Software technology for automated feedback generation

Bastiaan Heeren, ICS Colloquium December 17, 2020
Short bio

- Associate professor at the Open University of the Netherlands
- Head of OU’s Computer Science Department

At Utrecht University:
- PhD in Software Technology (2000-2005)
- Lecturer (2005-2007)
- Guest researcher with the Software Technology for Learning and Teaching research group

Bastiaan Heeren
Intelligent Tutoring Systems

Intelligent Tutoring Systems (ITS): computer systems that provide immediate and customized feedback to learners

Structure:
- Classical architecture with four components

Behaviour:
- Outer loop: solving one task after another
- Inner loop: the steps for solving one complex, multi-step problem

Software technology for automated feedback generation
Example: axiomatic proofs

<table>
<thead>
<tr>
<th>Step</th>
<th>Formula</th>
<th>Type</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$p \rightarrow p$</td>
<td>Assumption</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$p \rightarrow q, p \rightarrow q$</td>
<td>Assumption</td>
<td></td>
</tr>
<tr>
<td>998</td>
<td>$p, p \rightarrow q, q \rightarrow r \rightarrow r$</td>
<td>Deduction</td>
<td></td>
</tr>
<tr>
<td>999</td>
<td>$p \rightarrow q, q \rightarrow r \rightarrow p \rightarrow r$</td>
<td>Deduction</td>
<td>$998$</td>
</tr>
<tr>
<td>1000</td>
<td>$q \rightarrow r \rightarrow (p \rightarrow q) \rightarrow (p \rightarrow r)$</td>
<td>Deduction</td>
<td>$999$</td>
</tr>
</tbody>
</table>

*Step-wise construction*

*Multiple solutions are accepted*

*Feedback and hints*
Research motivation

1. Simplify construction of ITSs (which are complex software systems)
2. Represent expert domain knowledge explicitly (for better feedback)
3. Apply approach to a wide range of problem domains

Approach: use software technology for automated feedback generation

Techniques in this presentation (outline):
- Rewrite strategies for automated feedback (basics)
- Light-weight rewrite rules
- Generic traversals
Problem domains

Four recent PhD theses, for different problem domains, all based on the same approach
Ideas framework

Generic framework for constructing domain reasoners

- Developed in Haskell
- Size: 12,397 LOC
- Open source
- Independent of problem domain
- http://ideas.cs.uu.nl/tutorial/

Interactive Domain-specific Exercise Assistants
Rewrite strategies
Rewrite strategies for automated feedback

- Domain-specific language for specifying problem-solving procedures:
  - describe sequences of rule applications that solve a particular task
  - are formalized by a trace-based semantics (CSP)
  - allow new composition operators (interleaving, topological sorts)

- Problem-solving procedures are used for feedback generation:
  - recognizing the solution strategy
  - detecting detours
  - suggesting subgoals
  - providing next-step hints
  - providing worked-out examples
Example

Goal: rewrite proposition into negation normal form (NNF)

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

\[ \Leftrightarrow \] De Morgan

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

\[ \Leftrightarrow \] De Morgan

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

\[ \Leftrightarrow \] Double Neg

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

\[ \checkmark \]

Rewrite strategy for NNF:

alternatives

repeat (oncetd (doubleNeg .|. dmOr .|. dmAnd))

top-down application

rewrite rule
### Strategy combinators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p .*. q</td>
<td>sequence: first p, then q</td>
</tr>
<tr>
<td>succeed</td>
<td>always succeeds</td>
</tr>
<tr>
<td>p .</td>
<td>. q</td>
</tr>
<tr>
<td>p ./. q</td>
<td>preference: p is preferred over q</td>
</tr>
<tr>
<td>p</td>
<td>&gt; q</td>
</tr>
<tr>
<td>fix</td>
<td>fixed-point combinator</td>
</tr>
</tbody>
</table>

Derived combinators:
- `try s = s |> succeed`
- `repeat s = try (s .*. repeat s)`

Finite representation with explicit recursion:

```
repeat s = fix $ \x ->
try (s .*. x)
```

Advantages:
- Extract rules from strategy
- Customize strategy
- Document/visualise strategy
Light-weight rewrite rules
Proposition logic

data Logic = Logic :&&: Logic    -- conjunction
     | Logic :||: Logic    -- disjunction
     | Not Logic          -- negation
     | Var String         -- variable

Representation can be more complex, with nested and parameterised datatypes, e.g.:

\[ 3x + 9 = 0 \lor x = 1 \]
Rewrite rules

doubleNeg = rewriteRule "doubleNeg" $
\phi \rightarrow \text{Not (Not } \phi) \ :~>\ : \phi$

dmAnd = rewriteRule "dmArd" $
\phi \ psi \rightarrow \text{Not } (\phi :&&: \ psi) \ :~>\ : \text{Not } \phi :||: \text{Not } \psi$

meta-variables are introduced by lambdas

left-hand side

right-hand side

How to use such rewrite rules?
Embedding-projection pair

Approach: conversion from/to a generic Term datatype with support for meta-variables

```
  toTerm    :: Logic    -> Term
  fromTerm :: Term      -> Maybe Logic
```

- From/to should be inverse functions (intuitively)
- Conversion allows generic functions, such as unification and zippers
- Pair can be derived automatically from the datatype definition

Note: more powerful generic programming libraries exist that can guarantee more type safety, with less overhead
Compiling rewrite rules

\phi \to \text{Not(Not}\ phi) :\to \phi

Step 1: use two different values (e.g. Var “p” and Var “q”):
- \text{Not(Not(Var “p”))} :\to \text{Var “p”}
- \text{Not(Not(Var “q”))} :\to \text{Var “q”}

Step 2: convert to Term datatype:
- \text{TCon “Not” [TCon “Not” [TVar “p”]]} :\to \text{TVar “p”}
- \text{TCon “Not” [TCon “Not” [TVar “q”]]} :\to \text{TVar “q”}

Step 3: find meta-variables by comparing left-hand sides and right-hand sides
- \text{TCon “Not” [TCon “Not” [TMeta 0]]} :\to \text{TMeta 0}

values provided by user for problem domain

TCon, TVar, and TMeta are constructors of Term

Software technology for automated feedback generation
Applying rewrite rules

Not (Not (Var “p” :&&: Var “r”))

Step 1: convert to Term datatype:

\[ \text{TCon “Not” [TCon “Not” [TCon “And” [TVar “p”, TVar “q”]]]} \]

Step 2: match with rule’s left-hand side:

\[ 0 = \text{TCon “And” [TVar “p”, TVar “q”]} \]

Step 3: substitute in rule’s right-hand side:

\[ \text{TCon “And” [TVar “p”, TVar “q”]} \]

Step 4: convert back to Logic:

\[ \text{Var “p” :&&: Var “r”} \]

Rewrite rule:

\[ \text{TCon “Not” [TCon “Not” [TMeta 0]] :~> TMeta 0} \]
Knuth-Bendix completion

Use case for explicit representation: search for missing rewrite rules (and reach confluence)

Critical pair

\[ \neg(\phi \land \psi) \]

\[ \phi \land \psi \quad \neg(\neg\phi \lor \neg\psi) \]

\[ \iff \]

\[ \neg\neg\phi \land \neg\neg\psi \]

\[ \iff \]

\[ \phi \land \psi \]

\[ \checkmark \]

\[ \neg\neg\phi \iff \phi \]

\[ \neg(\phi \land \psi) \iff \neg\phi \lor \neg\psi \]

Missing rule:

\[ \neg(\phi \lor \psi) \iff \neg\phi \land \neg\psi \]
Light-weight rewrite rules

Advantages of explicit representation:
- Knuth-Bendix completion (analysis)
- AC-rewriting
- Rule inversion
- Automated testing
- Documentation (pretty-printing)

Summary for rewrite rules:
- Simplify construction (light-weight embedding)
- Explicit representation (for better feedback)
- Many problem domains
Generic traversals
Tree representation

\[ \neg((p \lor q) \land \neg(p \land r)) \]
Point of focus

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

- Implemented as a so-called zipper over the generic Term datatype
- Stored in a Context
Five navigational actions:
- up
- left
- right
- down
- downLast

- Actions may fail
- Many useful laws, e.g.:
  - left $\circ$ right $\approx$ id
  - up $\circ$ down $\approx$ id
Navigation (extended)

Zippers keep a position for the point of focus

Position information is useful for generating feedback
Horizontal visits

visitOne $s = \text{fix} \; \lambda x \rightarrow s \cdot.|. (\text{right} \; .*. x)$

visitFirst $s = \text{fix} \; \lambda x \rightarrow s \rightarrow (\text{right} \; .*. x)$

visitAll $s = \text{fix} \; \lambda x \rightarrow s \; .*. (\text{not} \; \text{right} \rightarrow (\text{right} \; .*. x))$

- Approach: define traversals as (normal) strategy combinators
- Idea: also parameterize “next” function to also support right-to-left visits
One-layer visits

layer s = down .*. s .*. up

layerOne s = layer (visitOne s)
Traversals

somewhere $s = \text{fix} \ ($ $x \rightarrow s \ .|. \ \text{layerOne} \ x$)

\quad \text{once} \text{td} \ s \quad = \text{fix} \ ($ $x \rightarrow s \ |> \ \text{layerOne} \ x$) \quad \text{-- top down}

\quad \text{once} \text{bu} \ s \quad = \text{fix} \ ($ $x \rightarrow \text{layerOne} \ x \ |> \ s$) \quad \text{-- bottom up}

- Also: full traversals, spine traversals, innermost, outermost, etc.
Example trace

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

\[\equiv \text{De Morgan}\]

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

\[\equiv \text{De Morgan}\]

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

\[\equiv \text{Double Neg}\]

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

Corresponding trace:

- De Morgan at [0]
- down
- De Morgan at [1]
- up
- Double Neg at [1]
- right
- Double Neg at [1]
- up

Summary for traversals:

- Simplify construction (traversals are first-class strategy combinators)
- Explicit representation (for better feedback)
- Many problem domains
Conclusion
Trends and challenges

- **Authoring** intelligent tutoring system
  - Literature reports 200-300 authoring hours for 1 hour of instruction
  - We believe software technology can help

- **Data-driven** intelligent tutoring system
  - Use AI techniques to generate feedback from collected data
  - Raises questions about the role of expert domain knowledge

- **Further** adaptation and personalization
  - Models for mastery learning (e.g. Bayesian knowledge tracing)

- **Designing tools for** less-structured problem domains
  - For example, domains of software design and learning languages
Conclusion

- Rewrite strategies are used for feedback generation
- Rewrite rules can be embedded by using datatype-generic programming techniques
- Generic traversals can be composed from navigational actions and strategy combinators
- The presented approach can be applied to a wide range of problem domains

✉ Bastiaan.Heeren@ou.nl

Project website: http://ideas.cs.uu.nl/
Related publications


