



## Introduction

### Procedural skills

In many subjects students have to acquire procedural skills:

- ▶ Mathematics:
  - ▷ calculate the value an expression
  - ▷ differentiate a function
- ▶ Logic:
  - ▷ rewrite an expression to disjunctive normal form

### Our research

We use strategies to specify how a wide range of exercises can be solved incrementally (stepwise), such as reducing a matrix, or calculating with fractions. In contrast to other e-learning systems, we can calculate all kinds of feedback automatically from a strategy specification. It becomes less labor-intensive and less ad-hoc to specify new exercise domains and exercises within that domain.

### What kind of feedback?

With a strategy, we can automatically generate worked-out solutions, track the progress of a student by inspecting submitted intermediate answers, report back suggestions in case the student deviates from the strategy, and much more.

## Strategy language

Strategies are constructed using the following components:

- ▶ Transformation rules. Basic rewrite steps, e.g. 'renaming'
- ▶ Combinators:
  - ▷ sequence ("first ... then ...")
  - ▷ choice ("either this or that")
  - ▷ apply exhaustively ("repeat ... as long as possible")
  - ▷ traversals ("apply ... top down")
  - ▷ ...
- ▶ Labels. Used to specify feedback

We can generate feedback on the syntax, rule, and strategy level. The user of the generated feedback is, however, in charge of its deployment.

Conceptually we view the strategy as a grammar. This means that solving an exercise is now constructing a sentence.

## Conclusions

### Current status

- ▶ A library for specifying strategies, <http://ideas.cs.uu.nl/>
- ▶ Tested on linear algebra, propositional logic, arithmetic, equations, relation algebra, and introductory programming exercises
- ▶ The use of our feedback leads to a lower mental effort and a better far transfer
- ▶ Automated assessment of programming exercises based on programming strategies
- ▶ Already used in courses (to a limited extent)
- ▶ Available as web services
- ▶ Further developed in the context of Surf and EU eContentPlus projects
- ▶ Our tools will be used in software that accompanies several Dutch high-school mathematics text books, such as Getal & Ruimte

### Future work

- ▶ Develop a strategy framework for high-school mathematics
- ▶ Generate semantically rich feedback for programming exercises

## Example: adding fractions

### Example

Consider the problem of adding two fractions: if the result is an improper fraction, then it should be converted to a mixed number. We define a possible strategy to solve this type of exercise:

- Step 1. Find the least common denominator (LCD) of the fractions: let this be  $n$
- Step 2. Rename the fractions such that  $n$  is the denominator
- Step 3. Add the fractions by adding the numerators
- Step 4. Simplify the fraction if it is improper

### Derivation

A possible derivation of adding two fractions:

$$\begin{aligned} & \frac{4}{5} + \frac{2}{3} \\ &= \left\{ \text{rename : } \frac{b}{c} = \frac{ab}{ac} \right\} \\ & \frac{12}{15} + \frac{2}{3} \\ &= \left\{ \text{rename : } \frac{b}{c} = \frac{ab}{ac} \right\} \\ & \frac{12}{15} + \frac{10}{15} \\ &= \left\{ \text{add : } \frac{a}{c} + \frac{b}{c} = \frac{a+b}{c} \right\} \\ & \frac{22}{15} \\ &= \left\{ \text{simplify : } \frac{a+b}{b} = 1\frac{a}{b} \right\} \\ & 1\frac{7}{15} \end{aligned}$$

### Feedback

Our tool gives feedback when a buggy rule has been applied:

$$\begin{aligned} & \frac{1}{2} + \frac{1}{3} \\ &= \left\{ \text{buggy add : } \frac{a}{b} + \frac{c}{d} \neq \frac{a+c}{b+d} \right\} \\ & \frac{2}{5} \end{aligned}$$

We also provide strategy feedback. Although correct, the following rewrite step does not lead to an efficient solution:

$$\begin{aligned} & \frac{2}{5} + \frac{1}{4} \\ &= \left\{ \text{rename : } \frac{b}{c} = \frac{ab}{ac} \right\} \\ & \frac{2}{5} + \frac{2}{8} \end{aligned}$$

