# Automated feedback for intelligent tutoring systems



Bastiaan Heeren

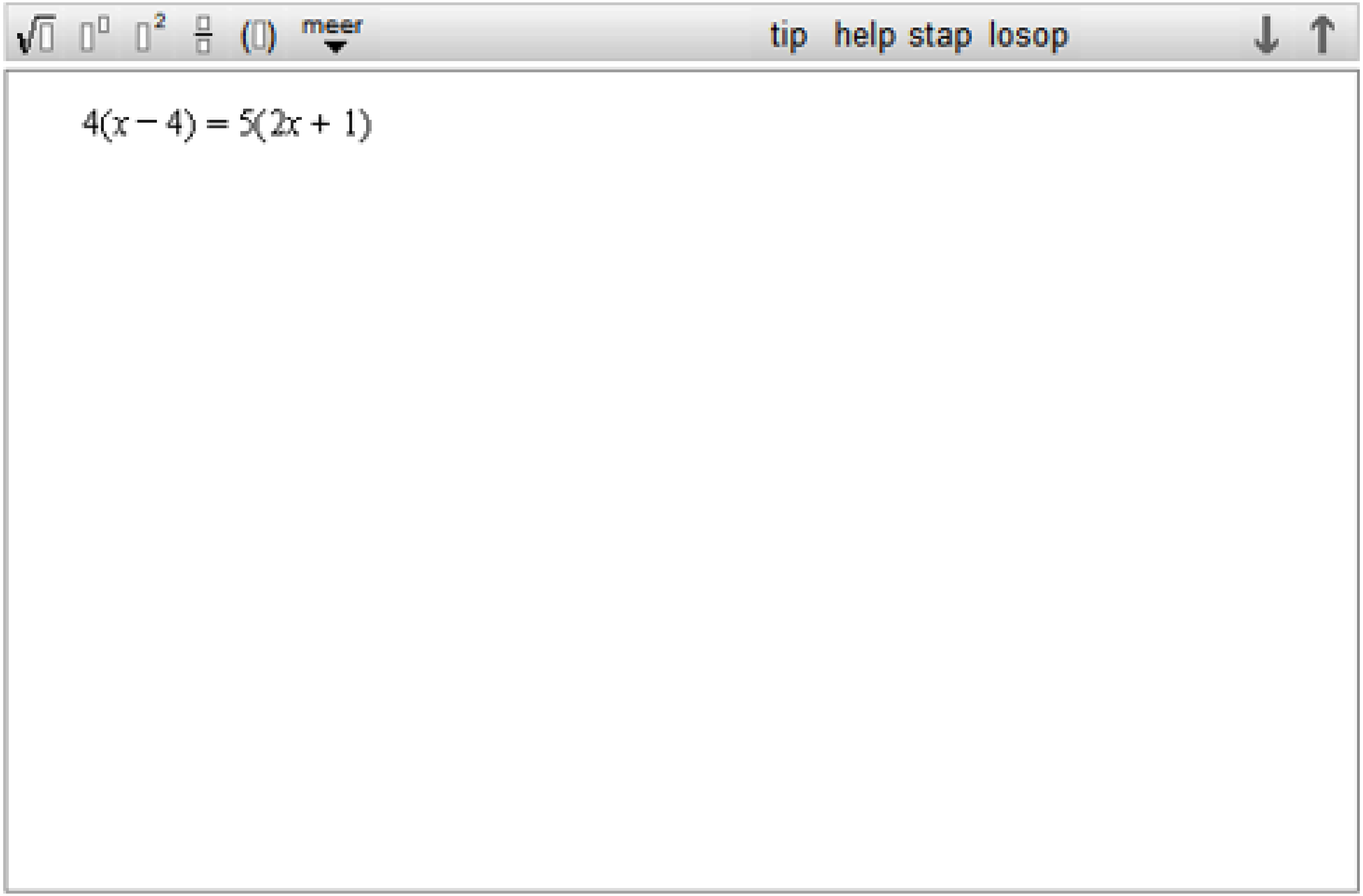
Informatica studiedag, 6-9-2019, Utrecht

Open Universiteit
www.ou.nl

```
------------------------------------------------------------
-- Template for the hypothesis testing strategy

hypothesisStrategy :: LabeledStrategy ComponentSet
hypothesisStrategy = label "Hypothesis testing" $
   label "Preparation" (whileNotReady $ choice $
      [ 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
```

**www.numworx.nl**



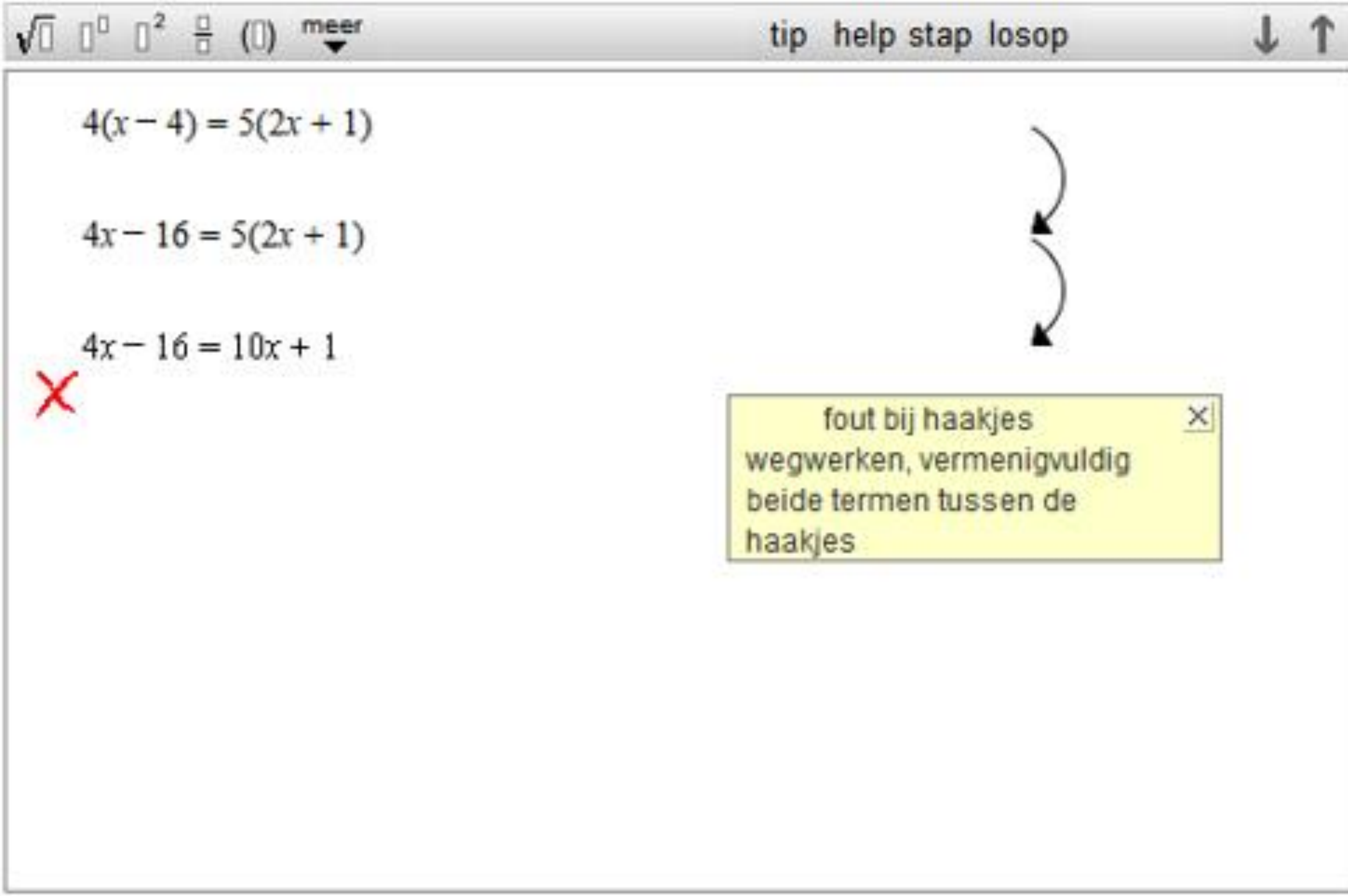
**www.numworx.nl**

meer tip help stap losop

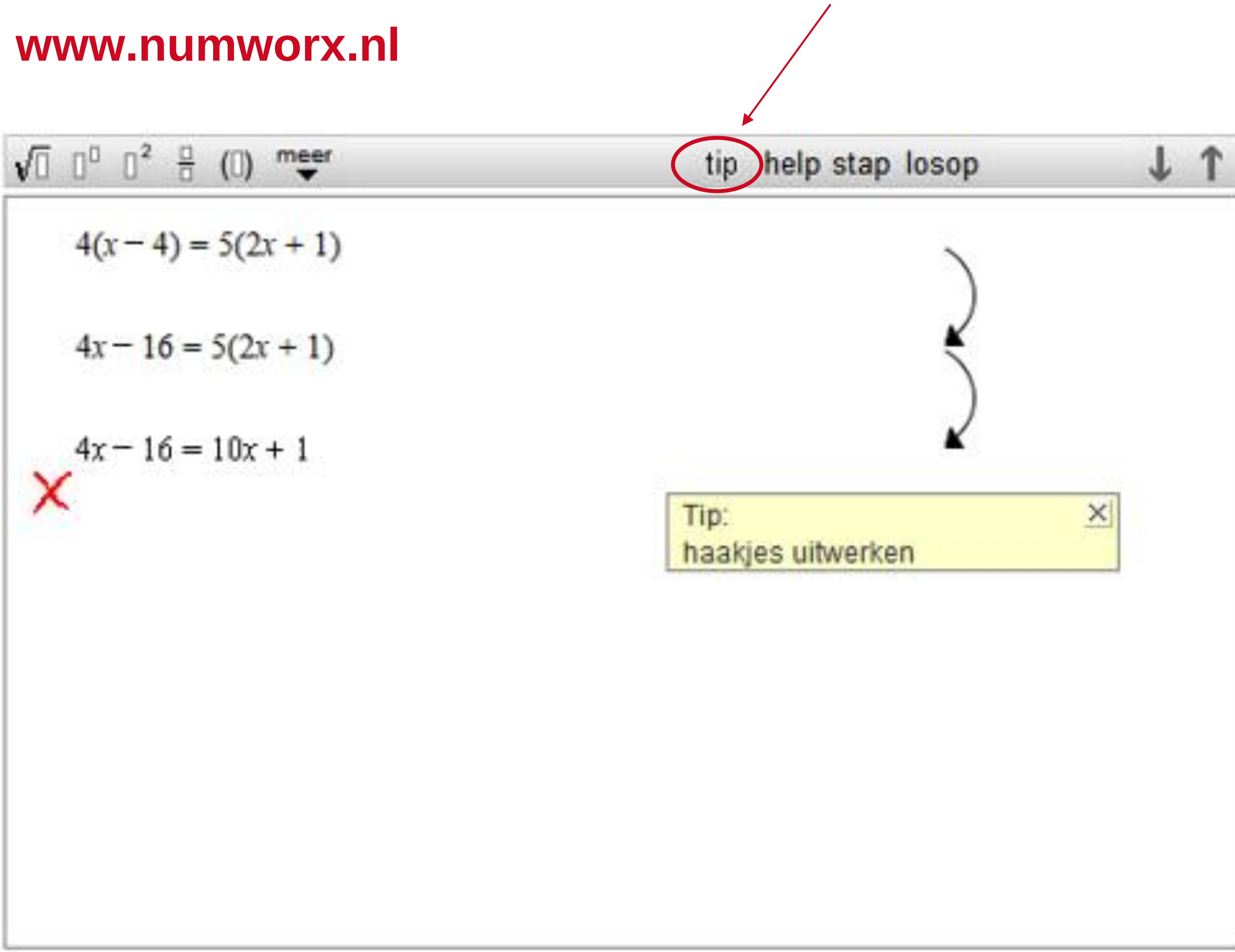
$$4(x - 4) = 5(2x + 1)$$

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

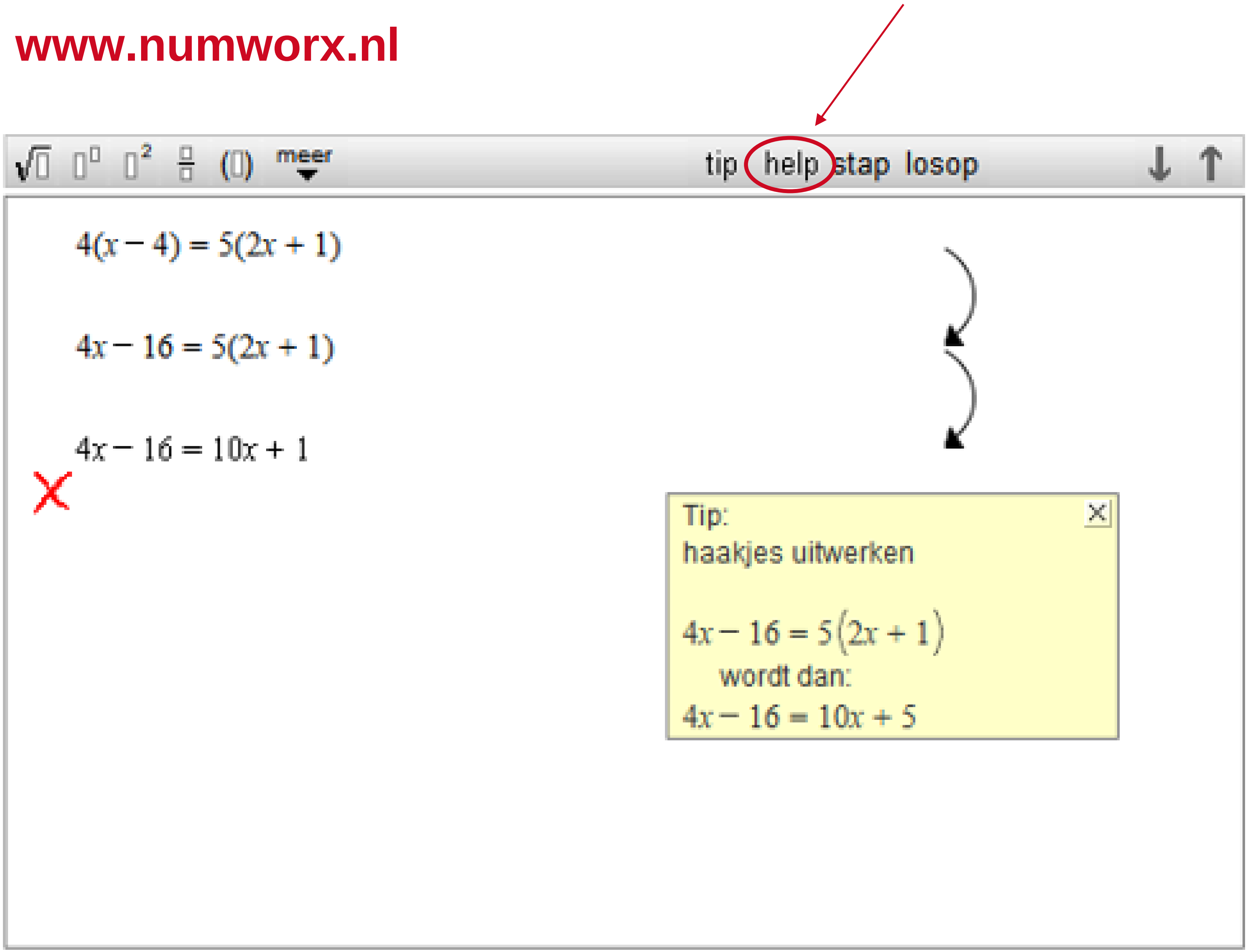
# www.numworx.nl



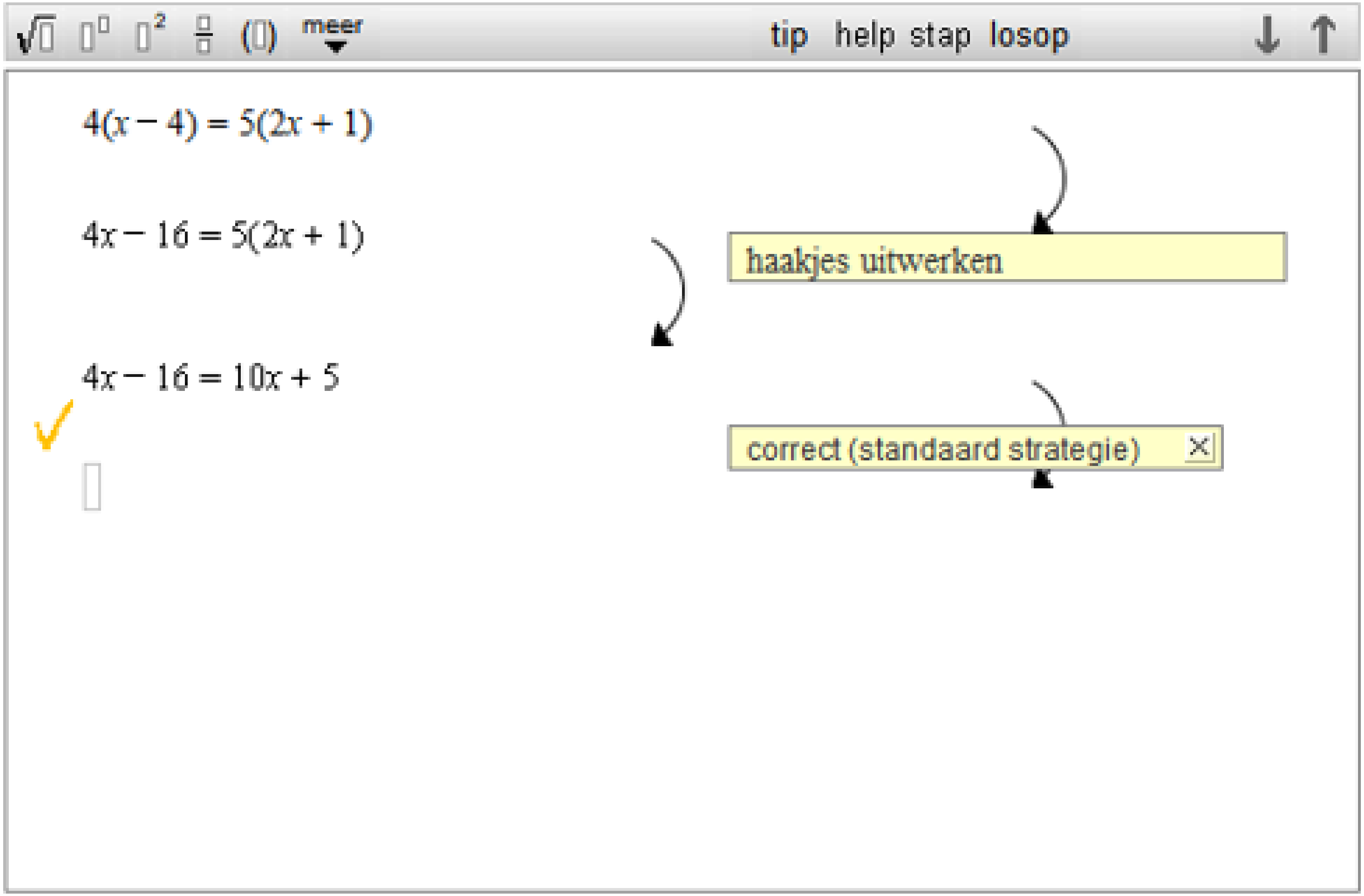
# www.numworx.nl



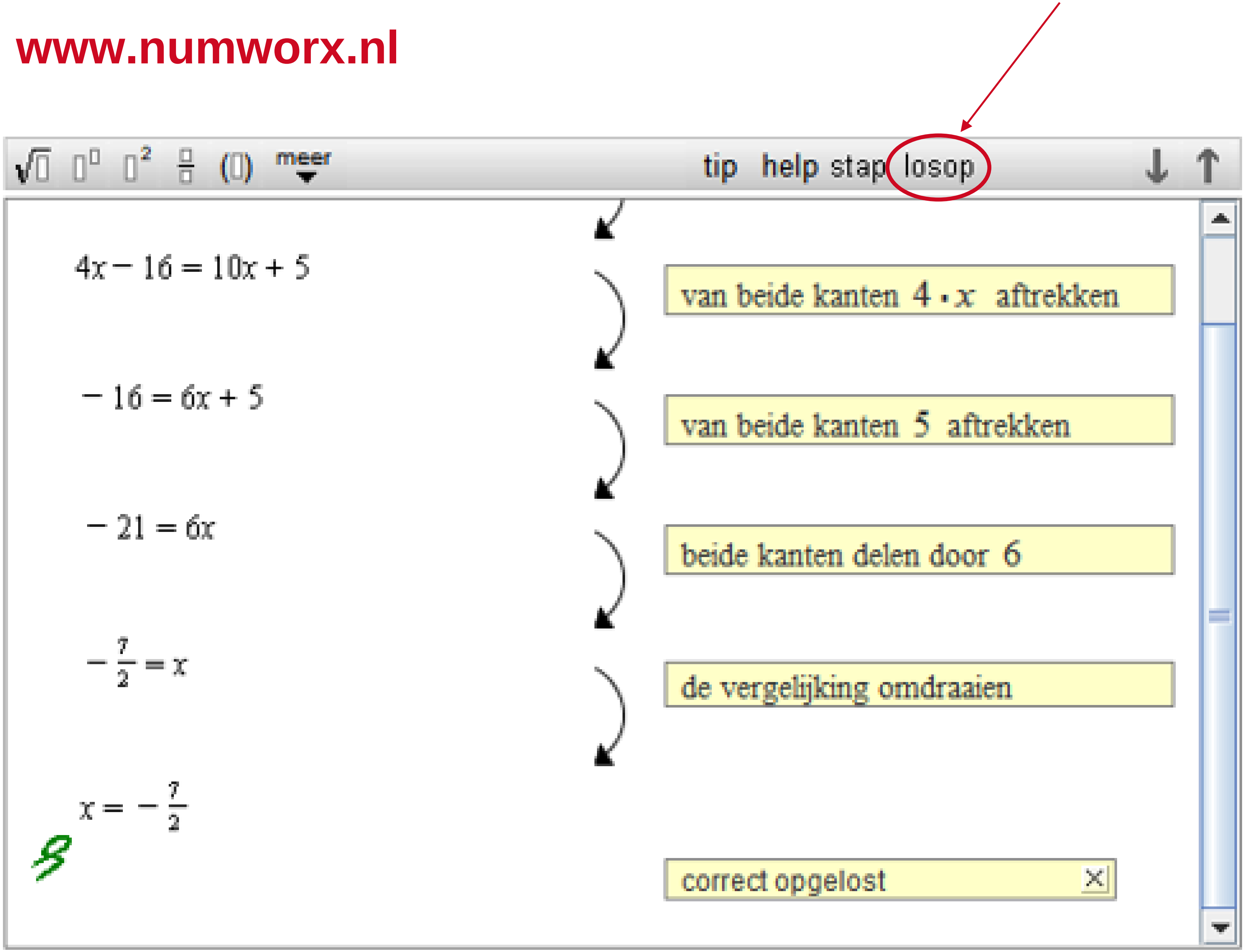
**www.numworx.nl**



## www.numworx.nl



## www.numworx.nl



# Overview of this talk

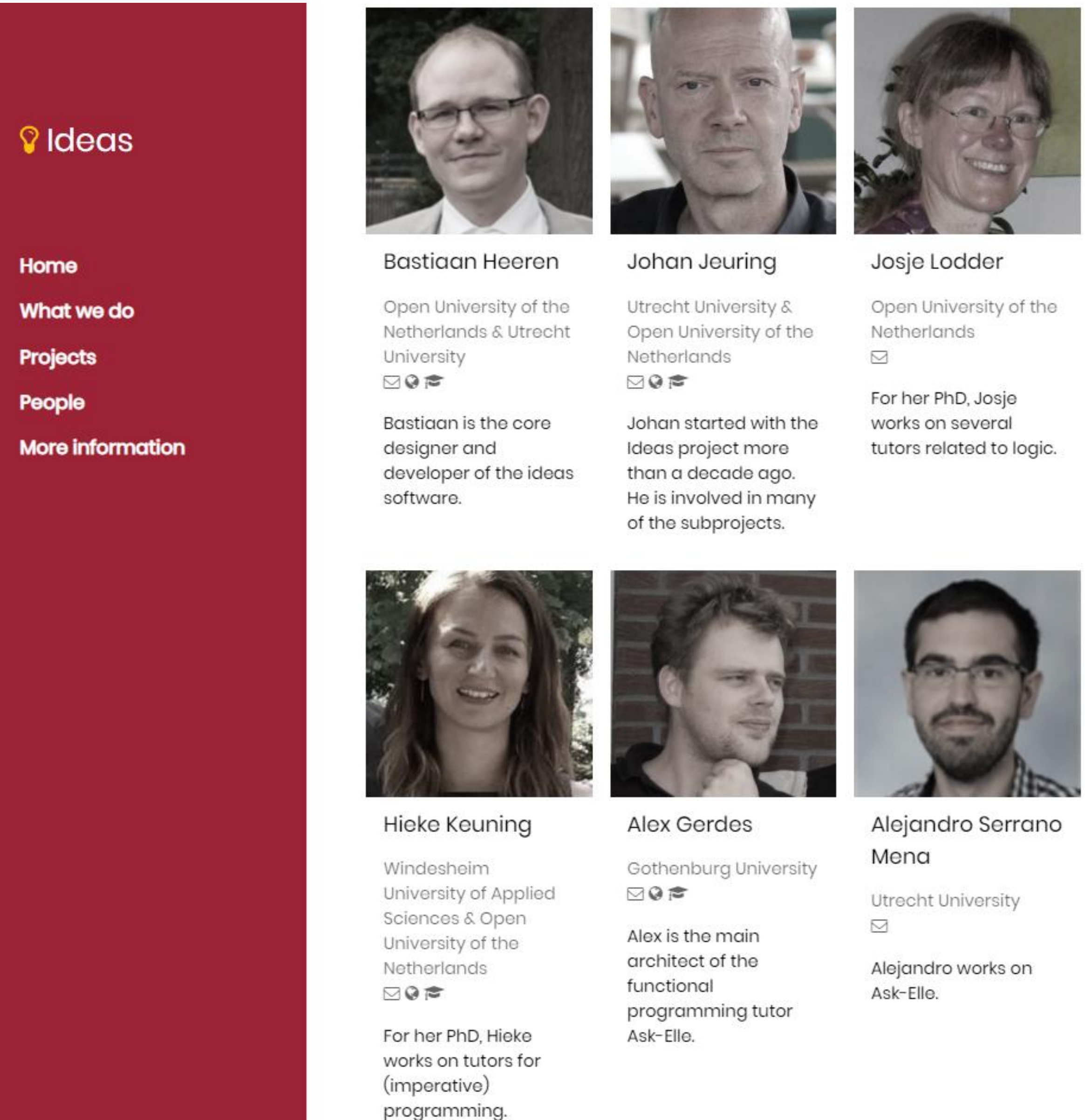
1. Intelligent Tutoring Systems (ITS)
   – Domain reasoners
   – Feedback services
2. Expert domain knowledge
   – Problem-solving procedures
   – Granularity (step-size)
3. Examples of domain reasoners

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

Ideas

Part 1:

# Intelligent Tutoring Systems (ITS)

Open Universiteit
www.ou.nl

# Inner and outer loops (VanLehn 2006)

## The Behavior of Tutoring Systems

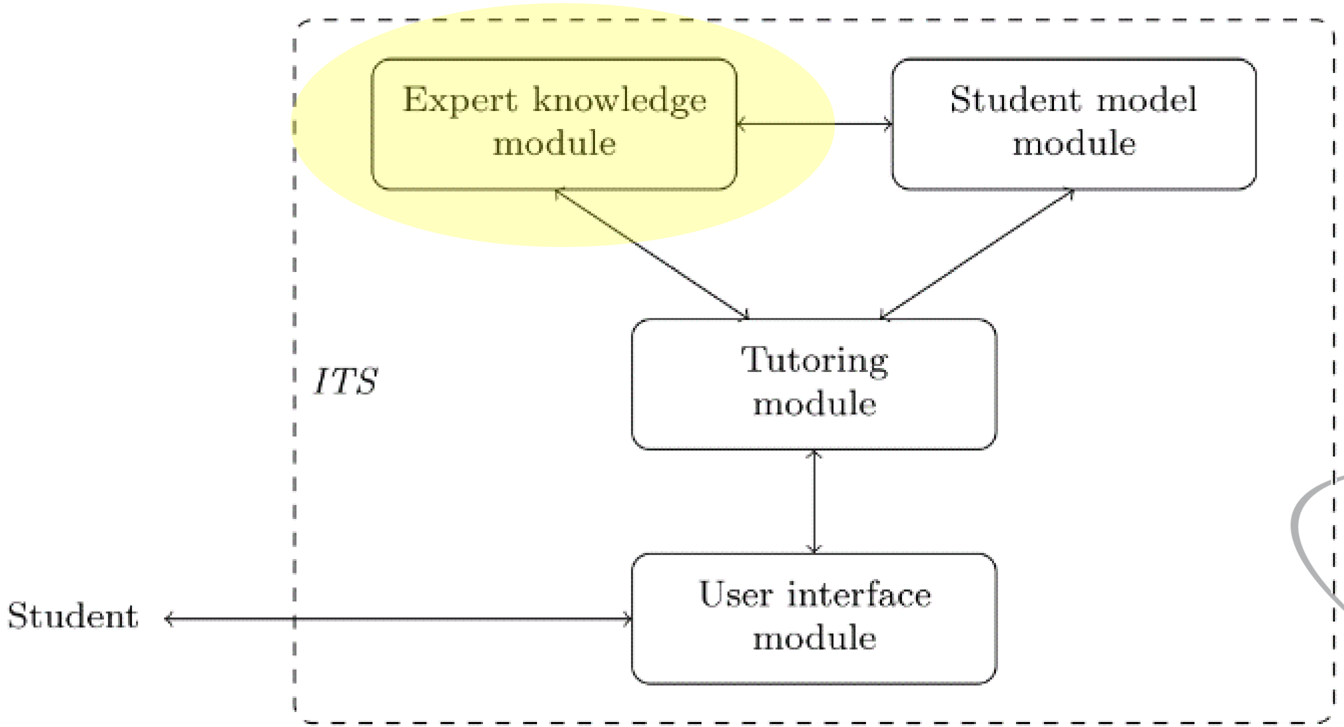
**Kurt VanLehn,** *LRDC, University of Pittsburgh, Pittsburgh, PA, USA*
*VanLehn@pitt.edu*

**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

# Four component ITS architecture




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

Open Universiteit
www.ou.nl

# 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

Open Universiteit
www.ou.nl

# 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:

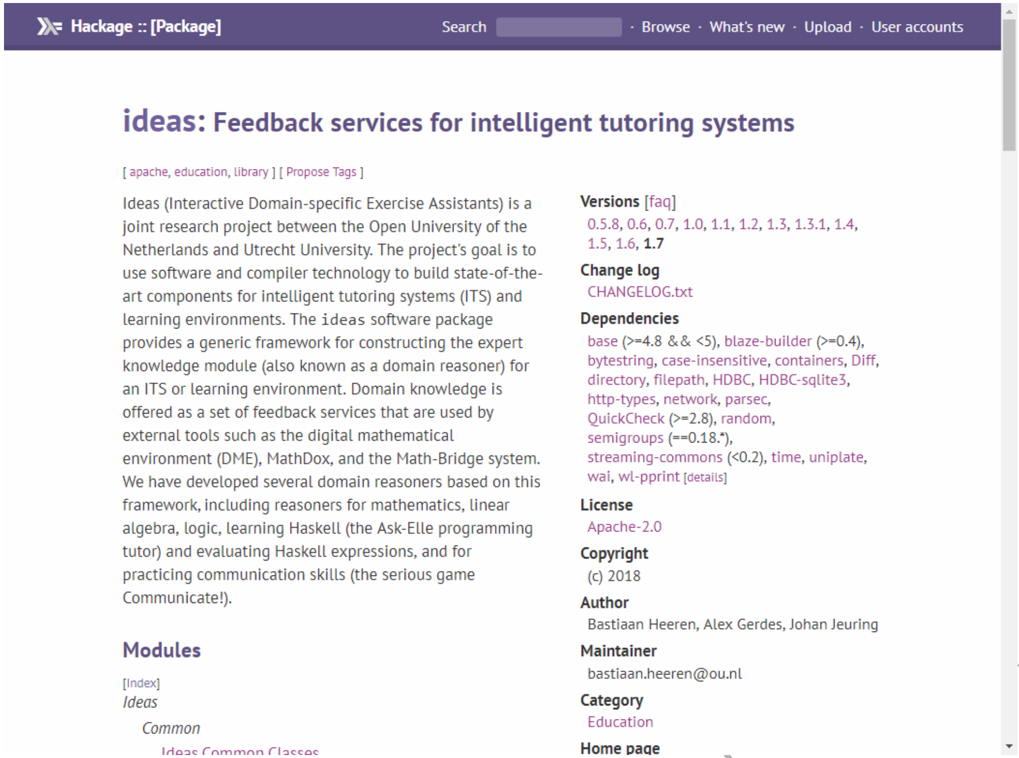
- Am I finished?
- Give me a next-step hint
- Give me a worked-out solution
- Is my step correct (step diagnosis)?
  - If yes: does the step bring me closer to a solution?
  - If no: is it a common mistake?

Open Universiteit
www.ou.nl

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/

Ideas

Interactive Domain-specific Exercise Assistants

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

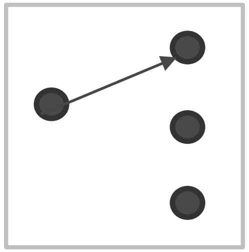
**Exercise algebra.equations.linear**
*solve a linear equation*

## 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

*version 1.6.1 (revision 07564cf39b3905f72d0eb5137413719613b4a95d, Mon Dec 4 09:37:03 2017 +0100)*

# 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

# Implementing rewrite rules

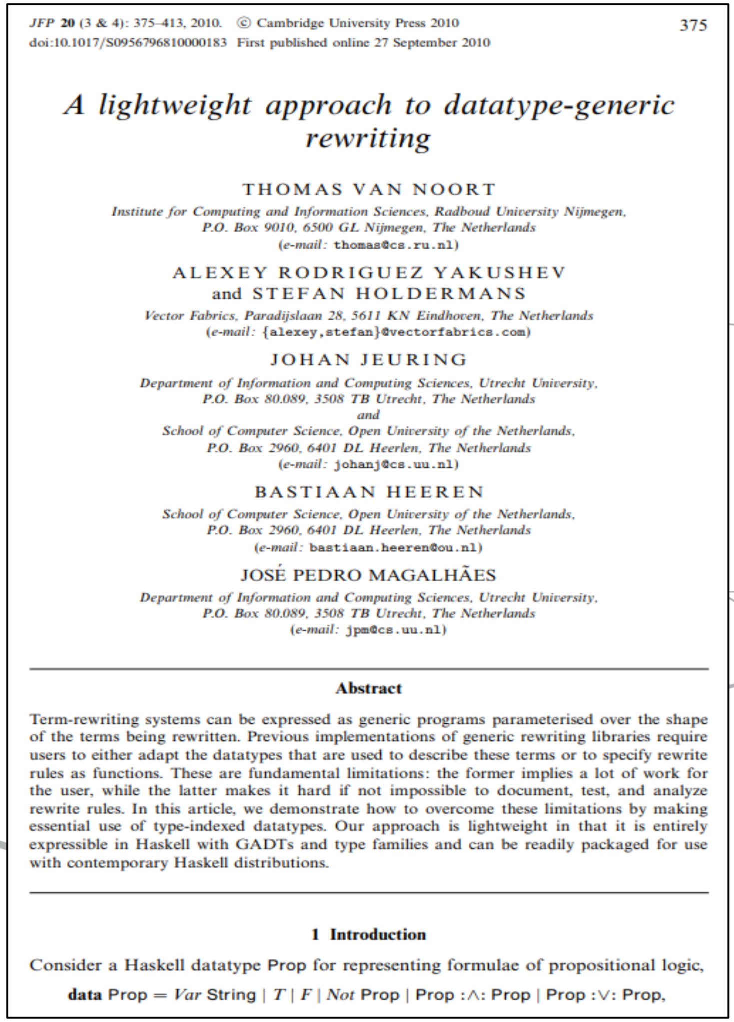
```
distr :: Rule Expr

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:* johanj@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**

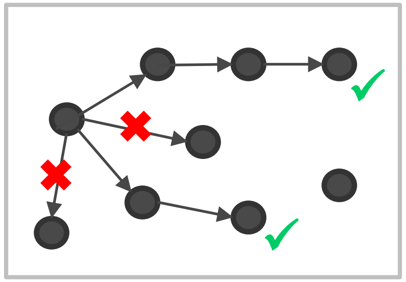
Term-rewriting systems can be expressed as generic programs parameterised over the shape of the terms being rewritten. Previous implementations of generic rewriting libraries require users to either adapt the datatypes that are used to describe these terms or to specify rewrite rules as functions. These 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 :∨: Prop,

# Problem-solving procedures



Problem-solving procedures describe sequences of rule applications that solve a particular task

Example procedure for adding two fractions:

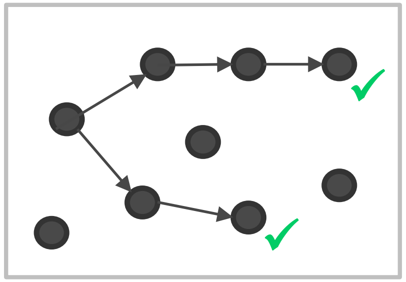
1. find the lowest common denominator (LCD)
2. convert fractions to LCD as denominator
3. add the resulting fractions
4. simplify the result

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



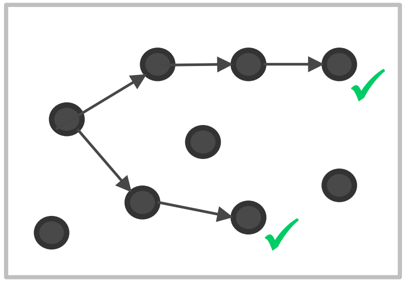
We have developed a domain-specific language for specifying procedures: sequence, choice, repeat, try, prefer, somewhere, etc.

$$\texttt{FindLCD} \; ; \; \mathit{many} \, (\mathit{somewhere} \; \texttt{Convert}) \; ; \; \texttt{Add} \; ; \; \mathit{try} \; \texttt{Simplify}$$

Resulting in:

$$\frac{1}{2} + \frac{4}{5} \overset{\texttt{FindLCD}}{\Longrightarrow} \frac{1}{2} + \frac{4}{5} \overset{\texttt{Convert}}{\Longrightarrow} \frac{5}{10} + \frac{4}{5} \overset{\texttt{Convert}}{\Longrightarrow} \frac{5}{10} + \frac{8}{10} \overset{\texttt{Add}}{\Longrightarrow} \frac{13}{10} \overset{\texttt{Simplify}}{\Longrightarrow} 1\,\frac{3}{10}$$

# Theoretical foundations



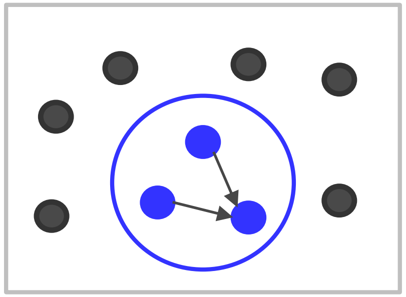
Problem-solving procedures:

- 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)

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

Open Universiteit
www.ou.nl

# Normal forms (equivalence classes)



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

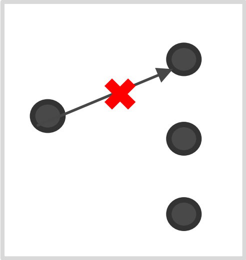
Example: 10 + 5x ≈ 5x + 10 ≈ 5x + 5·2

- 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: $5x = 2x + 3 \rightarrow 7x = 3$

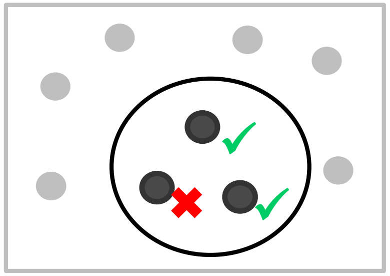
- Buggy rules are often associated with a sound rule

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

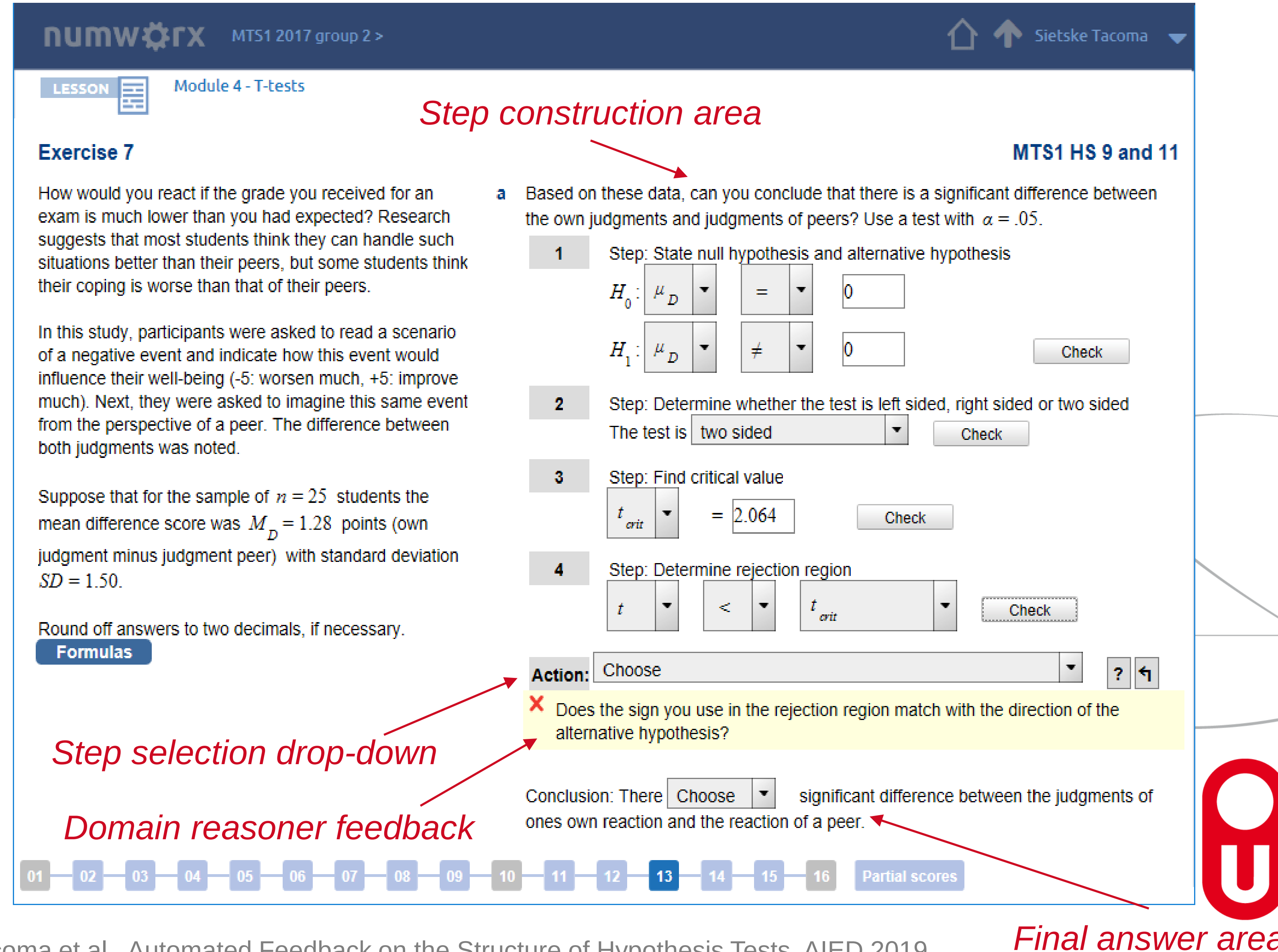
- Based on theory of learning from performance errors (Ohlsson 1992)

Constraints are used for:

- checking properties or attributes
- reporting violations

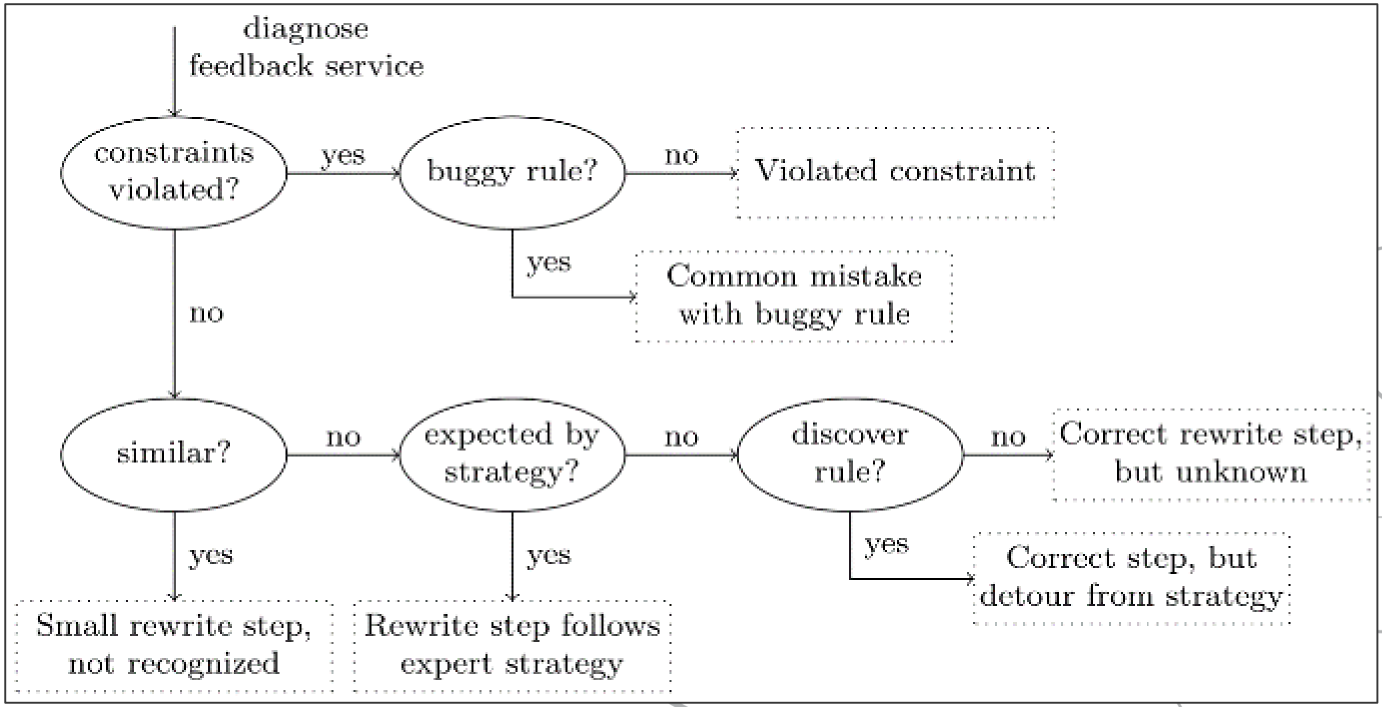
Open Universiteit
www.ou.nl

# Feedback on the structure of hypothesis tests



# Feedback on the structure of hypothesis tests

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




Open Universiteit
www.ou.nl

Part 3:

# Examples of domain reasoners

Advise-Me

# 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)

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.

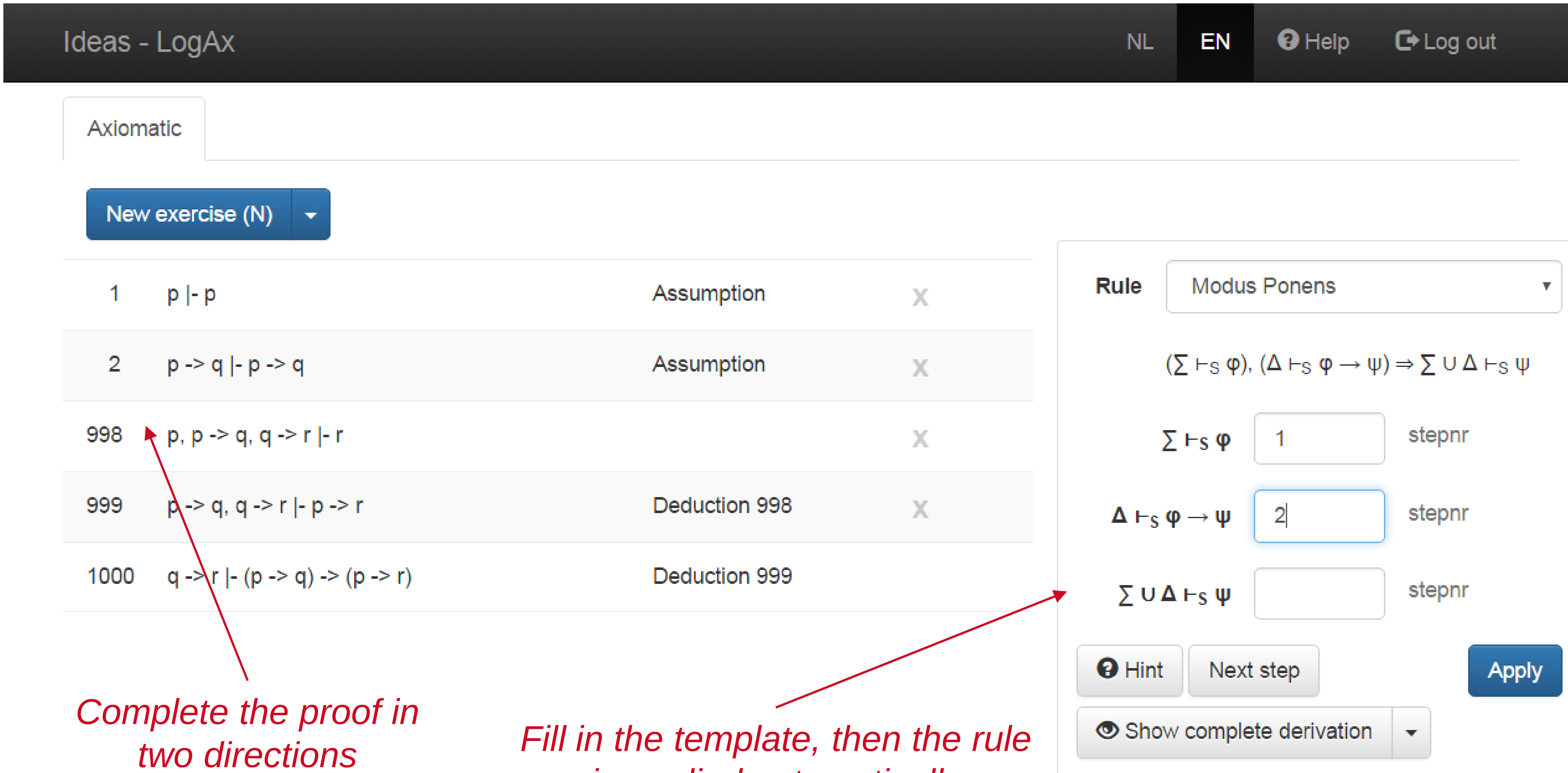
erasmus+ meer perspectief

Erasmus+

Open Universiteit www.ou.nl

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

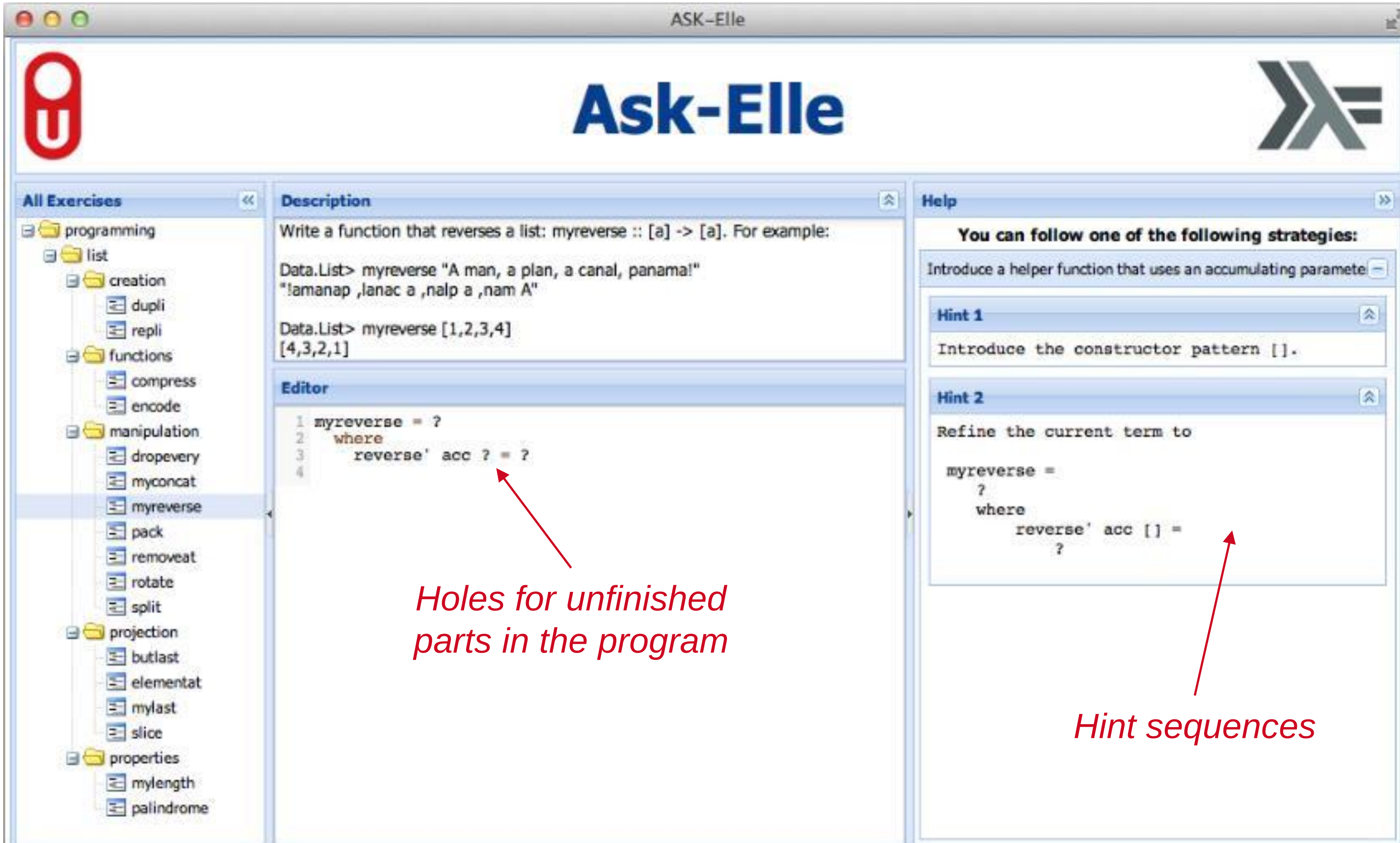



*Complete the proof in two directions*

*Fill in the template, then the rule is applied automatically*

Open Universiteit
www.ou.nl

# Domain reasoner for functional programming




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:

```
public static int sumValues(int [] values,
                            boolean positivesOnly) {
    int sum = 0;
    for (int i = 0; i < values.length; i++) {
        if (positivesOnly == true) {
            if (values[i] >= 0) {
                sum += values[i];
            }
        }
        else {
            sum += values[i];
        }
    }
    return sum;
}
```

*foreach*

*== true*

*>= 0*

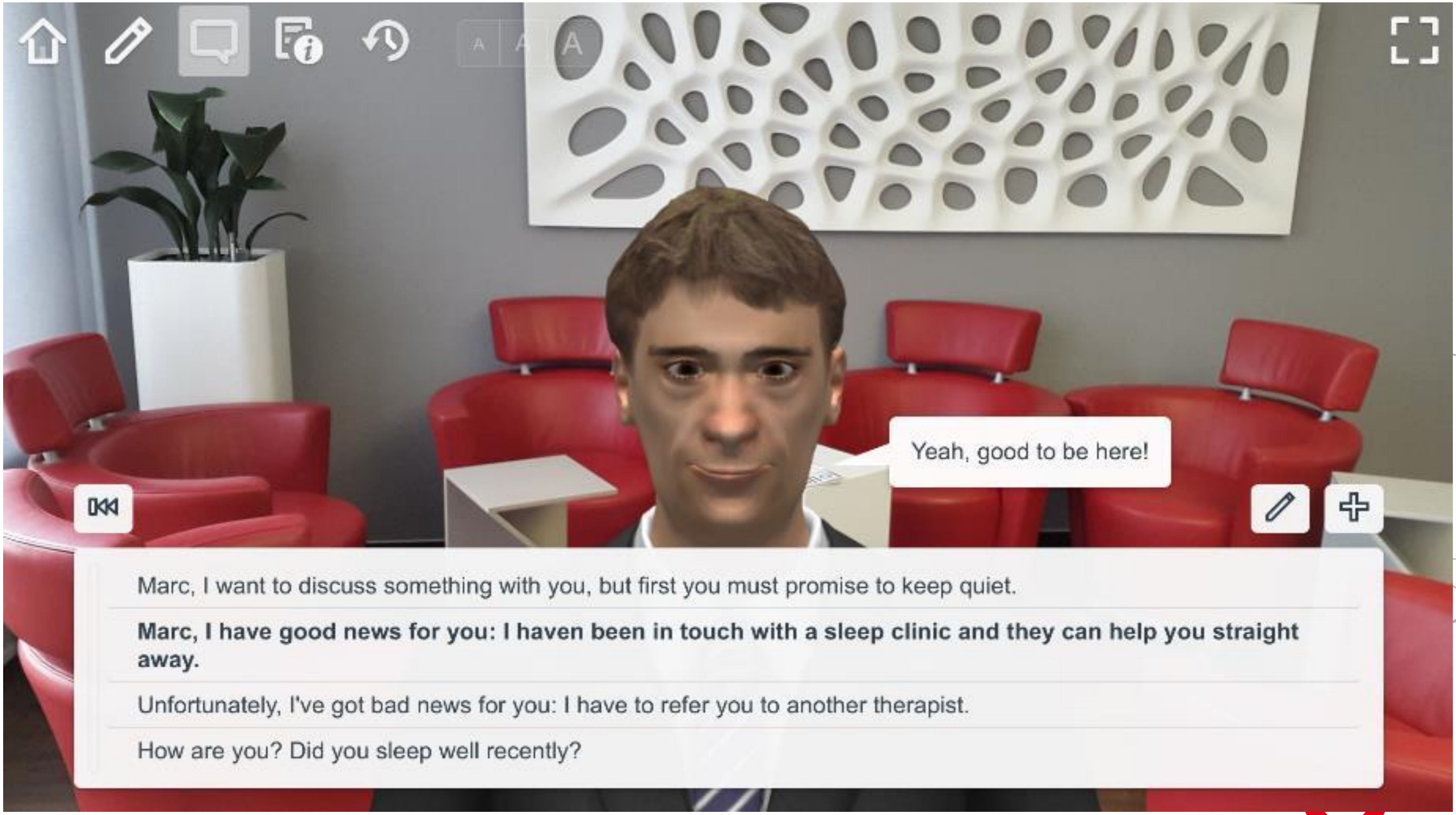
*duplication*

Check progress | Hint | Hint tree | Show next step

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

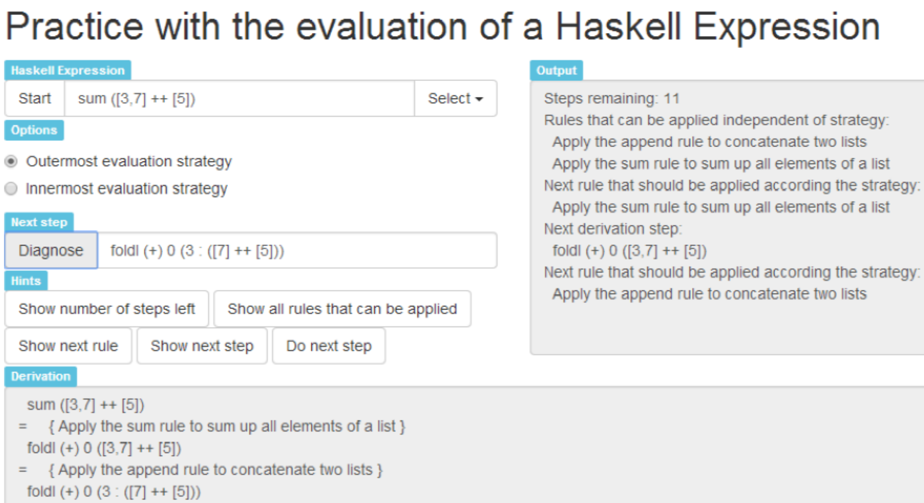
Open Universiteit
www.ou.nl

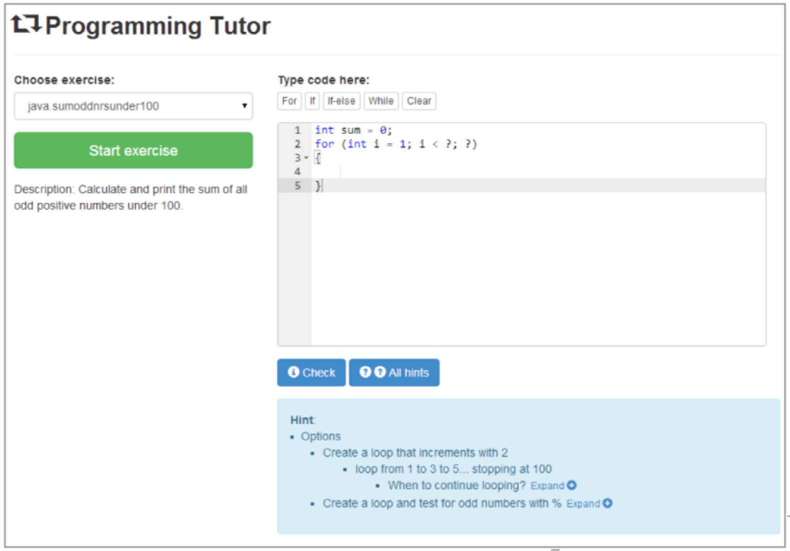
# Domain reasoners for communication skills




Johan Jeuring et al., Communicate! - A Serious Game for Communication Skills, EC-TEL 2015

# OU Master theses about domain reasoners



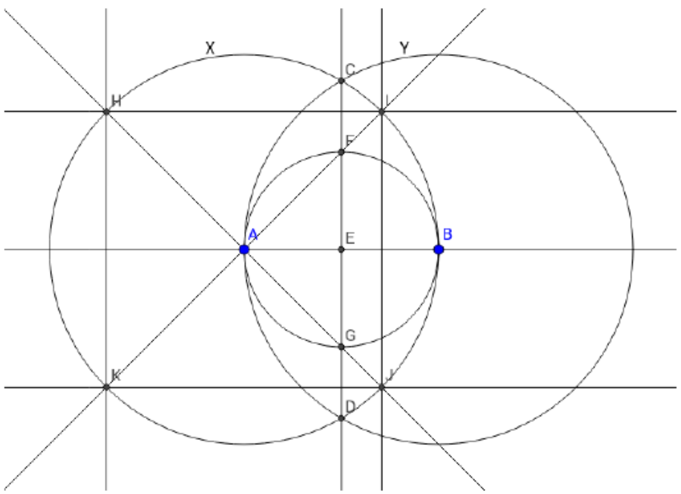
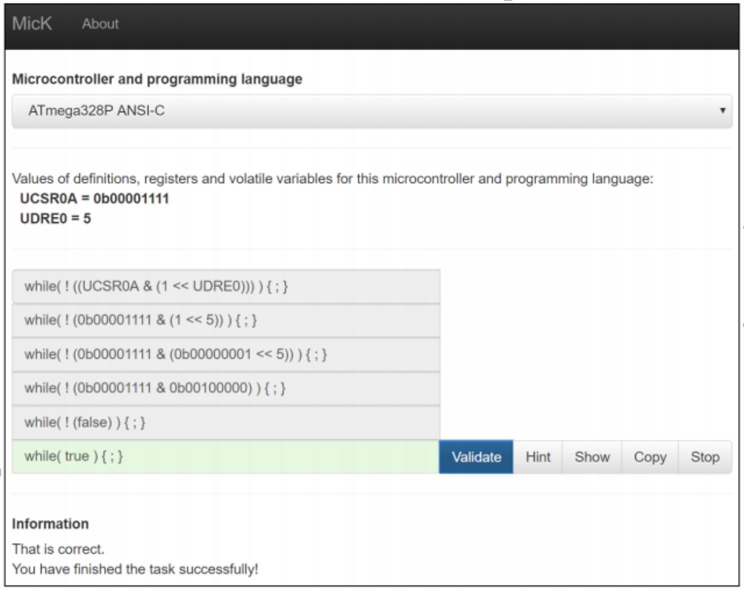
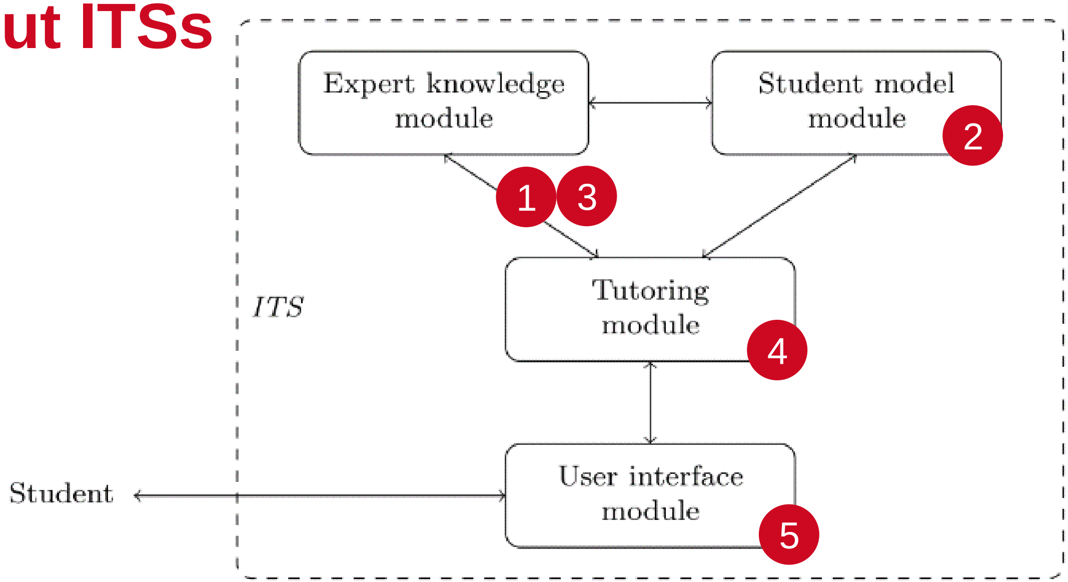




Figure 1: Constructing a square using circles and lines (screenshot from GeoGebra – a mathematics tool that allows drawing of geometric structures).




Tim Olmer (2014, TFPIE), Hieke Keuning (2014, CSERC), Stéphane Thibaud (2017), Hugo Arends (2017, Koli Calling)

# 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

Open Universiteit
www.ou.nl

# 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

Open Universiteit
www.ou.nl

Bastiaan.Heeren@ou.nl

# 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/

Open Universiteit
www.ou.nl