Software technology for learning and teaching

Bastiaan Heeren*

Open University of the Netherlands

* Joint work with Johan Jeuring and others

Symposium on the occasion of the Honorary Doctorate for Kurt VanLehn (March 25, 2015)
Software technology for learning and teaching

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

Such tools offer many advantages to users:

1. User can work at any time
2. User can select material and exercises
3. Tool can select exercises based on a user-profile
4. Mistakes can be logged, and reported back to teachers
5. Tool can give immediate feedback
Typical features of such a tool:

- 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
Rewriting to disjunctive normal form

- Rewrite rules for logical propositions:
  \[-\neg \phi \Rightarrow \phi\]
  \[\phi \land (\psi \lor \chi) \Rightarrow (\phi \land \psi) \lor (\phi \land \chi)\]
  \[-(\phi \land \psi) \Rightarrow \neg \phi \lor \neg \psi\]
  \[\phi \lor \psi \lor \chi \Rightarrow (\phi \lor \chi) \lor (\psi \lor \chi)\]
  \[-(\phi \lor \psi) \Rightarrow \neg \phi \land \neg \psi\]

- Exercise: bring \(-\neg(p \lor q) \land r\) to DNF
Rewriting to disjunctive normal form

- Rewrite rules for logical propositions:

\[
\begin{align*}
\neg
\neg \phi & \Rightarrow \phi \\
\phi \land (\psi \lor \chi) & \Rightarrow (\phi \land \psi) \lor (\phi \land \chi) \\
\neg (\phi \land \psi) & \Rightarrow \neg \phi \lor \neg \psi \\
(\phi \lor \psi) \land \chi & \Rightarrow (\phi \land \chi) \lor (\psi \land \chi) \\
\neg (\phi \lor \psi) & \Rightarrow \neg \phi \land \neg \psi
\end{align*}
\]

- Exercise: bring \(\neg(\neg(p \lor q) \land r)\) to DNF

\[
\begin{align*}
\neg(\neg(p \lor q) \land r) & \\
\Rightarrow & \quad \neg\neg(p \lor q) \lor \neg r \\
\Rightarrow & \quad p \lor q \lor \neg r
\end{align*}
\]
Rewriting to disjunctive normal form

- Rewrite rules for logical propositions:

  \[-\neg\phi \Rightarrow \phi \quad \phi \land (\psi \lor \chi) \Rightarrow (\phi \land \psi) \lor (\phi \land \chi)\]

  \[-(\phi \land \psi) \Rightarrow \neg\phi \lor \neg\psi \quad (\phi \lor \psi) \land \chi \Rightarrow (\phi \land \chi) \lor (\psi \land \chi)\]

  \[-(\phi \lor \psi) \Rightarrow \neg\phi \land \neg\psi\]

- Exercise: bring \(\neg(\neg(p \lor q) \land r)\) to DNF

  \[\neg(\neg(p \lor q) \land r) \Rightarrow \neg\neg(p \lor q) \lor \neg r \Rightarrow p \lor q \lor \neg r\]

  \[\neg(\neg(p \lor q) \land r) \Rightarrow \neg((\neg p \land \neg q) \land r) \Rightarrow \neg(\neg p \land \neg q) \lor \neg r \Rightarrow \neg\neg p \lor \neg\neg q \lor \neg r \Rightarrow p \lor \neg\neg q \lor \neg r \Rightarrow p \lor q \lor \neg r\]
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
Goal

Use **software technology**:

- languages and grammars
- algebras
- rewriting techniques

For **learning and teaching**:

- to determine what a student has done
- to determine what a student should do
- to explain (instead of show) why a student performs badly
Outline of presentation

1. Introduction
2. Strategy specification language
3. Feedback services
4. Programming tutors
5. Conclusion
Outline of presentation

1. Introduction

2. Strategy specification language

3. Feedback services

4. Programming tutors

5. Conclusion
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
- ...
To model intelligence in a computer program, Alan 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

To model intelligence in a computer program, Alan 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
A strategy specification language

We need the following concepts for specifying a strategy:

- apply a basic rewrite rule
  - ("\(\land\) distributes over \(\lor\)")

- sequence, <⋆>
  - ("first \ldots then \ldots")

- choice, <■>
  - ("use one of the negation rules")

- apply exhaustively
  - ("repeat \ldots as long as possible")

- traversals
  - ("apply \ldots top down")

The same concepts are found in:

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

<table>
<thead>
<tr>
<th>Feedback service</th>
<th>Parsing problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>ready</td>
<td>is the empty sentence ($\epsilon$) accepted?</td>
</tr>
<tr>
<td>provide hint</td>
<td>compute the “first set”</td>
</tr>
<tr>
<td>worked-out solution</td>
<td>construct a sentence</td>
</tr>
<tr>
<td>after a step</td>
<td>try to recognize the rewrite rule that was used, and parse this rule as the next symbol of the input</td>
</tr>
</tbody>
</table>
Requirements for the strategy language §2

1. **Extensible**: to support common patterns by introducing new combinators

2. **Feedback-oriented**: to generate feedback or hints from a strategy specification at any time

3. **Compositional**: to construct complex strategies by combining and reusing simpler parts

4. **Efficient**: to calculate feedback and hints within a reasonable amount of time

5. **Adaptable**: to fulfil specific needs of teachers and students
More strategy combinators

Easy to add more combinators (inspired by Hoare’s CSP):

**interleaving:** \( a_1a_2a_3 \prec\% b_1b_2 = \{ a_1a_2a_3b_1b_2, a_1a_2b_1a_3b_2, a_1a_2b_1b_2a_3, a_1b_1a_2a_3b_2, \text{ and 6 others} \} \)

**alternating:** \( a_1a_2a_3 \prec\@ b_1b_2 = \{ a_1b_1a_2b_2a_3 \} \)

- We study the **algebraic laws and trace semantics** of our strategy language
1. Introduction

2. Strategy specification language

3. Feedback services

4. Programming tutors

5. Conclusion
Traditionally, an ITS is described by four components:

- Monitoring module for teachers, authoring environment, etc.
- We focus on the expert knowledge module
Following Goguadze, we use the term **domain reasoner**

**Design goals:**
- External, separate component reusable by other learning environments
- Feedback-oriented (e.g., not a CAS)
- Support for an exercise class (not one exercise)
- Calculating feedback is not tied to a particular domain
Proposed design

§3

MathDox

DME

Math-Bridge

Logic tool

LinAlg

Math

Logic

learning environment

client

server

feedback script

generic framework

domain-specific knowledge

XML over HTTP

JSON over HTTP

client

server

[ Software technology for learning and teaching ]
Outline of presentation

1. Introduction
2. Strategy specification language
3. Feedback services
4. Programming tutors
5. Conclusion
Same approach can be applied to the domain of programming:

- Step-wise construction of programs
- Program transformation and normalization
- Program evaluation
- Refactoring programs
- Refinement calculus for program construction
Incrementally solve introductory programming problems
Most work done by **Alex Gerdes** (former PhD student)
Techniques used in Ask-Elle

- Tracing based on teacher specified model solutions
- Program annotations for customizing feedback
- (Semantics-preserving) programming transformations
- Tracing is supplemented by property-based testing

```haskell
{-# DESC Use the prelude function foldl #-}
myreverse =
  {-# FEEDBACK foldl takes an operator and a ... #-}
  (foldl {-# FEEDBACK Use flip and (:) #-}
    (flip (:
      []
    )))
```
Prototype by **Tim Olmer** (MSc student)
Tutor by Hieke Keuning (NWO Doctoral Grant for Teachers)
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.

- Project webpage at http://ideas.cs.uu.nl/
- For more information, contact me at bhr@ou.nl