Knowledge of the correct response (KCR)
  - Provides the correct answer
- Elaborated feedback
  - Additional information besides KR and KCR
- Answer-until-correct and Multiple-try feedback
Feedback services

- A domain reasoner provides feedback services:
  - Intuitively, just request-response communication
  - Services are derived from the feedback types
  - Services for the inner loop and for the outer loop

Examples of services:

- Am I finished?
- Give me a next-step hint
- Give me a worked-out solution
- Is my step correct (step diagnosis)?
  - If yes: does the step bring me closer to a solution?
  - If no: is it a common mistake?
Part 2:

Expert domain knowledge
Ideas framework

Generic framework for constructing domain reasoners

- Developed in Haskell
- Size: 12,397 LOC
- Open source
- Independent of problem domain
- http://ideas.cs.uu.nl/tutorial/

Interactive Domain-specific Exercise Assistants
Interactive explorer for domain reasoners

Exercise algebra.equations.linear
solve a linear equation

Derivation

\[ \frac{3}{4}x - (x-1) = 3 + 2[1/2](x-1) \]

\[ \Rightarrow \text{algebra.equations.linear.remove-div.factor=4} \]

\[ -x + 4 = 12 + 10(x-1) \]

\[ \Rightarrow \text{algebra.equations.linear.distr-times} \]

\[ -x + 4 = 12 + 10x - 10 \]

\[ \Rightarrow \text{algebra.equations.linear.merge} \]

\[ -x + 4 = 2 + 10x \]

\[ \Rightarrow \text{algebra.equations.linear.var-left.term=10*x} \]

\[ -11x + 4 = 2 \]

\[ \Rightarrow \text{algebra.equations.coverup.onevar.plus} \]

\[ -11x = -2 \]

\[ \Rightarrow \text{algebra.equations.coverup.times} \]

\[ x = 2/11 \]
**Rules**

- **Rules** specify the steps (manipulations) that are allowed
  - rewriting steps
  - refinement steps

Distributivity rule: $\forall abc. a(b + c) \rightarrow ab + ac$

Example: $5(x + 2) \rightarrow 5x + 10$

Preferably specified as a **rewrite rule** (for further analysis):

```
\texttt{distr = rule "distr" $ \backslash a \ b \ c \rightarrow a*(b+c) :~>\ a*b + a*c}$
```

**Rules** are used for:
- recognizing steps
- suggesting possible next steps
Implementing rewrite rules

distr :: Rule Expr
distr = rule “distr” $ \ a \ b \ c \to \ a*\(b+c) \ :~> \ a*b + a*c

- Meta-variables are introduced by a lambda abstraction?

Type-index datatypes approach supports:
- Knuth-Bendix completion (analysis)
- AC-rewriting
- Rule inversion
- Automated testing
- Documentation (pretty-printing)
Problem-solving procedures describe sequences of rule applications that solve a particular task.

Example procedure for adding two fractions:

1. find the lowest common denominator (LCD)
2. convert fractions to LCD as denominator
3. add the resulting fractions
4. simplify the result

Problem-solving procedures are used for:

- recognizing the strategy
- detecting detours
- providing next-step hints
- providing worked-out examples
Problem-solving procedures

We have developed a domain-specific language for specifying procedures: sequence, choice, repeat, try, prefer, somewhere, etc.

\( \text{FindLCD} ; \text{many (somewhere Convert)} ; \text{Add ; try Simplify} \)

Resulting in:

\[
\frac{1}{2} + \frac{4}{5} \quad \text{FindLCD} \quad \frac{1}{2} + \frac{4}{5} \quad \text{Convert} \quad \frac{5}{10} + \frac{4}{5} \quad \text{Convert} \quad \frac{5}{10} + \frac{8}{10} \quad \text{Add} \quad \frac{13}{10} \quad \text{Simplify} \quad 1 \frac{3}{10}
\]
Theoretical foundations

Problem-solving procedures:

- are inspired by context-free grammars
- have been formalized by a trace-based semantics (CSP)
- allow new composition operators (interleaving, topological sorts)
- enable various tree traversal strategies (topdown, outermost)

\[
\begin{align*}
\mathcal{T}(s; t) &= \{ x \mid x \in \mathcal{T}(s), \checkmark \notin x \} \cup \{ xy \mid x\checkmark \in \mathcal{T}(s), y \in \mathcal{T}(t) \} \quad \text{(sequence)} \\
\mathcal{T}(s \mid t) &= \mathcal{T}(s) \cup \mathcal{T}(t) \quad \text{(choice)} \\
\mathcal{T}(\mu x.f(x)) &= \mathcal{T}(f(\mu x.f(x))) \quad \text{(fixed point)} \\
\mathcal{T}(r) &= \{ \epsilon, r, r\checkmark \} \quad \text{(rule)} \\
\mathcal{T}(\text{succeed}) &= \{ \epsilon, \checkmark \} \quad \text{(success)} \\
\mathcal{T}(\text{fail}) &= \{ \epsilon \} \quad \text{(failure)}
\end{align*}
\]
Normal forms (equivalence classes)

Normal forms define classes of expressions that are treated the same, and select one canonical element for such a class.

Example: $10 + 5x \approx 5x + 10 \approx 5x + 5 \cdot 2$

- In math: associativity, commutativity, calculations, simplifications, etc.
- Used for relations such as equal, equivalent, similar, indistinguishable
- The granularity (step size) of a task is often left implicit

Normal forms are used for:
- recognizing steps
- rewriting atypical expressions, e.g. $4 + (-5)$
- deciding whether finished or not
Buggy rules describe common mistakes and enable specialized feedback messages when detected.

Buggy distribution: \( \forall abc \ . \ a(b + c) \rightarrow ab + c \)

Example: \( 5(x + 3) \rightarrow 5x + 3 \)

Sign mistake: \( 5x = 2x + 3 \rightarrow 7x = 3 \)

- Buggy rules are often associated with a sound rule

Buggy rules are used for:
- detecting common mistakes
Constraints have a relevance condition and a satisfaction condition: on violation, a special message can be reported.

Example: if the equation is linear (relevance), then the equation’s right-hand side should not contain x (satisfaction).

Constraint message: the equation is not yet solved.

- Based on theory of learning from performance errors (Ohlsson 1992)

Constraints are used for:
- checking properties or attributes
- reporting violations
Feedback on the structure of hypothesis tests

Exercise 7

How would you react if the grade you received for an exam is much lower than you had expected? Research suggests that most students think they can handle such situations better than their peers, but some students think their coping is worse than that of their peers.

In this study, participants were asked to read a scenario of a negative event and indicate how this event would influence their well-being (-5: worsen much, +5: improve much). Next, they were asked to imagine the same event from the perspective of a peer. The difference between both judgments was noted.

Suppose that for the sample of $n = 25$ students the mean difference score was $M_D = 1.28$ points (own judgment minus judgment peer) with standard deviation $SD = 1.50$.

Round off answers to two decimals, if necessary.

Formulas

Based on these data, can you conclude that there is a significant difference between the own judgments and judgments of peers? Use a test with $\alpha = 0.05$.

1. State null hypothesis and alternative hypothesis

   $H_0: \mu_D = 0$

   $H_1: \mu_D \neq 0$

2. Determine whether the test is left sided, right sided or two sided

   The test is [two sided]

3. Find critical value

   $t_{crit} = 2.064$

4. Determine rejection region

   $t < t_{crit}$

Action: Choose

Does the sign you use in the rejection region match with the direction of the alternative hypothesis?

Conclusion: There [Choose] a significant difference between the judgments of one's own reaction and the reaction of a peer.
Feedback on the structure of hypothesis tests

The tutor’s diagnose feedback service combines several knowledge components:

- **Constraints violated?**
  - Yes: buggy rule?
    - Yes: Common mistake with buggy rule
    - No: Violated constraint
  - No: similar?
    - Yes: expected by strategy?
      - Yes: Correct step, but expert strategy
      - No: Small rewrite step, not recognized
    - No: discover rule?
      - Yes: Correct step, but detour from strategy
      - No: Correct rewrite step, but unknown
Part 3:

Examples of domain reasoners
Advise-Me: project goal

- Automatic Diagnostics with Intermediate Steps in Mathematics Education
- Assessment of free-text input for math story problems:
  - Set up algebraic expressions and simplify them
  - Set up equations and inequalities and solve them

- Task design resources:
  - Pépite materials (Paris)
  - CITO (Arnhem)
  - Freudenthal Institute (Utrecht)
  - USAAR (Saarbrucken)

This project has received funding from the European Union’s ERASMUS+ Programme, Strategic Partnerships for school education for the development of innovation, under grant agreement number 2016-1-NL01-KA201-023022.
Domain reasoner for axiomatic proofs

Complete the proof in two directions

Fill in the template, then the rule is applied automatically
Domain reasoner for functional programming

Holes for unfinished parts in the program

Hint sequences
Tutoring system to learn code refactoring

- Tool based on rules extracted from input by 30 experienced teachers

Description: The sumValues method adds up all numbers from the array parameter, or only the positive numbers if the positivesOnly boolean parameter is set to true.

The solution is already correct, but can you improve this program?
Domain reasoners for communication skills

Marc, I want to discuss something with you, but first you must promise to keep quiet.

Marc, I have good news for you: I have been in touch with a sleep clinic and they can help you straight away.

Unfortunately, I've got bad news for you: I have to refer you to another therapist.

How are you? Did you sleep well recently?
OU Master theses about domain reasoners

Practice with the evaluation of a Haskell Expression

<table>
<thead>
<tr>
<th>Haskell Expression</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sum ([3,7] ++ [5])</code></td>
<td><code>Select</code></td>
</tr>
<tr>
<td>Options</td>
<td></td>
</tr>
<tr>
<td>□ Outermost evaluation strategy</td>
<td>□ Innermost evaluation strategy</td>
</tr>
<tr>
<td>Next step</td>
<td></td>
</tr>
<tr>
<td>Diagnose</td>
<td>fold (+) 0 (3 : ([7] ++ [5]))</td>
</tr>
<tr>
<td>Tests</td>
<td></td>
</tr>
<tr>
<td>Show number of steps left</td>
<td>Show all rules that can be applied</td>
</tr>
<tr>
<td>Show next rule</td>
<td></td>
</tr>
<tr>
<td>Show next step</td>
<td></td>
</tr>
<tr>
<td>Do next step</td>
<td></td>
</tr>
</tbody>
</table>

Derivation

- `sum ([3,7] ++ [5])`
- `fold (+) 0 (3 : ([7] ++ [5]))`
- `fold (+) 0 (3 : ([7] ++ [5]))`
- `Apply the sum rule to sum up all elements of a list`
- `Apply the append rule to concatenate two lists`

Figure 1: Constructing a square using circles and lines (screenshot from GeoGebra – a mathematics tool that allows drawing of geometric structures).

Tim Olmer (2014, TFPIE), Hieke Keuning (2014, CSERC), Stéphane Thibaud (2017), Hugo Arends (2017, Koli Calling)
4. Rob Smit (in progress). A domain-specific language for generating feedback in Intelligent Tutoring Systems
5. Cor Zijlstra (in progress). Student interaction module – Architecture trade-offs for a logic student interaction module
Trends and challenges

- **Authoring** intelligent tutoring system
  - Literature reports 200-300 authoring hours for 1 hour of instruction
  - We believe software technology can help
- **Data-driven** intelligent tutoring system
  - Use AI techniques to generate feedback from collected data
  - Raises questions about the role of expert domain knowledge
- **Further adaptation and personalization**
  - Models for mastery learning (e.g. Bayesian knowledge tracing)
- **Designing tools for less-structured problem domains**
  - For example, domains of software design and learning languages
Take-home messages

1. Domain reasoners with feedback services simplify the construction of ITSs
   – Services result in loosely coupled, reusable software components
   – Services can be derived from popular feedback types

2. Represent expert domain knowledge explicitly (for better feedback)
   – Rules, problem-solving procedures, normal forms, buggy rules, constraints
   – The step-size of a task matters

3. The presented approach can be applied to a wide range of problem domains

Websites:
- http://ideas.cs.uu.nl/
- http://advise-me.ou.nl/

✉️ Bastiaan.Heeren@ou.nl