An extensible strategy language for describing cognitive skills

Bastiaan Heeren\textsuperscript{1} \quad Johan Jeuring\textsuperscript{1,2}

\textsuperscript{1} Open University of the Netherlands \quad \textsuperscript{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
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?

⇒ 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.)

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

\[
\frac{1}{2} + \frac{4}{5} \quad \xrightarrow{\text{Find LCD}} \quad \frac{1}{2} + \frac{4}{5} \quad \xrightarrow{\text{Convert}} \quad \frac{5}{10} + \frac{4}{5} \quad \xrightarrow{\text{Convert}} \quad \frac{5}{10} + \frac{8}{10} \quad \xrightarrow{\text{Add}} \quad \frac{13}{10} \quad \xrightarrow{\text{Simplify}} \quad 1 \frac{3}{10} \quad \checkmark
\]
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

\[
\frac{1}{2} + \frac{4}{5} \Rightarrow \frac{1}{2} + \frac{4}{5} \Rightarrow \frac{5}{10} + \frac{4}{5} \Rightarrow \frac{5}{10} + \frac{8}{10} \Rightarrow \frac{13}{10} \Rightarrow 1 \frac{3}{10} \checkmark
\]

Procedure specified as a strategy:

\text{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 ($r$)

$$s, t ::= succed | fail | s <> t | r \rightarrow s$$
Core grammar

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

$$s, t ::= \text{succeed} \mid \text{fail} \mid s \leftrightarrow t \mid r \rightarrow s$$

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

\[
\frac{1}{2} + \frac{4}{5} \Rightarrow \frac{1}{2} + \frac{4}{5} \Rightarrow \frac{5}{10} + \frac{4}{5} \Rightarrow \frac{5}{10} + \frac{8}{10} \Rightarrow \frac{13}{10} \Rightarrow 1 \frac{3}{10} \checkmark
\]
**Semantics: empty and firsts**

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

  \[
  \begin{align*}
  \text{empty}(\text{succeed}) &= \text{true} \\
  \text{empty}(\text{fail}) &= \text{false} \\
  \text{empty}(s \not<|> t) &= \text{empty}(s) \lor \text{empty}(t) \\
  \text{empty}(r \to s) &= \text{false}
  \end{align*}
  \]

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

  \[
  \begin{align*}
  \text{firsts}(\text{succeed}) &= \emptyset \\
  \text{firsts}(\text{fail}) &= \emptyset \\
  \text{firsts}(s \not<|> t) &= \text{firsts}(s) \uplus \text{firsts}(t) \\
  \text{firsts}(r \to s) &= \{ r \mapsto s \}
  \end{align*}
  \]
Traces

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

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

- Calculate the set of traces:

$$traces(s, a) = \{ a \} \cup \{ a \checkmark \mid empty(s) \} \cup \{ a \xrightarrow{r} x \mid (r, t, b) \in steps(s, a), x \in traces(t, b) \}$$
Equality

Two strategies are equal when their traces are equal:

\[( s = t ) =_{\text{def}} \forall a : \text{traces}(s, a) = \text{traces}(t, a)\]

- With equality, we can formulate algebraic laws, e.g.:
  - Choice \((\text{<>})\) is associative, and has \text{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

- $s ⨀ t$: first do $s$, then $t$
- Sequences can be compiled into the core language:

  \[
  \begin{align*}
  \text{succeed} & \quad ⨀ t = t \\
  \text{fail} & \quad ⨀ t = \text{fail} \\
  (s_1 ⨀ s_2) & \quad ⨀ t = (s_1 ⨀ t) ⨀ (s_2 ⨀ t) \\
  (r \rightarrow s) & \quad ⨀ t = r \rightarrow (s ⨀ t)
  \end{align*}
  \]

- New laws follow from this definition:
  - Sequence ($⨀$) 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:

  $\text{option } s = s \langle\rangle \text{ succeed}$

  $\text{many } s = \text{ option } (s \langle\rangle \text{ many } s)$

  $\text{many1 } s = s \langle\rangle \text{ many } s$

- For $\text{many}$ we need a fixed-point combinator

- Also: greedy variants for $\text{option}$, $\text{many}$, and $\text{many1}$
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/