An extensible domain-specific language for describing problem-solving procedures

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

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

July 1, AIED 2017, Wuhan

Problem-solving procedures

- Inner loop of ITS supports solving tasks step by step
  - inner loop services of ITSs are very similar (VanLehn 2006)
  - ... but their internal structures and representations are not
- Various approaches and paradigms for intelligent tutoring
  - model-tracing, constraint-based, data-driven, etc.

Problem-solving procedures

- Inner loop of ITS supports solving tasks step by step
  - inner loop services of ITSs are very similar (VanLehn 2006)
  - . . . but their internal structures and representations are not
- Various approaches and paradigms for intelligent tutoring
  - model-tracing, constraint-based, data-driven, etc.

- Many domains have problem-solving procedures expressing how to solve a task by applying rules in a controlled way
  - procedures can be used for providing hints and feedback

$\frac{1}{2}x - 4 = \frac{1}{4}(x-3)$	*multiply by 4*
$2x - 16 = x - 3$	*variable x to the left*
$x - 16 = -3$	*constants to the right*
$x = 13$	✓

Contributions

1. We evaluate how problem-solving procedures are specified in ITS paradigms, based on reported design principles

   These principles include:

   - explicit knowledge representation for procedures
   - representation should be modular and reusable

2. We propose an extensible domain-specific language (DSL) for describing problem-solving procedures

   - the DSL provides a rich vocabulary for common patterns
   - we discuss how the DSL has been used for different task domains

Design principles for inner loop

We collected 17 principles from five papers: Anderson et al. (1995), Beeson (1998), Murray (1998), Murray (2003), and Aleven et al. (2009).

Beeson's principles for MathPert (algebra and calculus tutor)

- cognitive fidelity
- glass box computation
- customize step size to individual user

Design principles for inner loop

We collected 17 principles from five papers: Anderson et al. (1995), Beeson (1998), Murray (1998), Murray (2003), and Aleven et al. (2009).

Beeson's principles for MathPert (algebra and calculus tutor)

- cognitive fidelity
- glass box computation
- customize step size to individual user

Important qualities:

- Expressive representation: repetitive and template-like content should be avoided
- Customizable/extensible: for example, change step size
- Cost-effective: proven tactics are authoring tools and reuse

ITS paradigms

In our paper, we discuss:

- Cognitive tutors (based on production rules)
- Model-tracing tutors (based on procedures)
  - xPST: procedures are specified in a 'sequence section', based on 4 operators (Gilbert et al. 2015)
  - ASTUS: hierarchical procedure knowledge is represented as a graph (Paquette et al. 2015)
- Constraint-based tutors
- Example-tracing tutors
- Data-driven tutors

Observation: in most paradigms, an explicit description of a problem-solving procedure is missing

Problem-solving procedures

- Basic operators for combining procedures
  - Sequence $(A ; B)$: first do A, then B
  - Choice $(A \mid B)$: do A or B
  - Fixed-point: for expressing recursive procedures
- Special procedures *succeed* and *fail*
- Primitive procedures are the steps or rules

Realized qualities (by design):

- representation is explicit
- procedures are modular

Trace-based semantics

- Trace-based semantics for step-wise execution
- Traces are inspired by the CSP calculus (Hoare 1985)
- Sequence:

$$\begin{aligned}\mathcal{T}(s\,;\,t) = {} & \{x \mid x \in \mathcal{T}(s), \checkmark \notin x\} \\ & \cup \{xy \mid x\checkmark \in \mathcal{T}(s), y \in \mathcal{T}(t)\}\end{aligned}$$

- Choice:

$$\mathcal{T}(s \mid t) = \mathcal{T}(s) \cup \mathcal{T}(t)$$

Extensible: new composition operators can be added

- by using existing operators: $\mathit{many}\ s = \mu x.(s\,;\,x) \mid \mathit{succeed}$
- or by defining its trace-based semantics

Domain-specific language for procedures

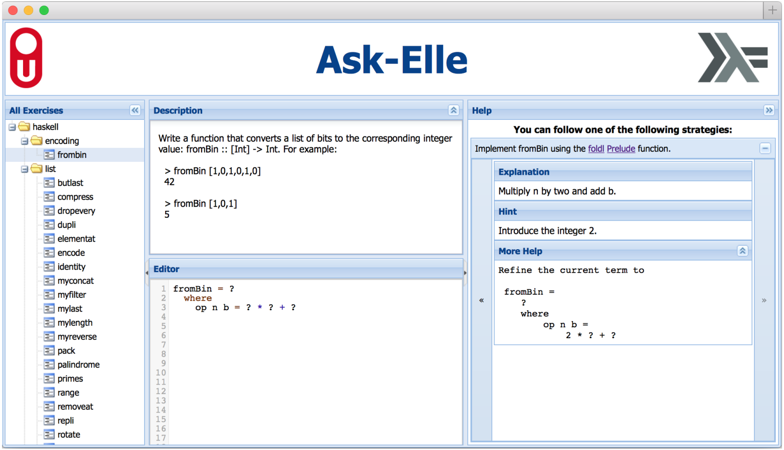
- The composition operators are a simple DSL
- DSL helps authors to articulate procedures
  - generic traversal operators (for domains with sub-terms)
  - variations for choice (e.g., left-biased, preference)
  - interleaving (or permuting) procedures
- It captures common patterns and provides a rich vocabulary, which make the language expressive

Domain-specific language for procedures

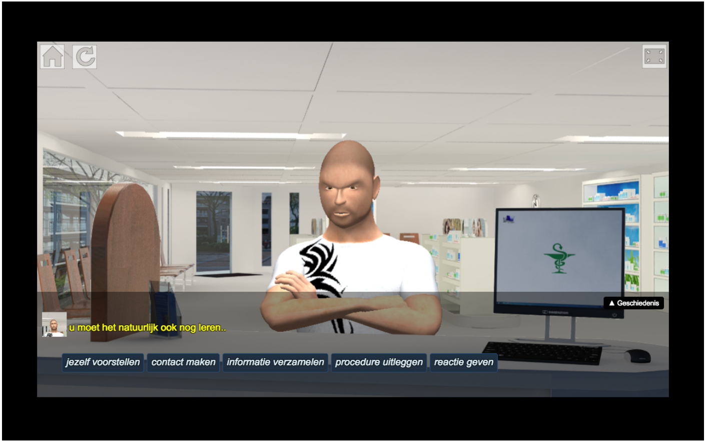
- The composition operators are a simple DSL
- DSL helps authors to articulate procedures
  - generic traversal operators (for domains with sub-terms)
  - variations for choice (e.g., left-biased, preference)
  - interleaving (or permuting) procedures
- It captures common patterns and provides a rich vocabulary, which make the language expressive

$s \,;\, t$	first s, then t	*not s*	succeeds if procedure s is not applicable
$s \mid t$	either s or t		
succeed	succeeding procedure	*repeat s*	apply s as long as possible
fail	failing procedure	*repeat1 s*	as *repeat*, but at least once
$\mu x.f(x)$	fixed point operator	*try s*	apply s once if possible
label ℓ *s*	attach label ℓ to s	$s \triangleright t$	apply s, or else t
many s	apply s 0 or more times	*somewhere s*	apply s at some location
many1 s	apply s 1 or more times	*bottomup s*	search location bottom-up
option s	either apply s or not	*topdown s*	search location top-down

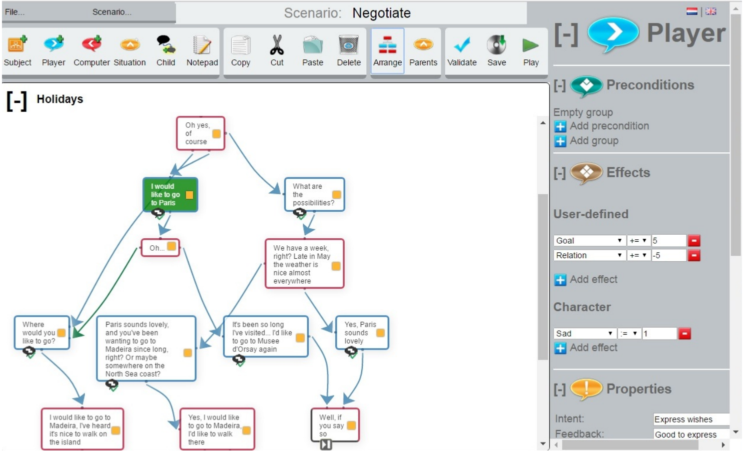
[An extensible domain-specific language for describing problem-solving procedures]



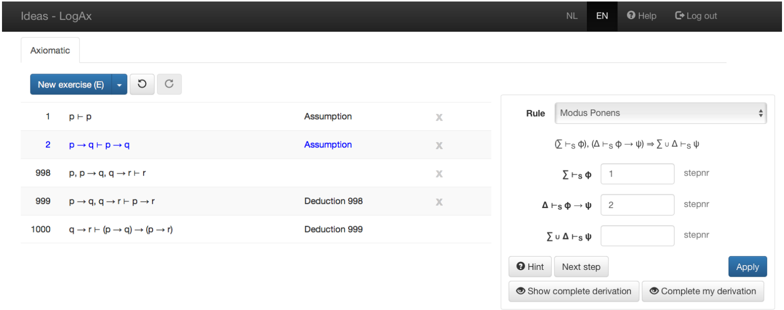
- Develop programs by step-wise refining holes (?)
- Feedback and hints calculated with procedures generated from annotated model solutions (Gerdes et al. 2016)



- Communicate! is a serious game for practicing interpersonal communication skills
- Final score and feedback are calculated afterwards
- It has a specialized scenario editor (Jeuring et al. 2015)



- Domain-specific features for consultations in the scenario editor:
  - conditions under which certain options are offered or not
  - (parts of) consultations may be interleaved in any order
  - (parts of) consultations may be stopped at any point



- Construct Hilbert-style axiomatic proofs by applying rules, forward and backward (Lodder et al. 2017)
- Feedback, hints, and worked-out solutions are available
- Procedure is captured in a graph-like structure

Conclusion

We presented a DSL for problem-solving procedures that:

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

- Our approach positioned more towards productivity and expressiveness than learnability (Murray 2003)
- The trend is away from having explicit procedures; the DSL can help to make authoring procedures more cost-effective

- For more information, contact me at bhr@ou.nl,
  or see the project website http://ideas.cs.uu.nl/

References

- V. Aleven, B.M. McLaren, J. Sewall, and K.R. Koedinger. A new paradigm for intelligent tutoring systems: Example-tracing tutors. Journal of AIED, 19(2):105–154, 2009.
- J.R. Anderson, A.T. Corbett, K.R. Koedinger, and R. Pelletier. Cognitive tutors: lessons learned. Journal of the Learning Sciences, 4(2):167–207, 1995.
- M.J. Beeson. Design principles of MathPert: Software to support education in algebra and calculus. In Computer-Human Interaction in Symbolic Computation, pages 89–115. Springer, 1998.
- A. Gerdes, B. Heeren, J. Jeuring, and L.T. van Binsbergen. Ask-Elle: an adaptable programming tutor for Haskell giving automated feedback. Journal of AIED, pages 1–36, 2016.
- S.B. Gilbert, S.B. Blessing, and E. Guo. Authoring effective embedded tutors: An overview of the extensible problem specific tutor (xPST) system. Journal of AIED, 25(3):428–454, 2015.
- C.A.R. Hoare. Communicating sequential processes. Prentice-Hall, Inc., 1985.
- J. Jeuring, F. Grosfeld, B. Heeren, M. Hulsbergen, R. IJntema, V. Jonker, N. Mastenbroek, M. van der Smagt, F. Wijmans, M. Wolters, and H. van Zeijts. Communicate! – a serious game for communication skills. In EC-TEL 2015, volume 9307 of LNCS, pages 513–517. Springer, 2015.
- J. Lodder, B. Heeren, and J. Jeuring. Generating hints and feedback for Hilbert-style axiomatic proofs. In SIGCSE 2017, pages 387–392, 2017.
- T. Murray. Authoring knowledge-based tutors: Tools for content, instructional strategy, student model, and interface design. Journal of the Learning Sciences, 7(1):5–64, 1998.
- T. Murray. An overview of intelligent tutoring system authoring tools: Updated analysis of the state of the art. In Authoring Tools for Advanced Technology Learning Environments, pages 491–544. Springer, 2003.
- L. Paquette, J.-F. Lebeau, G. Beaulieu, and A. Mayers. Designing a knowledge representation approach for the generation of pedagogical interventions by MTTs. Journal of AIED, 25(1):118–156, 2015.
- K. VanLehn. The behavior of tutoring systems. Journal of AIED, 16(3):227–265, 2006.