

Automated feedback for intelligent tutoring systems



Bastiaan Heeren

Informatica studiedag, 6-9-2019, Utrecht

Open Universiteit

www.ou.nl



```

13 -----
14 -- Template for the hypothesis testing strategy
15
16 hypothesisStrategy :: LabeledStrategy ComponentSet
17 hypothesisStrategy = label "Hypothesis testing" $
18   label "Preparation" (whileNotReady $ choice $
19     [ addHypothesesRule, addH0FromHARule, addH0FromHAEqualSignRule, addHARule
20     , addHypothesesChiSquaredRule
21     , addAlphaRule, determineSided, chooseTTestRule
22     , chooseTTestTwoRule, chooseTTestPairedRule, chooseZTestRule
23     , chooseRPearsonRule, chooseAnovaRule, chooseChiSquaredRule
24     ] ++ sampleStatistics)
25   *.*
26   check (\cs -> all (derived cs `contains`) [NullHypothesis, AlternativeHypothesis])
27   *.*
28   label "Computation" (whileNotReady $
29     (check (\cs -> derived cs `doesNotContain` TestValue) *.*
30     (addTestFormulaRule .|. choice sampleStatistics))
31     .|. (check allowCriticalRoute *.* choice
32     [ addTestValueRule, addRejectionRule
33     , lookupZValueRule, lookupTValueRule, lookupRValueRule, lookupFValueRule, lookupChiValueRule
34     ])
35     .|. (check allowPValueRoute *.* choice
36     [ computePValueZTest, computePValueTTest
37     ])
38   )
39   *.*
40   check (\cs -> derived cs `contains` TestValue &&
41     derived cs `contains` Critical || derived cs `contains` PValue)
42   *.*
43   label "Conclusion" (
44     whileNotReady (criticalConclusionRule .|. addConclusionPValueRule)
45     *.*
46     (hypothesesConclusionCriticalRule .|. hypothesesConclusionPValueRule))
47 where
48   sampleStatistics =
49     [ addNRrule, addAverageRule, addVarianceRule, addStandardDeviationRule

```

$\sqrt{\square}$ \square^{\square} \square^2 $\frac{\square}{\square}$ (\square) meer tip help stap losop \downarrow \uparrow

$$4(x - 4) = 5(2x + 1)$$

$\sqrt{\quad}$ \square^{\square} \square^2 $\frac{\square}{\square}$ (\square) meer tip help stap losop \downarrow \uparrow

$4(x - 4) = 5(2x + 1)$

$4x - 16 = 5(2x + 1)$

\square

$\sqrt{\square}$ \square^{\square} \square^2 $\frac{\square}{\square}$ (\square) meer tip help stap losop ↓ ↑

$4(x - 4) = 5(2x + 1)$

$4x - 16 = 5(2x + 1)$

$4x - 16 = 10x + 1$

X

fout bij haakjes
wegwerken, vermenigvuldig
beide termen tussen de
haakjes

$\sqrt{\quad}$ \square^{\square} \square^2 $\frac{\square}{\square}$ (\square) meer tip help stap losop ↓ ↑

$4(x - 4) = 5(2x + 1)$

$4x - 16 = 5(2x + 1)$

$4x - 16 = 10x + 1$

X

Tip: haakjes uitwerken

$\sqrt{\quad}$ \square^{\square} \square^2 $\frac{\square}{\square}$ (\square) meer tip help stap losop \downarrow \uparrow

$$4(x - 4) = 5(2x + 1)$$

$$4x - 16 = 5(2x + 1)$$

$$4x - 16 = 10x + 1$$



Tip: ✕
haakjes uitwerken

$$4x - 16 = 5(2x + 1)$$

wordt dan:

$$4x - 16 = 10x + 5$$

$\sqrt{\quad}$ \square^{\square} \square^2 $\frac{\square}{\square}$ (\square) meer tip help stap losop ↓ ↑

$4(x - 4) = 5(2x + 1)$

$4x - 16 = 5(2x + 1)$

$4x - 16 = 10x + 5$

✓

haakjes uitwerken

correct (standaard strategie) ✕

√ □ □² □ (□) meer tip help stap losop ↓ ↑

$4x - 16 = 10x + 5$

van beide kanten $4 \cdot x$ aftrekken

$-16 = 6x + 5$

van beide kanten 5 aftrekken

$-21 = 6x$

beide kanten delen door 6

$-\frac{7}{2} = x$

de vergelijking omdraaien

$x = -\frac{7}{2}$

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

Open Universiteit
www.ou.nl



Research team

Many more scientists collaborate

Started around 2006

>20 BSc students

>20 MSc students

13,269 SVN commits, by 52 authors

Open Universiteit

www.ou.nl



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 Keuning

Windesheim University of Applied Sciences & Open University of the Netherlands



For her PhD, Hieke works on tutors for (imperative) programming.

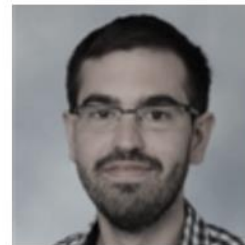


Alex Gerdes

Gothenburg University



Alex is the main architect of the functional programming tutor Ask-Elle.



Alejandro Serrano Mena

Utrecht University



Alejandro works on Ask-Elle.

💡 Ideas

Home

What we do

Projects

People

More information

Part 1:

Intelligent Tutoring Systems (ITS)

Open Universiteit
www.ou.nl



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.

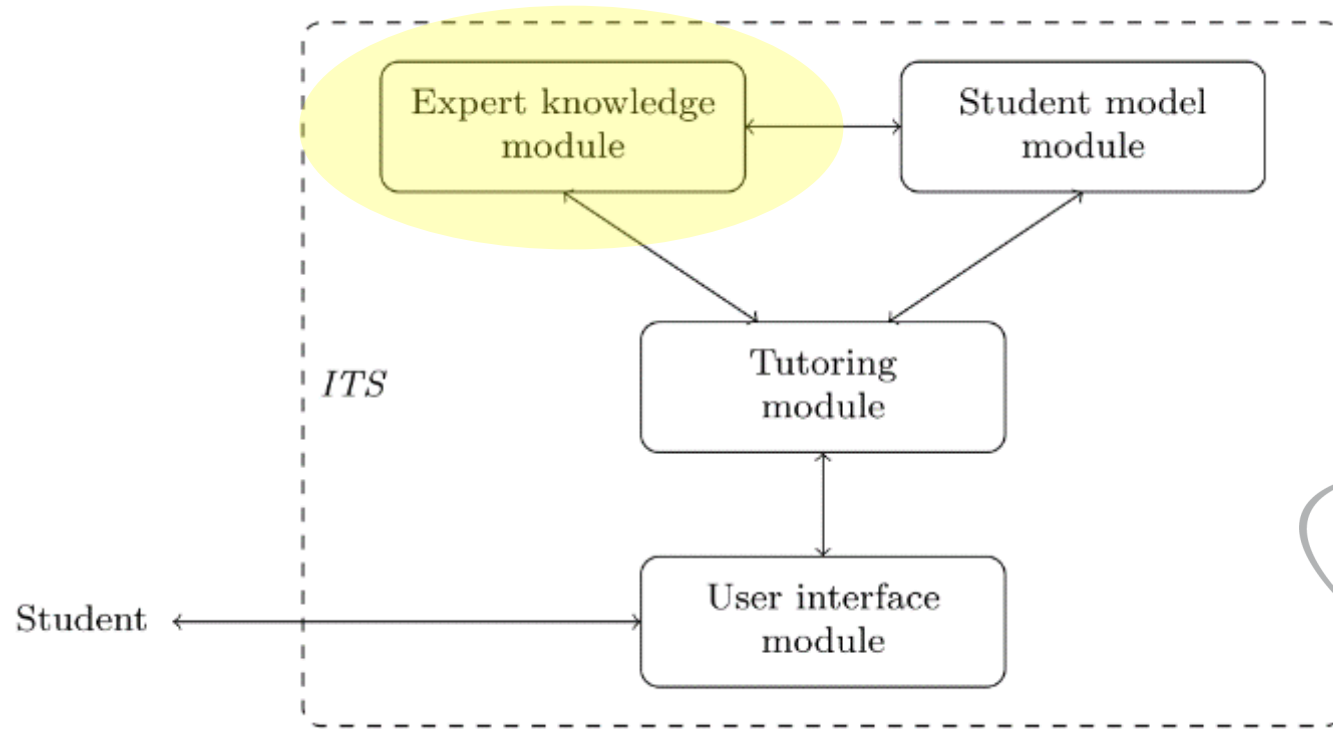
Keywords. Intelligent tutoring systems, knowledge components, learning events, tutoring

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

Open Universiteit
www.ou.nl



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

Open Universiteit

www.ou.nl



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

ideas: Feedback services for intelligent tutoring systems

[apache, education, library] [Propose Tags]

Ideas (Interactive Domain-specific Exercise Assistants) is a joint research project between the Open University of the Netherlands and Utrecht University. The project's goal is to use software and compiler technology to build state-of-the-art components for intelligent tutoring systems (ITS) and learning environments. The `ideas` software package provides a generic framework for constructing the expert knowledge module (also known as a domain reasoner) for an ITS or learning environment. Domain knowledge is offered as a set of feedback services that are used by external tools such as the digital mathematical environment (DME), MathDox, and the Math-Bridge system. We have developed several domain reasoners based on this framework, including reasoners for mathematics, linear algebra, logic, learning Haskell (the Ask-Elle programming tutor) and evaluating Haskell expressions, and for practicing communication skills (the serious game Communicate!).

Modules

[Index]
Ideas

Common

Ideas.Common.Classes

Versions [faq]

0.5.8, 0.6, 0.7, 1.0, 1.1, 1.2, 1.3, 1.3.1, 1.4, 1.5, 1.6, 1.7

Change log

[CHANGELOG.txt](#)

Dependencies

`base` (>=4.8 && <5), `blaze-builder` (>=0.4), `bytestring`, `case-insensitive`, `containers`, `Diff`, `directory`, `filepath`, `HDBC`, `HDBC-sqlite3`, `http-types`, `network`, `parsec`, `QuickCheck` (>=2.8), `random`, `semigroups` (==0.18.*), `streaming-commons` (<0.2), `time`, `uniplate`, `wai`, `wl-pprint` [details]

License

Apache-2.0

Copyright

(c) 2018

Author

Bastiaan Heeren, Alex Gerdes, Johan Jeuring

Maintainer

bastiaan.heeren@ou.nl

Category

Education

Home page

Open Universiteit
www.ou.nl



Interactive explorer for domain reasoners

IDEAS

Index

Exercises 44

Services 42

Exercise

Information

Strategy

Rules

Constraints

Examples

Derivations

Test report

Exercise `algebra.equations.linear`

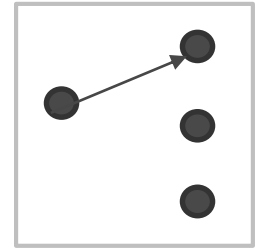
solve a linear equation

Derivation

```
3/4*x-(x-1) == 3+2[1/2]*(x-1)
  => algebra.equations.linear.remove-div,factor=4
-x+4 == 12+10*(x-1)
  => algebra.equations.linear.distr-times
-x+4 == 12+10*x-10
  => algebra.equations.linear.merge
-x+4 == 2+10*x
  => algebra.equations.linear.var-left,term=10*x
-11*x+4 == 2
  => algebra.equations.coverup.onevar.plus
-11*x == -2
  => 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):

```
distr = rule "distr" $ \a b c -> a*(b+c) :~> a*b + a*c
```

Rules are used for:

- recognizing steps
- suggesting possible next steps

Open Universiteit
www.ou.nl



Implementing rewrite rules

distr :: Rule Expr

distr = rule "distr" \$ \a b c -> 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)

JFP 20 (3 & 4): 375–413, 2010. © Cambridge University Press 2010
doi:10.1017/S0956796810000183 First published online 27 September 2010 375

A lightweight approach to datatype-generic rewriting

THOMAS VAN NOORT
*Institute for Computing and Information Sciences, Radboud University Nijmegen,
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
(e-mail: thomas@cs.ru.nl)*

ALEXEY RODRIGUEZ YAKUSHEV
and STEFAN HOLDERMANS
*Vector Fabrics, Paradijslaan 28, 5611 KN Eindhoven, The Netherlands
(e-mail: {alexey,stefan}@vectorfabrics.com)*

JOHAN JEURING
*Department of Information and Computing Sciences, Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
and
School of Computer Science, Open University of the Netherlands,
P.O. Box 2960, 6401 DL Heerlen, The Netherlands
(e-mail: johan@cs.uu.nl)*

BASTIAAN HEEREN
*School of Computer Science, Open University of the Netherlands,
P.O. Box 2960, 6401 DL Heerlen, The Netherlands
(e-mail: bastiaan.heeren@ou.nl)*

JOSÉ PEDRO MAGALHÃES
*Department of Information and Computing Sciences, Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
(e-mail: jpm@cs.uu.nl)*

Abstract

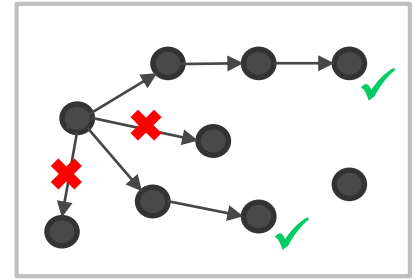
Term-rewriting systems can be expressed as generic programs parameterised over the shape of the terms being rewritten. Previous implementations of generic rewriting libraries require users to either adapt the datatypes that are used to describe these terms or to specify rewrite rules as functions. These are fundamental limitations: the former implies a lot of work for the user, while the latter makes it hard if not impossible to document, test, and analyze rewrite rules. In this article, we demonstrate how to overcome these limitations by making essential use of type-indexed datatypes. Our approach is lightweight in that it is entirely expressible in Haskell with GADTs and type families and can be readily packaged for use with contemporary Haskell distributions.

1 Introduction

Consider a Haskell datatype `Prop` for representing formulae of propositional logic,

```
data Prop = Var String | T | F | Not Prop | Prop :^: Prop | Prop :v: Prop,
```

Problem-solving procedures



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

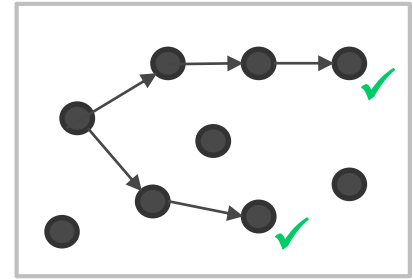
Problem-solving procedures are used for:

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

Open Universiteit
www.ou.nl



Problem-solving procedures



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

FindLCD ; *many (somewhere Convert)* ; Add ; *try Simplify*

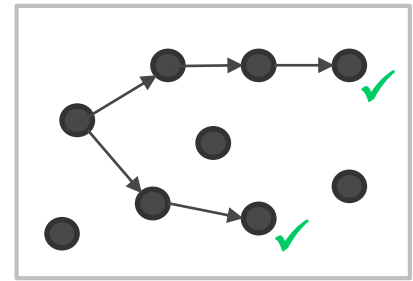
Resulting in:

$$\frac{1}{2} + \frac{4}{5} \xRightarrow{\text{FindLCD}} \frac{1}{2} + \frac{4}{5} \xRightarrow{\text{Convert}} \frac{5}{10} + \frac{4}{5} \xRightarrow{\text{Convert}} \frac{5}{10} + \frac{8}{10} \xRightarrow{\text{Add}} \frac{13}{10} \xRightarrow{\text{Simplify}} 1 \frac{3}{10}$$

Open Universiteit
www.ou.nl



Theoretical foundations



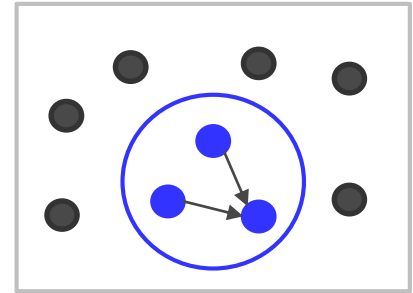
Problem-solving procedures:

- 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)

$\mathcal{T}(s ; t)$	$= \{x \mid x \in \mathcal{T}(s), \checkmark \notin x\} \cup \{xy \mid x\checkmark \in \mathcal{T}(s), y \in \mathcal{T}(t)\}$	(sequence)
$\mathcal{T}(s \mid t)$	$= \mathcal{T}(s) \cup \mathcal{T}(t)$	(choice)
$\mathcal{T}(\mu x.f(x))$	$= \mathcal{T}(f(\mu x.f(x)))$	(fixed point)
$\mathcal{T}(r)$	$= \{\epsilon, r, r\checkmark\}$	(rule)
$\mathcal{T}(\text{succed})$	$= \{\epsilon, \checkmark\}$	(success)
$\mathcal{T}(\text{fail})$	$= \{\epsilon\}$	(failure)



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

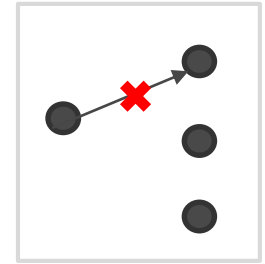
Normal forms are used for:

- recognizing steps
- rewriting atypical expressions, e.g. $4 + (-5)$
- deciding whether finished or not

Open Universiteit
www.ou.nl



Buggy rules



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

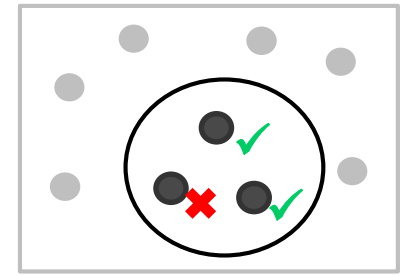
Buggy rules are used for:

- detecting common mistakes

Open Universiteit
www.ou.nl



Constraints



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

Open Universiteit
www.ou.nl



Feedback on the structure of hypothesis tests

numw@rx MT51 2017 group 2 >

LESSON Module 4 - T-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 this 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

Step construction area

MTS1 HS 9 and 11

a 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 $\alpha = .05$.

1 Step: State null hypothesis and alternative hypothesis

$H_0: \mu_D = 0$

$H_1: \mu_D \neq 0$

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

The test is

3 Step: Find critical value

$t_{crit} = 2.064$

4 Step: Determine rejection region

$t < t_{crit}$


Action: ? ↩

Domain reasoner feedback

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

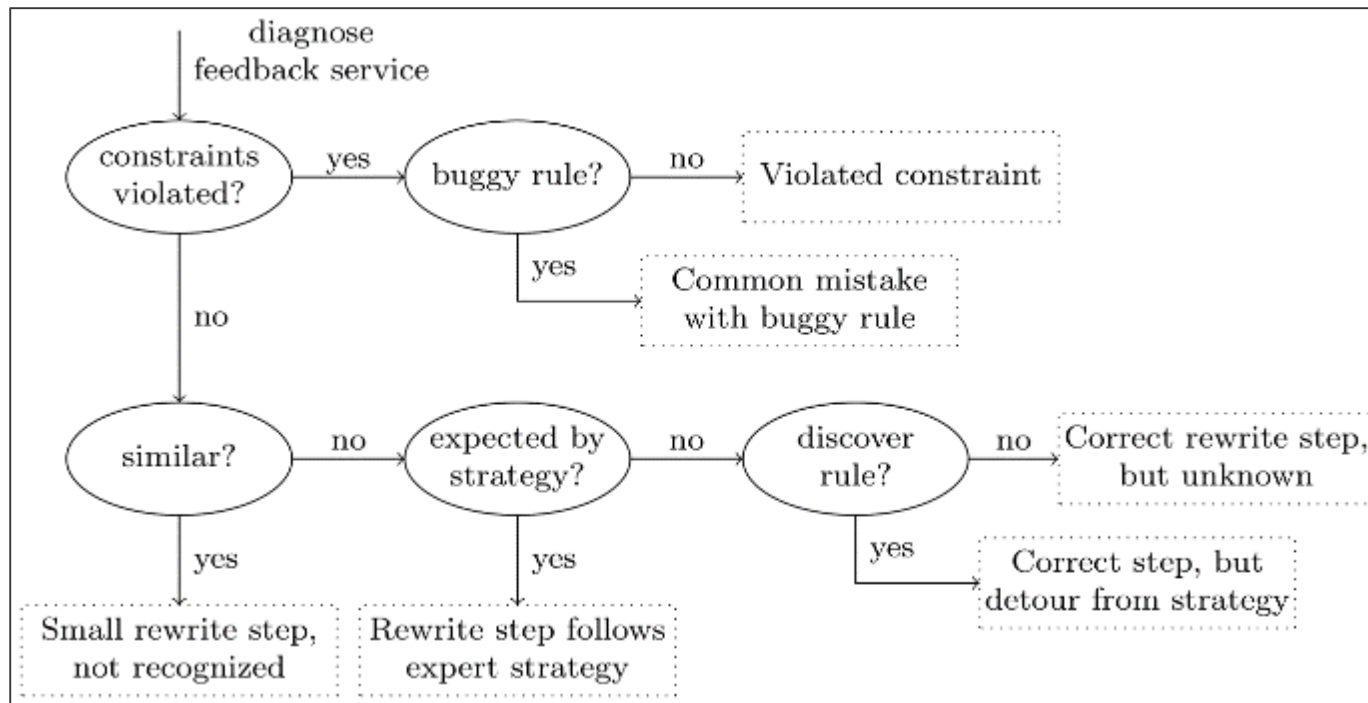
Conclusion: There significant difference between the judgments of ones own reaction and the reaction of a peer.

01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 Partial scores



Feedback on the structure of hypothesis tests

The tutor's **diagnose feedback service** combines several knowledge components:



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 (Saarbrücken)

On your right hand side you see the first three of a series Matryoshka dolls. The puppets fit into each other, due to a decreasing height. The biggest puppet is 32 cm high. Each next puppet is 25% smaller than the previous one. In this sequence, there are no puppets smaller than 6 cm.



How many puppets are there in this series?
Write down your intermediate steps.

Domain reasoner for axiomatic proofs

Ideas - LogAx

NL

EN

Help

Log out

Axiomatic

New exercise (N)

1	$p \mid\text{-} p$	Assumption	X
2	$p \rightarrow q \mid\text{-} p \rightarrow q$	Assumption	X
998	$p, p \rightarrow q, q \rightarrow r \mid\text{-} r$		X
999	$p \rightarrow q, q \rightarrow r \mid\text{-} p \rightarrow r$	Deduction 998	X
1000	$q \rightarrow r \mid\text{-} (p \rightarrow q) \rightarrow (p \rightarrow r)$	Deduction 999	

Rule: Modus Ponens

$(\Sigma \vdash_S \varphi), (\Delta \vdash_S \varphi \rightarrow \psi) \Rightarrow \Sigma \cup \Delta \vdash_S \psi$

$\Sigma \vdash_S \varphi$ stepnr

$\Delta \vdash_S \varphi \rightarrow \psi$ stepnr

$\Sigma \cup \Delta \vdash_S \psi$ stepnr

Complete the proof in two directions

Fill in the template, then the rule is applied automatically

Open Universiteit
www.ou.nl



Domain reasoner for functional programming

The screenshot shows the Ask-Elle IDE interface. The window title is "ASK-Elle". The main area is divided into three panes:

- All Exercises:** A tree view on the left showing a hierarchy of exercises under "programming" > "list". The "myreverse" exercise is selected.
- Description:** A text area containing the problem statement: "Write a function that reverses a list: myreverse :: [a] -> [a]. For example: Data.List> myreverse 'A man, a plan, a canal, panama!' '!amanap ,lanac a ,nalp a ,nam A' Data.List> myreverse [1,2,3,4] [4,3,2,1]".
- Editor:** A code editor showing the Haskell code:

```
1 myreverse = ?  
2   where  
3     reverse' acc ? = ?  
4
```

 A red arrow points to the question marks in the code, with the text "Holes for unfinished parts in the program".
- Help:** A panel on the right with a "Help" title and a "You can follow one of the following strategies:" section. It contains two hints: "Hint 1: Introduce the constructor pattern []" and "Hint 2: Refine the current term to" followed by a code snippet:

```
myreverse =  
  ?  
  where  
    reverse' acc [] =  
      ?
```

 A red arrow points to the question marks in the hint code, with the text "Hint sequences".

www.ou.nl



Tutoring system to learn code refactoring

Choose exercise:

ref.sumvalues

Start exercise

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?

Type code here:

```
1 public static int sumValues(int [] values,
2                             boolean positivesOnly) {
3     int sum = 0;
4     for (int i = 0; i < values.length; i++) {
5         if (positivesOnly == true) {
6             if (values[i] >= 0) {
7                 sum += values[i];
8             }
9         }
10        else {
11            sum += values[i];
12        }
13    }
14    return sum;
15 }
```

foreach (points to the for loop)

== true (points to the condition in the inner if)

>= 0 (points to the condition in the inner if)

duplication (points to the two separate if blocks)

Check progress

Hint

Hint tree

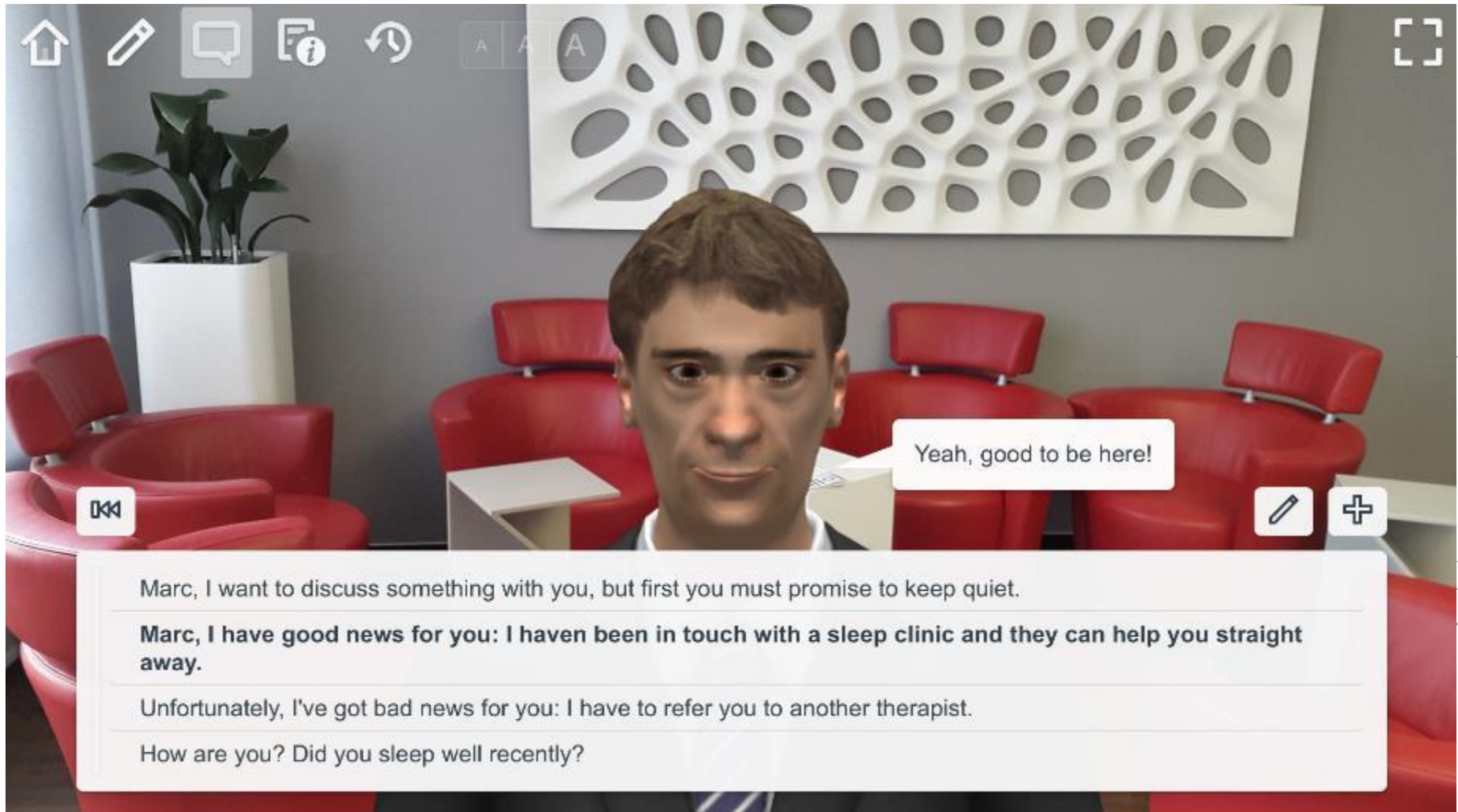
Show next step

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

Open Universiteit
www.ou.nl



Domain reasoners for communication skills



OU Master theses about domain reasoners

Practice with the evaluation of a Haskell Expression

Haskell Expression

Output

Start
Select ▾

Options

Outermost evaluation strategy
 Innermost evaluation strategy

Next step

Diagnose

Hints

Show number of steps left
Show all rules that can be applied

Show next rule
Show next step
Do next step

Derivation

```

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

Steps remaining: 11

Rules that can be applied independent of strategy:

- Apply the append rule to concatenate two lists
- Apply the sum rule to sum up all elements of a list

Next rule that should be applied according the strategy:

Apply the sum rule to sum up all elements of a list

Next derivation step:

foldl (+) 0 ([3,7] ++ [5])

Next rule that should be applied according the strategy:

Apply the append rule to concatenate two lists

Programming Tutor

Choose exercise:

Start exercise

Description: Calculate and print the sum of all odd positive numbers under 100.

Type code here:

For If If-else While Clear

```

1 int sum = 0;
2 for (int i = 1; i < ?; ?)
3 {
4
5 }
          
```

Check
All hints

Hint

- Options
 - Create a loop that increments with 2
 - loop from 1 to 3 to 5...stopping at 100
 - When to continue looping? [Expand](#)
 - Create a loop and test for odd numbers with % [Expand](#)

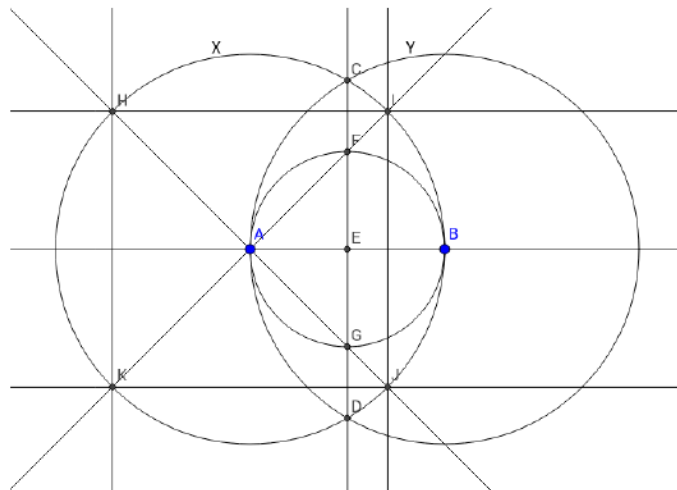


Figure 1: Constructing a square using circles and lines (screenshot from GeoGebra – a mathematics tool that allows drawing of geometric structures).

MicK
About

Microcontroller and programming language

Values of definitions, registers and volatile variables for this microcontroller and programming language:

```

UCSR0A = 0b00001111
UDRE0 = 5
          
```

```

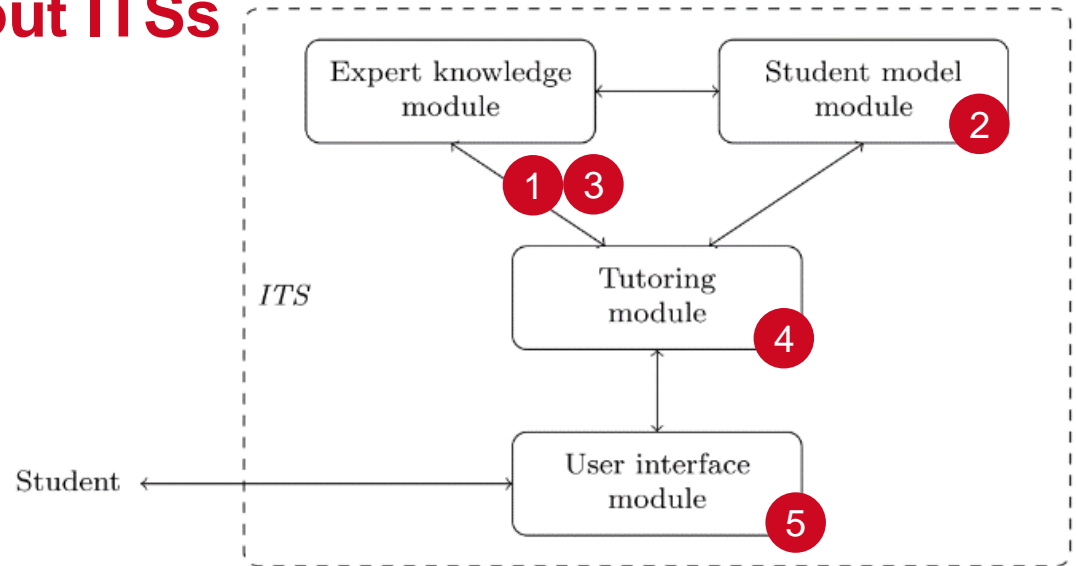
while( !(UCSR0A & (1 << UDRE0)) ) { ; }
while( !(0b00001111 & (1 << 5)) ) { ; }
while( !(0b00001111 & (0b00000001 << 5)) ) { ; }
while( !(0b00001111 & 0b00100000) ) { ; }
while( ! (false) ) { ; }
while( true ) { ; }
          
```

Validate
Hint
Show
Copy
Stop

Information

That is correct.
You have finished the task successfully!

OU Master theses about ITSs



1. **Gideon Teeuwen** (2016). Comparing architectural styles for distributed expert knowledge modules in intelligent tutoring systems
2. **Johan Eikelboom** (2017), Towards lightweight student modelling for Functional Programming Tutors
3. **Niels Kolthoff** (2019). ITS Authoring – Integrating a distributed expert knowledge module into existing authoring tools
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/>

