An extensible strategy language for describing cognitive skills

Bastiaan Heeren[1] Johan Jeuring[1,2]

[1] Open University of the Netherlands
[2] Utrecht University

May 5, 2017, Chalmers, Göteborg

Intelligent tutoring system (ITS)

- Problem-solving procedures (cognitive skills/strategies) can be found in many domains:
  - Solving equations (mathematics)
  - Constructing programs (computer science)
  - Practicing communication skills (e.g. pharmacy)
  - ...
- ITSs can help students to practice such tasks
- ITSs are almost as effective as human tutors (VanLehn, 2011)
- ITSs have an inner loop for solving tasks step by step

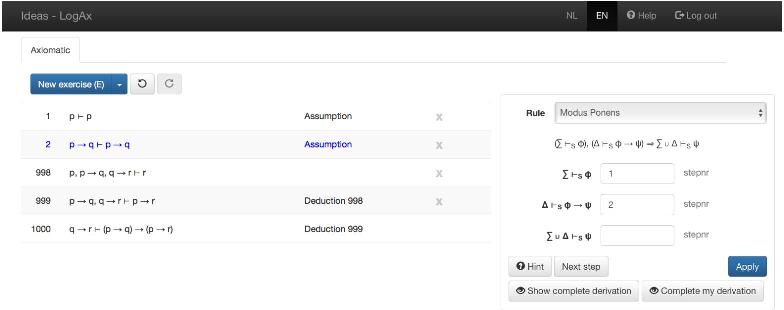
Intelligent tutoring system (ITS)

- Problem-solving procedures (cognitive skills/strategies) can be found in many domains:
  - Solving equations (mathematics)
  - Constructing programs (computer science)
  - Practicing communication skills (e.g. pharmacy)
  - ...
- ITSs can help students to practice such tasks
- ITSs are almost as effective as human tutors (VanLehn, 2011)
- ITSs have an inner loop for solving tasks step by step

How can we specify problem-solving procedures and automatically calculate feedback and hints?

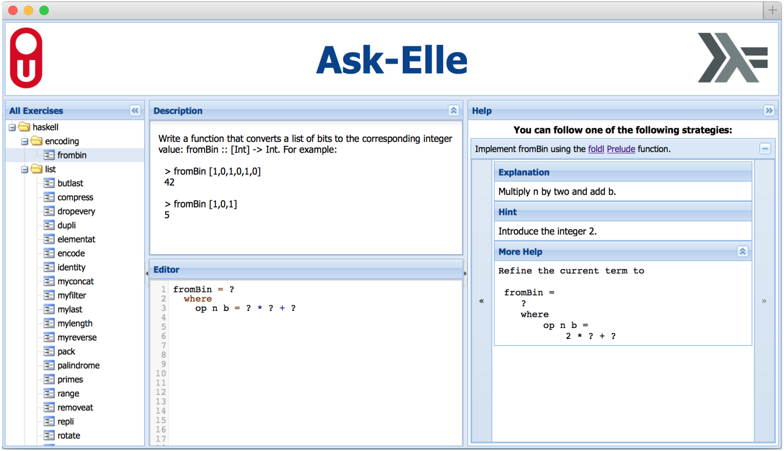
$\Rightarrow$ *We define an extensible strategy language (DSL).*

Axiomatic proofs (Lodder et al.)



- Construct proofs by applying rules (forward and backward)
- Feedback after each step (also for common mistakes)
- Hints and worked-out solutions available

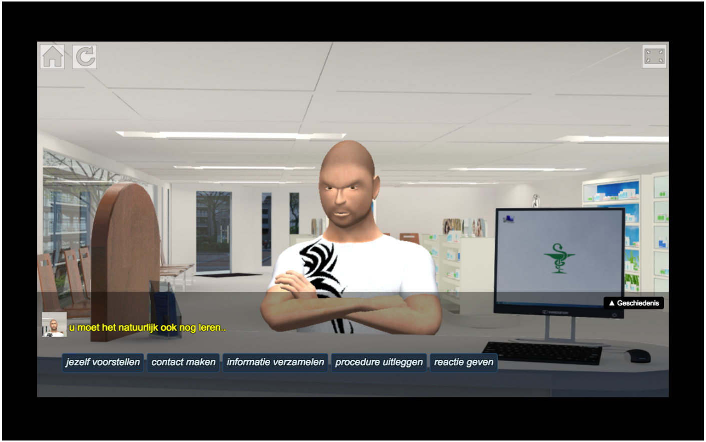
Functional programming tutor (Gerdes et al.) §1



- Develop programs by step-wise refining holes (?)
- Feedback and hints calculated from model solutions

[An extensible strategy language for describing cognitive skills]

Communicate! serious game (Jeuring et al.)



- Game for practicing interpersonal communication skills
- Final score and feedback afterwards

Example: adding fractions

Problem-solving procedure

1. Find lowest common denominator (LCD)
2. Convert fractions to LCD as denominator
3. Add resulting fractions
4. Simplify final result

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

Example: adding fractions

Problem-solving procedure

1. Find lowest common denominator (LCD)
2. Convert fractions to LCD as denominator
3. Add resulting fractions
4. Simplify final result

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

Procedure specified as a strategy:

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

Strategy language

What are the requirements for the strategy language?

1. Universal: not for one particular domain (reusable)
2. Extensible: easy to extend language with new patterns
3. Feedback and hints: should be available at any time
4. Compositional: combine simple procedures into more complex procedures
5. Adaptable: possible to customize procedures
6. Efficient: hints and feedback can be calculated in a reasonable amount of time

The strategy language needs a rigorous semantics

Core grammar

- Starting point: a minimal language
  - Support for choice: $<\!\!|\!\!>$
  - Left-hand side of prefix ($\rightarrow$) is restricted to rules (r)

$$s, t \;::=\; \textit{succeed} \;\mid\; \textit{fail} \;\mid\; s <\!\!|\!\!> t \;\mid\; r \rightarrow s$$

Core grammar

- Starting point: a minimal language
  - Support for choice: $<\!|\!>$
  - Left-hand side of prefix ($\rightarrow$) is restricted to rules (r)

$$s, t \ ::= \ \mathit{succeed} \ \mid \ \mathit{fail} \ \mid \ s <\!|\!> t \ \mid \ r \rightarrow s$$

- Approach: define which traces are allowed by a strategy
- Trace set includes partial traces and unsuccessful traces
- Example of a successful trace:

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

Semantics: empty and firsts

- *empty:* is the strategy (successfully) finished?

$$\begin{aligned}
\mathit{empty}(\mathit{succeed}) &= \mathit{true} \\
\mathit{empty}(\mathit{fail}) &= \mathit{false} \\
\mathit{empty}(s <\!|\!> t) &= \mathit{empty}(s) \vee \mathit{empty}(t) \\
\mathit{empty}(r \rightarrow s) &= \mathit{false}
\end{aligned}$$

- *firsts:* calculates which rules can be taken at this point, together with their remainders (finite map):

$$\begin{aligned}
\mathit{firsts}(\mathit{succeed}) &= \emptyset \\
\mathit{firsts}(\mathit{fail}) &= \emptyset \\
\mathit{firsts}(s <\!|\!> t) &= \mathit{firsts}(s) \uplus \mathit{firsts}(t) \\
\mathit{firsts}(r \rightarrow s) &= \{ r \mapsto s \}
\end{aligned}$$

Traces

- Not all rules suggested by *firsts* can be applied to current object a:

$$\mathit{steps}(s, a) = \{(r, t, b) \mid r \mapsto t \in \mathit{firsts}(s), b \in r(a)\}$$

- Calculate the set of traces:

$$\begin{aligned}\mathit{traces}(s, a) = \{a\} &\cup \{a \checkmark \mid \mathit{empty}(s)\} \\ &\cup \{a \xrightarrow{r} x \mid (r, t, b) \in \mathit{steps}(s, a), x \in \mathit{traces}(t, b)\}\end{aligned}$$

Equality

Two strategies are equal when their traces are equal:

$$(s = t) \ =_{def} \ \forall a : traces(s, a) = traces(t, a)$$

- With equality, we can formulate algebraic laws, e.g.:
  - Choice ($<\!|\!>$) is associative, and has *fail* as its unit element
  - Prefix ($\rightarrow$) is left-distributive over choice
- Laws help to reason about strategies
- Laws help to optimize strategies
- Laws help to extend the strategy language

Extension: sequential composition

- $s <\star> t$: first do s, then t
- Sequences can be compiled into the core language:

$$
\begin{aligned}
\mathit{succeed} \quad & <\star> t = t \\
\mathit{fail} \quad & <\star> t = \mathit{fail} \\
(s_1 <|> s_2) & <\star> t = (s_1 <\star> t) <|> (s_2 <\star> t) \\
(r \rightarrow s) \quad & <\star> t = r \rightarrow (s <\star> t)
\end{aligned}
$$

- New laws follow from this definition:
  - Sequence ($<\star>$) is associative, and has *succeed* as its unit element
  - Sequence distributes over choice

Extension: repetition

- Apply strategy s optionally, zero or more times, or one or more times:

$$\begin{aligned} \mathit{option}\ \ s &= s \mathbin{<\!|\!>} \mathit{succeed} \\ \mathit{many}\ \ s &= \mathit{option}\ (s \mathbin{<\!\star\!>} \mathit{many}\ s) \\ \mathit{many1}\ s &= s \mathbin{<\!\star\!>} \mathit{many}\ s \end{aligned}$$

- For *many* we need a fixed-point combinator
- Also: greedy variants for *option*, *many*, and *many1*

More extensions

- Traversal combinators: for domains with sub-terms
  - *somewhere*, *oncebu*, *innermost*, etc.
- Interleaving: switch between strategies, e.g.
  $\{ a_1\, a_2 \} <\%> \{ b_1 \} = \{ a_1\, a_2\, b_1,\ a_1\, b_1\, a_2,\ b_1\, a_1\, a_2 \}$
- Permutation
- Topological sorts: for re-ordering statements
  - Based on a program's data-flow graph
- Initial prefixes: allow a conversation to stop at any time
- Left-biased choice: do s, or else t
- Preference: prefer some traces (hints) over other traces

Conclusions

We presented a strategy language that:

- is compositional
- is extensible (with new patterns)
- has a precise semantics (with laws)
- works for many domains

- Traces can be used for generating feedback and hints
- Similar to other formalisms (CSP, rewriting systems), but specific for tools in education
- For more information, see the project websites:
  http://ideas.cs.uu.nl/
  http://ideastest.cs.uu.nl/