Universiteit Utrecht

[Faculty of Science
Information and Computing Sciences]

Ideas

Part 1: Procedural skills

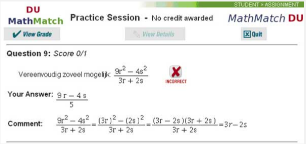
Johan Jeuring

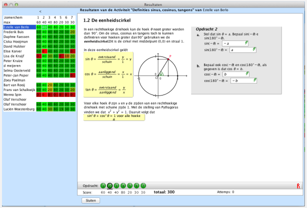
Utrecht University

Monday, September 7, 2016

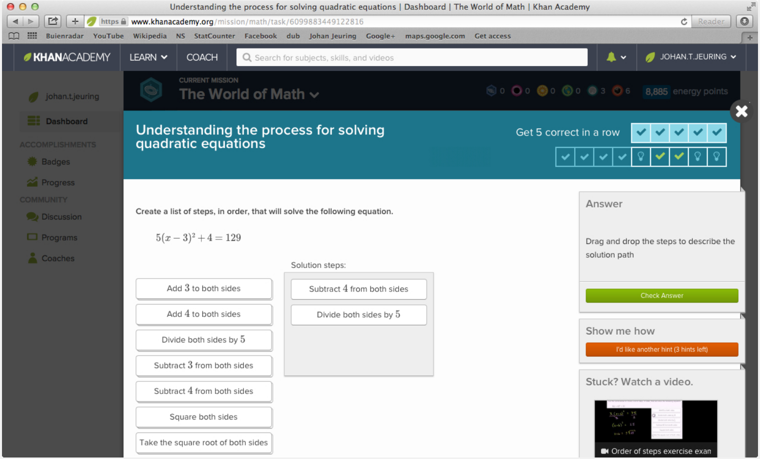
1. Introduction

ICT & Education



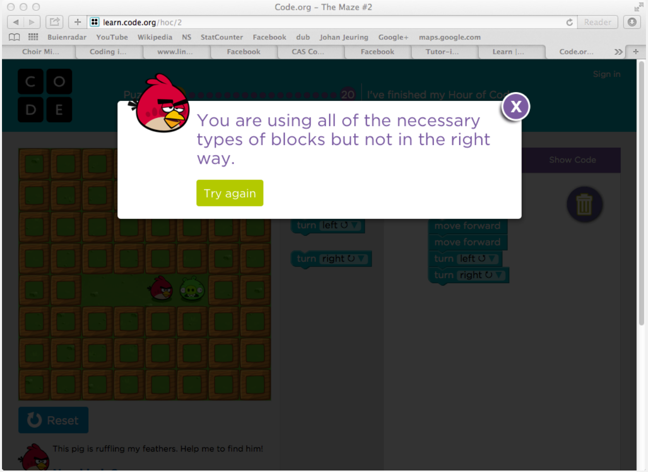


Free input?


Understanding the process for solving quadratic equations | Dashboard | The World of Math | Khan Academy
www.khanacademy.org/mission/math/task/6099883449122816
KHANACADEMY
LEARN
COACH
Search for subjects, skills, and videos
JOHAN.T.JEURING
johan.t.jeuring
Dashboard
ACCOMPLISHMENTS
Badges
Progress
COMMUNITY
Discussion
Programs
Coaches
CURRENT MISSION
The World of Math
8,885 energy points
Understanding the process for solving quadratic equations
Get 5 correct in a row
Create a list of steps, in order, that will solve the following equation.
5(x − 3)² + 4 = 129
Add 3 to both sides
Add 4 to both sides
Divide both sides by 5
Subtract 3 from both sides
Subtract 4 from both sides
Square both sides
Take the square root of both sides
Solution steps:
Subtract 4 from both sides
Divide both sides by 5
Answer
Drag and drop the steps to describe the solution path
Check Answer
Show me how
I'd like another hint (3 hints left)
Stuck? Watch a video.
Order of steps exercise exan

Quality of feedback?

`http://studio.code.org/hoc/2`



Help!


Enercities
www.enercities.eu/game.php
Buienradar YouTube Wikipedia NS StatCounter Facebook dub Johan Jeuring Google+ maps.google.com
Haskell Expre...
Ask–Elle
Ideas – LogEX
Rewriting qua...
Digitale Wisku...
Biology – PhE...
Enercities
EnerCities
Play Game
Highscores
Community
Teachers
Username
Password
Register
Login
2090
Score
161
55/100
Level 3
Project Page

Problems

- Simplified tasks
- Bad feedback
- No feedback

Goal

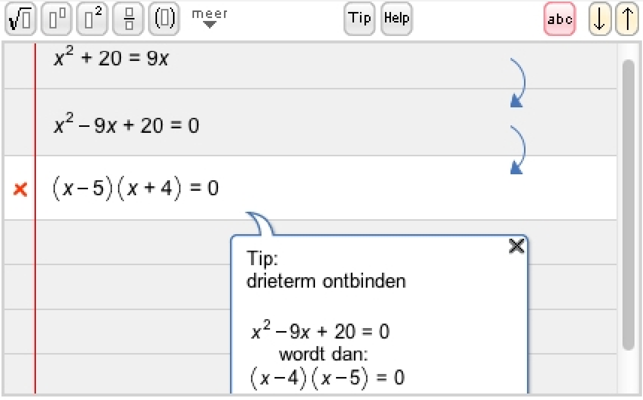
Use

- languages and grammars
- algebra's

To

- determine what a student has done
- determine what a student should do
- explain instead of show why a student performs badly

Resulting in



Outline of presentation

2. Procedural skills

Procedural skills

In many subjects students have to acquire procedural skills:

- Mathematics: find the derivative of a function
- Linear Algebra: solve a system of linear equations
- Logic: rewrite a proposition to disjunctive normal form
- Computer Science: construct a program from a specification using Dijkstra's calculus
- Physics: calculate the resistance of a circuit
- Biology: calculate inheritance values using Mendel's laws
- ...

Example

Theorie B

Het oplossen van kwadratische vergelijkingen

Om de vergelijking $x^2 - 7x - 18 = 0$ op te lossen, ontbind je eerst het linkerlid in factoren.

Vervolgens pas je toe $\boxed{A \cdot B = 0 \text{ geeft } A = 0 \vee B = 0.}$

Het teken $\vee$ betekent of.

Je krijgt

$x^2 - 7x - 18 = 0$ *Ontbind in factoren.*

$(x-9)(x+2) = 0$ *Pas toe* $A \cdot B = 0$ *geeft* $A = 0 \vee B = 0.$

$x - 9 = 0 \vee x + 2 = 0$

$x = 9 \vee x = -2$

Bij het oplossen van een kwadratische vergelijking gebruik je het volgende werkschema.

Werkschema: zo los je een kwadratische vergelijking op

1 Maak het rechterlid nul.

2 Ontbind het linkerlid in factoren.

3 Gebruik: uit $A \cdot B = 0$ volgt $A = 0 \vee B = 0.$

Tutoring tools for procedural skills

- Typical features of these tools:
  - Generate exercises
  - Stepwise construction of a solution
  - Select rewriting rule or transformation
  - Suggest how to continue
  - Check correctness of a step/solution
- Such tools offer many advantages to users:
  - User can work at any time
  - User can select material and exercises
  - Tool can select exercises based on a user-profile
  - Mistakes can be logged, and reported back to teachers
  - Tool can give immediate feedback

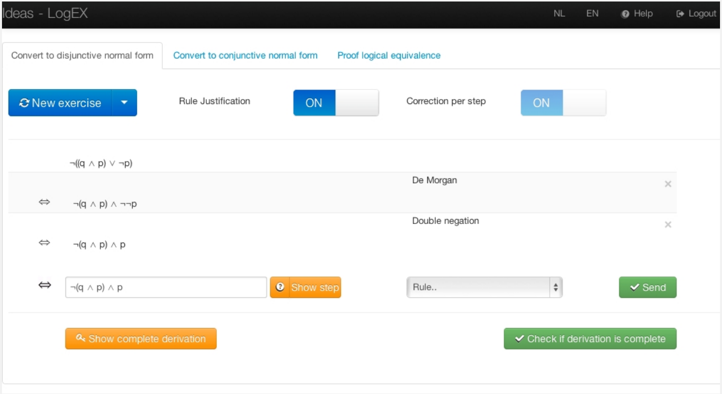
Do they work?

- Tutoring systems
- Serious games

3. Strategy specification language

Logex

http://ideas.cs.uu.nl/logex/



Rewriting to disjunctive normal form

- Rewrite rules for logical propositions:

$$\neg\neg\phi \Rightarrow \phi \qquad \phi \wedge (\psi \vee \chi) \Rightarrow (\phi \wedge \psi) \vee (\phi \wedge \chi)$$

$$\neg(\phi \wedge \psi) \Rightarrow \neg\phi \vee \neg\psi \qquad (\phi \vee \psi) \wedge \chi \Rightarrow (\phi \wedge \chi) \vee (\psi \wedge \chi)$$

$$\neg(\phi \vee \psi) \Rightarrow \neg\phi \wedge \neg\psi$$

- Exercise: bring $\neg(\neg(p \vee q) \wedge r)$ to DNF

Rewriting to disjunctive normal form

- Rewrite rules for logical propositions:

$$\neg\neg\phi \Rightarrow \phi \qquad \phi \wedge (\psi \vee \chi) \Rightarrow (\phi \wedge \psi) \vee (\phi \wedge \chi)$$
$$\neg(\phi \wedge \psi) \Rightarrow \neg\phi \vee \neg\psi \qquad (\phi \vee \psi) \wedge \chi \Rightarrow (\phi \wedge \chi) \vee (\psi \wedge \chi)$$
$$\neg(\phi \vee \psi) \Rightarrow \neg\phi \wedge \neg\psi$$

- Exercise: bring $\neg(\neg(p \vee q) \wedge r)$ to DNF

$$\begin{aligned} & \neg(\neg(p \vee q) \wedge r) \\ \Rightarrow\quad & \neg\neg(p \vee q) \vee \neg r \\ \Rightarrow\quad & p \vee q \vee \neg r \end{aligned}$$

Rewriting to disjunctive normal form

- Rewrite rules for logical propositions:

$$\neg\neg\phi \Rightarrow \phi \qquad \phi \wedge (\psi \vee \chi) \Rightarrow (\phi \wedge \psi) \vee (\phi \wedge \chi)$$
$$\neg(\phi \wedge \psi) \Rightarrow \neg\phi \vee \neg\psi \qquad (\phi \vee \psi) \wedge \chi \Rightarrow (\phi \wedge \chi) \vee (\psi \wedge \chi)$$
$$\neg(\phi \vee \psi) \Rightarrow \neg\phi \wedge \neg\psi$$

- Exercise: bring $\neg(\neg(p \vee q) \wedge r)$ to DNF

$$\begin{array}{ll} & \neg(\neg(p \vee q) \wedge r) \\ \Rightarrow & \neg\neg(p \vee q) \vee \neg r \\ \Rightarrow & p \vee q \vee \neg r \end{array}$$

$$\begin{array}{ll} & \neg(\neg(p \vee q) \wedge r) \\ \Rightarrow & \neg((\neg p \wedge \neg q) \wedge r) \\ \Rightarrow & \neg(\neg p \wedge \neg q) \vee \neg r \\ \Rightarrow & \neg\neg p \vee \neg\neg q \vee \neg r \\ \Rightarrow & p \vee \neg\neg q \vee \neg r \\ \Rightarrow & p \vee q \vee \neg r \end{array}$$

Strategies for reaching DNF

- Naive strategy:

  Apply rewrite rules exhaustively

Strategies for reaching DNF

- Naive strategy:

  Apply rewrite rules exhaustively

- Algorithmic strategy:

  (1) Remove constants
  (2) Unfold definitions of implication/equivalence
  (3) Push negations inside (top-down)
  (4) Then use the distribution rule

Strategies for reaching DNF

- Naive strategy:

  Apply rewrite rules exhaustively

- Algorithmic strategy:

  (1) Remove constants
  (2) Unfold definitions of implication/equivalence
  (3) Push negations inside (top-down)
  (4) Then use the distribution rule

- Expert strategy:

  Apply the algorithmic strategy, but use rules for tautologies and contradictions whenever possible

Modelling intelligence I

To model intelligence in a computer program, Bundy (*The Computer Modelling of Mathematical Reasoning*, 1983) identifies three important, basic needs:

1. The need to have knowledge about the domain
2. The need to reason with that knowledge
3. The need for knowledge about how to direct or guide that reasoning

Modelling intelligence I

To model intelligence in a computer program, Bundy (*The Computer Modelling of Mathematical Reasoning*, 1983) identifies three important, basic needs:

1. The need to have knowledge about the domain
2. The need to reason with that knowledge
3. The need for knowledge about how to direct or guide that reasoning

In our running example:

1. The domain consists of logical propositions
2. Reasoning uses rewrite rules for logical propositions
3. Strategies guide that reasoning

Modelling intelligence II

- Strategies can be used for any kind of procedural activities (not just maths)
- Alternatives: ACT-R (next week), CTAT (Cognitive Tutor Authoring Tools), Andes, many more
- Strategies are a declarative and compositional alternative

Our running example in ACT-R

1. The domain consists of logical propositions
2. Reasoning uses production rules for logical propositions
3. Reasoning is implemented by an interpreter which chooses which productions to fire. There is no explicit representation of complex cognitive skills

A strategy specification language

We need the following concepts for specifying a strategy:

- apply a basic rewrite rule *("$\wedge$ distributes over $\vee$")*
- sequence *("first . . . then . . . ")*
- choice *("use one of the rules for $\neg$")*
- apply exhaustively *("repeat . . . as long as possible")*
- traversals *("apply . . . top down")*

The same concepts are found in:

- (program) transformation languages
- proof plans and tacticals
- workflow languages

Strategy composition

- Basic strategy combinators:

1. Sequence $s \mathbin{<\star>} t$
2. Choice $s \mathbin{<|>} t$
3. Unit elements $succeed$, $fail$
4. Labels $label\ \ell\ s$
5. Recursion $fix\ f$

Strategy composition

- Basic strategy combinators:

1. Sequence	$s <\star> t$
2. Choice	$s <\|> t$
3. Unit elements	$succeed$, $fail$
4. Labels	$label\ \ell\ s$
5. Recursion	$fix\ f$

- Many more combinators can be added:

$$option\ s = s <|> succeed$$

$$many\ \ s = fix\ (\lambda x \rightarrow option\ (s <\star> x))$$

$$repeat\ s = many\ s <\star> not\ s$$

4. Feedback services

Calculating feedback automatically

With a strategy, we can calculate several kinds of feedback:

- Feedback after a step by a user
- Hints on how to continue
- Worked-out solutions
- Strategy unfolding (problem decomposition)
- Completion problems
- Progress (number of steps remaining)
- Report common mistakes

- Most categories appear in the tutoring principles of Anderson
- Offered as (web-)services to other learning environments

Reporting common mistakes

- Formulate misconceptions as buggy rules:

$$\neg(\phi \land \psi) \not\Rightarrow \neg\phi \land \neg\psi$$
$$\phi \land (\psi \lor \chi) \not\Rightarrow (\phi \land \psi) \lor \chi$$

- Buggy rules can be recognized and reported with a specialized feedback text
- Also: buggy strategies to describe procedural mistakes

Strategy unfolding

- Strategies have a hierarchical structure
- Use structure to decompose an exercise
  - First ask for the final answer
  - If the answer is incorrect, decompose the problem into subparts and let the user try again
  - Example from linear algebra: split the Gaussian Elimination method into a forward and a backward pass
- The structure of a strategy and its labels also provide a way to adapt and customize the strategy

How feedback is calculated

The main idea:

- A strategy describes valid sequences of rules
- View a strategy specification as a context-free grammar
- This turns tracking intermediate steps into a parsing problem

How feedback is calculated

The main idea:

- A strategy describes valid sequences of rules
- View a strategy specification as a context-free grammar
- This turns tracking intermediate steps into a parsing problem

Feedback service	Parsing problem
ready	is the empty sentence (ϵ) accepted?
provide hint	compute the “first set”
worked-out solution	construct a sentence
after a step	try to recognize the rewrite rule that was used, and parse this rule as the next symbol of the input

5. Application domains

Application domains

- Logic
- Mathematics
- Communication skills
- Infection and Immunology
- Programming

5.1 Logic

Proving equivalences

- Use strategies to prove the equivalence of logical propositions
- Allow student to make forward steps and backward steps

$$
\begin{aligned}
&\neg\,((p \to q) \to (p \wedge q)) \\
&\quad \Leftrightarrow \{\text{implication elimination}\} \\
&\neg\,(\neg\,(p \to q) \vee (p \wedge q)) \\
&\quad \Leftrightarrow \{\text{De Morgan}\} \\
&\neg\neg(p \to q) \wedge \neg\,(p \wedge q) \\
&\quad \Leftrightarrow \{\text{double negation}\} \\
&(p \to q) \wedge \neg\,(p \wedge q) \\
&\quad \Leftrightarrow \{\text{De Morgan}\} \\
&(p \to q) \wedge (\neg\,p \vee \neg\,q)
\end{aligned}
$$

Proving equivalences (how)

- The strategy rewrites a pair of propositions
- Rewrite both parts to disjunctive normal form, and then towards equal forms
- Two simple techniques simplify the generated proofs:
  - Try to decompose the proof into subproofs by inspecting the top-level operators
  - Search for common subformulas

$$\neg\,(\boxed{(p \rightarrow q)} \rightarrow (p \wedge q))$$

$$\Leftrightarrow \{\ldots\}$$

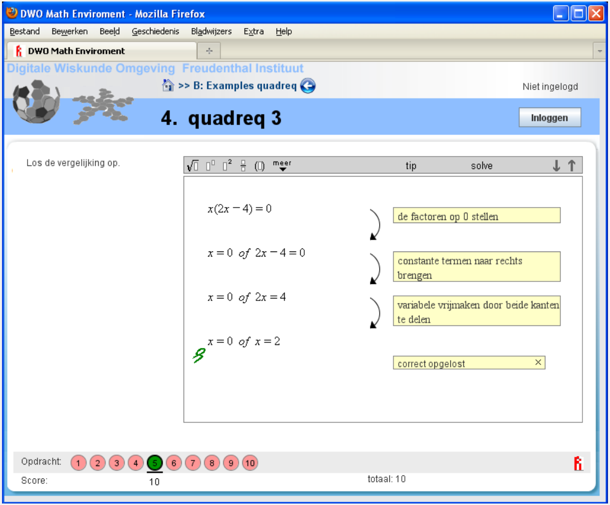
$$\boxed{(p \rightarrow q)} \wedge (\neg\, p \vee \neg\, q)$$

5.2 Mathematics

Mathematics

- We collaborate with the Freudenthal Institute to extend their applets with our feedback facilities
  - Covers most topics in secondary school mathematics: polynomial equations, inequalities, calculating with powers, derivatives, etc.
  - Applets are used by many schools (and a popular textbook)
- We participated in the Math-Bridge project
  - Large European consortium around the ActiveMath learning environment
  - Aims at providing a math bridging course to higher education
- We try to apply our approach to different types of exercises

DWO Math Environment (with feedback)



Tool by Peter Boon (Freudenthal Institute)

Challenges in a math tutor

- Support for canonical forms
  - To test for equality
  - To control the granularity of steps
  - To simplify terms

Examples:

- $2\sqrt{2}$ versus $\sqrt{8}$, $3\frac{1}{2}$ versus $\frac{7}{2}$ (or even 3.5)
- $x + (-3)$ versus $x - 3$
- pattern $ax + b$ versus $3 - 5x$

- Flexibility in strategies (customization)
- Parameterized rewrite steps (“divide both sides by 5”)

What does a step look like?

$$3 * (4 * x - 1) + 3 = 7 * x - 14 \Rightarrow 12 * x = 7 * x - 14?$$

You are doing a lot in this step!

What does a step look like?

$$3 * (4 * x - 1) + 3 = 7 * x - 14 \Rightarrow 12 * x = 7 * x - 14?$$

You are doing a lot in this step!

$$3 * (4 * x - 1) + 3$$

What does a step look like?

$$3 * (4 * x - 1) + 3 = 7 * x - 14 \Rightarrow 12 * x = 7 * x - 14?$$

You are doing a lot in this step!

$$\begin{array}{ll} & 3 * (4 * x - 1) + 3 \\ \Rightarrow & (3 * 4 * x - 3 * 1) + 3 \end{array}$$

What does a step look like?

$$3 * (4 * x - 1) + 3 = 7 * x - 14 \Rightarrow 12 * x = 7 * x - 14?$$

You are doing a lot in this step!

$$\begin{array}{ll} & 3 * (4 * x - 1) + 3 \\ \Rightarrow & (3 * 4 * x - 3 * 1) + 3 \\ \Rightarrow & (12 * x - 3 * 1) + 3 \end{array}$$

What does a step look like?

$$3 * (4 * x - 1) + 3 = 7 * x - 14 \Rightarrow 12 * x = 7 * x - 14?$$

You are doing a lot in this step!

$$
\begin{array}{ll}
 & 3 * (4 * x - 1) + 3 \\
\Rightarrow & (3 * 4 * x - 3 * 1) + 3 \\
\Rightarrow & (12 * x - 3 * 1) + 3 \\
\Rightarrow & (12 * x - 3) + 3
\end{array}
$$

What does a step look like?

$$3 * (4 * x - 1) + 3 = 7 * x - 14 \Rightarrow 12 * x = 7 * x - 14?$$

You are doing a lot in this step!

$$\begin{array}{ll} & 3 * (4 * x - 1) + 3 \\ \Rightarrow & (3 * 4 * x - 3 * 1) + 3 \\ \Rightarrow & (12 * x - 3 * 1) + 3 \\ \Rightarrow & (12 * x - 3) + 3 \\ \Rightarrow & (12 * x + (-3)) + 3 \end{array}$$

What does a step look like?

$$3 * (4 * x - 1) + 3 = 7 * x - 14 \Rightarrow 12 * x = 7 * x - 14?$$

You are doing a lot in this step!

$$\begin{array}{ll} & 3 * (4 * x - 1) + 3 \\ \Rightarrow & (3 * 4 * x - 3 * 1) + 3 \\ \Rightarrow & (12 * x - 3 * 1) + 3 \\ \Rightarrow & (12 * x - 3) + 3 \\ \Rightarrow & (12 * x + (-3)) + 3 \\ \Rightarrow & 12 * x + (-3 + 3) \end{array}$$

What does a step look like?

$$3 * (4 * x - 1) + 3 = 7 * x - 14 \Rightarrow 12 * x = 7 * x - 14?$$

You are doing a lot in this step!

$$\begin{array}{ll} & 3 * (4 * x - 1) + 3 \\ \Rightarrow & (3 * 4 * x - 3 * 1) + 3 \\ \Rightarrow & (12 * x - 3 * 1) + 3 \\ \Rightarrow & (12 * x - 3) + 3 \\ \Rightarrow & (12 * x + (-3)) + 3 \\ \Rightarrow & 12 * x + (-3 + 3) \\ \Rightarrow & 12 * x + 0 \end{array}$$

What does a step look like?

$$3 * (4 * x - 1) + 3 = 7 * x - 14 \Rightarrow 12 * x = 7 * x - 14?$$

You are doing a lot in this step!

$$
\begin{array}{ll}
 & 3 * (4 * x - 1) + 3 \\
\Rightarrow & (3 * 4 * x - 3 * 1) + 3 \\
\Rightarrow & (12 * x - 3 * 1) + 3 \\
\Rightarrow & (12 * x - 3) + 3 \\
\Rightarrow & (12 * x + (-3)) + 3 \\
\Rightarrow & 12 * x + (-3 + 3) \\
\Rightarrow & 12 * x + 0 \\
\Rightarrow & 12 * x
\end{array}
$$

Similar problems

- Economy of rules: I want to describe

$$a * (b + c) \Rightarrow a * b + a * c$$

but preferably not also:

$$a * (b - c) \Rightarrow a * b - a * c$$
$$-a * (b + c) \Rightarrow -a * b - a * c$$

- Canonical forms: $a + (-b)$ should be presented as $a - b$
- Granularity: users at different levels need different granularity of rules
- Recognizing user steps: when showing steps to users, we want to apply some simplifications automatically. When recognising steps, however, such simplifications are not obligatory

Views

A view views an expression in a particular format:

- a match function returns an equivalent value in a different format, for example:

$$\begin{array}{lcl} \mathit{match}\ \mathit{plusView}\ (a - b) & \Rightarrow & a + (-b) \\ \mathit{match}\ \mathit{plusView}\ (-(a + b)) & \Rightarrow & -a + -b \end{array}$$

- a build function to return to the original domain, for example:

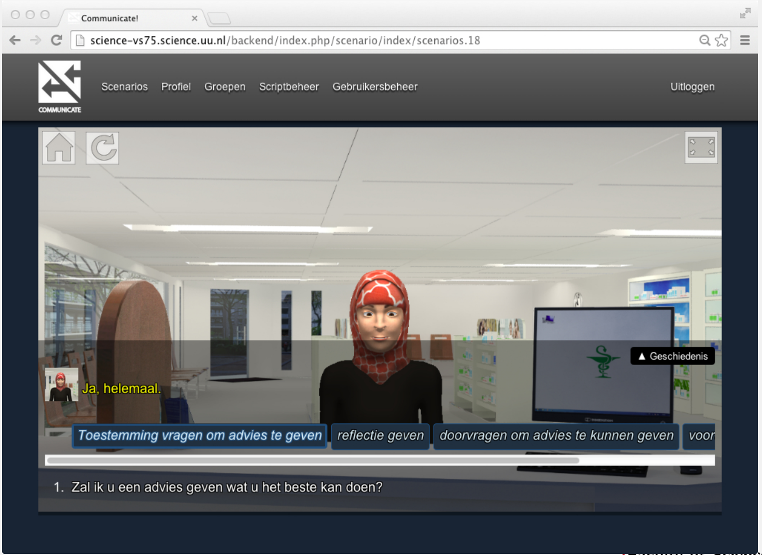
$$\begin{array}{ll} & 3 * (4 * x - 1) \\ \Rightarrow & \{\ \mathit{match}\ \mathit{plusView}\ \text{on}\ 4 * x - 1\ \} \\ & 3 * (4 * x + (-1)) \\ \Rightarrow & \{\ \text{distribute} * \text{over} +\ \} \\ & 3 * 4 * x + 3 * (-1) \\ \Rightarrow & \{\ \text{simplify using}\ \mathit{rationalView}\ \} \\ & 12 * x - 3 \end{array}$$

Views and rules

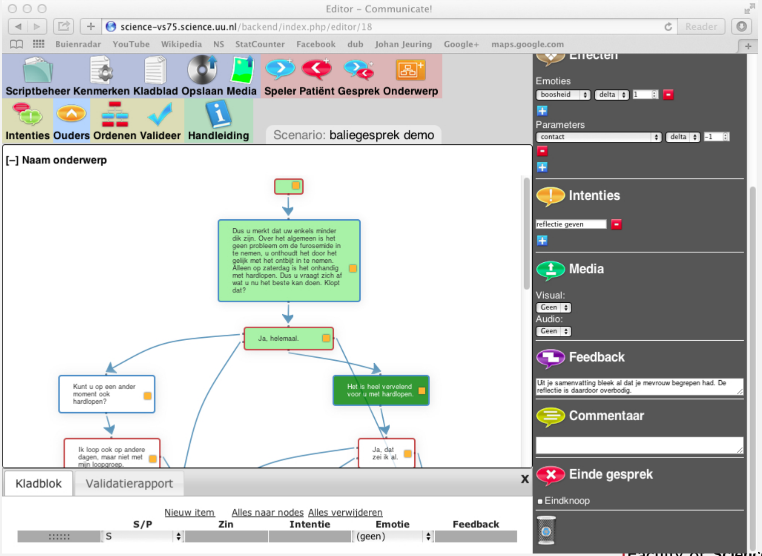
- Many rules use one or more views for matching on the left-hand side
- Many rules use one or more views to clean up a result expression after rewriting
- Views and parametrized rules solve the problem of making all steps in solving an exercise explicit

5.3 Serious games

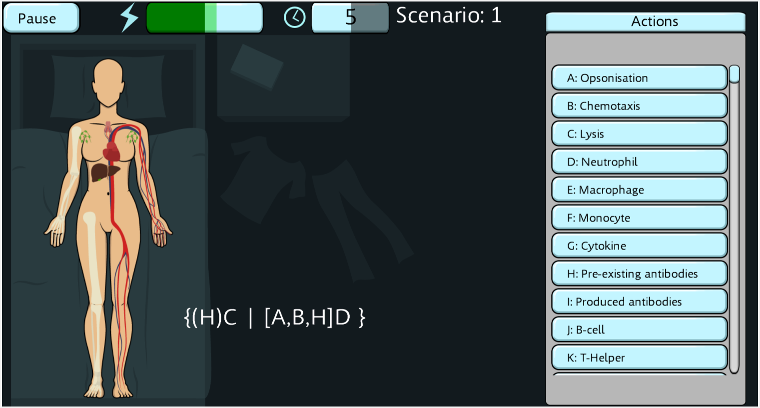
A communication skills game



Editing scenario's



An infection and immunity game



5.4 Programming

Programming

We have developed programming tutors for

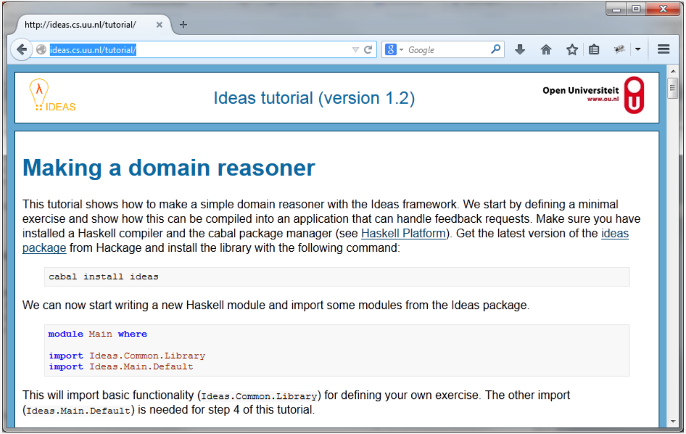
- Evaluating functional expressions
- Learning functional programming
- Learning imperative programming

More about this in the third lecture about ideas.

6. Lab assignment

Ideas tutorial

Visit `http://ideas.cs.uu.nl/tutorial/`



Ideas tutorial (version 1.2)

Making a domain reasoner

This tutorial shows how to make a simple domain reasoner with the Ideas framework. We start by defining a minimal exercise and show how this can be compiled into an application that can handle feedback requests. Make sure you have installed a Haskell compiler and the cabal package manager (see Haskell Platform). Get the latest version of the ideas package from Hackage and install the library with the following command:

```
cabal install ideas
```

We can now start writing a new Haskell module and import some modules from the Ideas package.

```
module Main where

import Ideas.Common.Library
import Ideas.Main.Default
```

This will import basic functionality (`Ideas.Common.Library`) for defining your own exercise. The other import (`Ideas.Main.Default`) is needed for step 4 of this tutorial.

Lab assignment

Start version, see `http://ideas.cs.uu.nl/tutorial/`, has,

- Simple arithmetic expression language
- Two evaluation rules

$$\textbf{data}\ Expr = Add\ Expr\ Expr \mid Negate\ Expr \mid Con\ Int$$

1. Add multiplication to the expression language (and extend the evaluation strategy)
2. Add distribution rules to the strategy
3. Add support for calculating with fractions (e.g. $\frac{5}{7} + \frac{1}{2}$)
   - Find the least common multiple of the denominators
   - Rewrite top-heavy fractions to mixed fractions (e.g. $1\frac{3}{14}$)

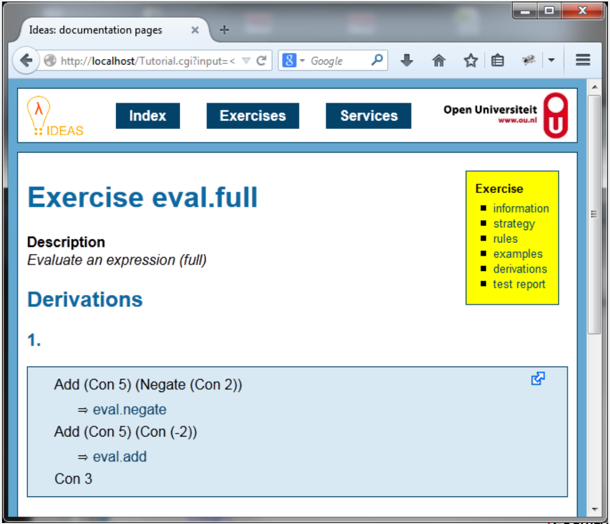
About the Ideas framework

- Latest release: version 1.5 (May 2016)
- Over 10,000 lines of Haskell code (in 110 modules)
- `http://hackage.haskell.org/package/ideas`

How to interact with a domain reasoner?

- Develop a client that calls the (server/cgi) domain reasoner
- Use the Haskell interpreter (ghci)
- Compile to a cgi binary (with support for HTML) and deploy on your localhost; use a browser
- Compile and send a request from the command-line (file)

Domain reasoner in browser



More information

Bastiaan Heeren and Johan Jeuring. Feedback services for stepwise exercises. Science of Computer Programming Special Issue on Software Development Concerns in the e-Learning Domain, volume 88, 110 - 129, 2014.

Bastiaan Heeren, Johan Jeuring, and Alex Gerdes. Specifying rewrite strategies for interactive exercises. In Mathematics in Computer Science 3(3), 349 - 370, 2010.

- You can discuss the lab amongst each other, but you cannot reuse code from somebody else
- Hand in your solution via email to me on or before 28/9

7. Concluding remarks

Concluding remarks

- We introduced a strategy language to make the procedure for solving an exercise explicit
- This language is what differentiates us from other tools
- Feedback is calculated from the strategy by turning feedback services into parsing problems
- Strategies can be used in many learning tools