Ask-Elle

An Adaptable Programming Tutor for Haskell Giving Automated Feedback

Bastiaan Heeren

April 26, 2016

OU Research Seminar

Open Universiteit
www.ou.nl

Int J Artif Intell Educ
DOI 10.1007/s40593-015-0080-x

ARTICLE

Ask-Elle: an Adaptable Programming Tutor for Haskell Giving Automated Feedback

Alex Gerdes[1] · Bastiaan Heeren[2] · Johan Jeuring[3] · L. Thomas van Binsbergen[4]

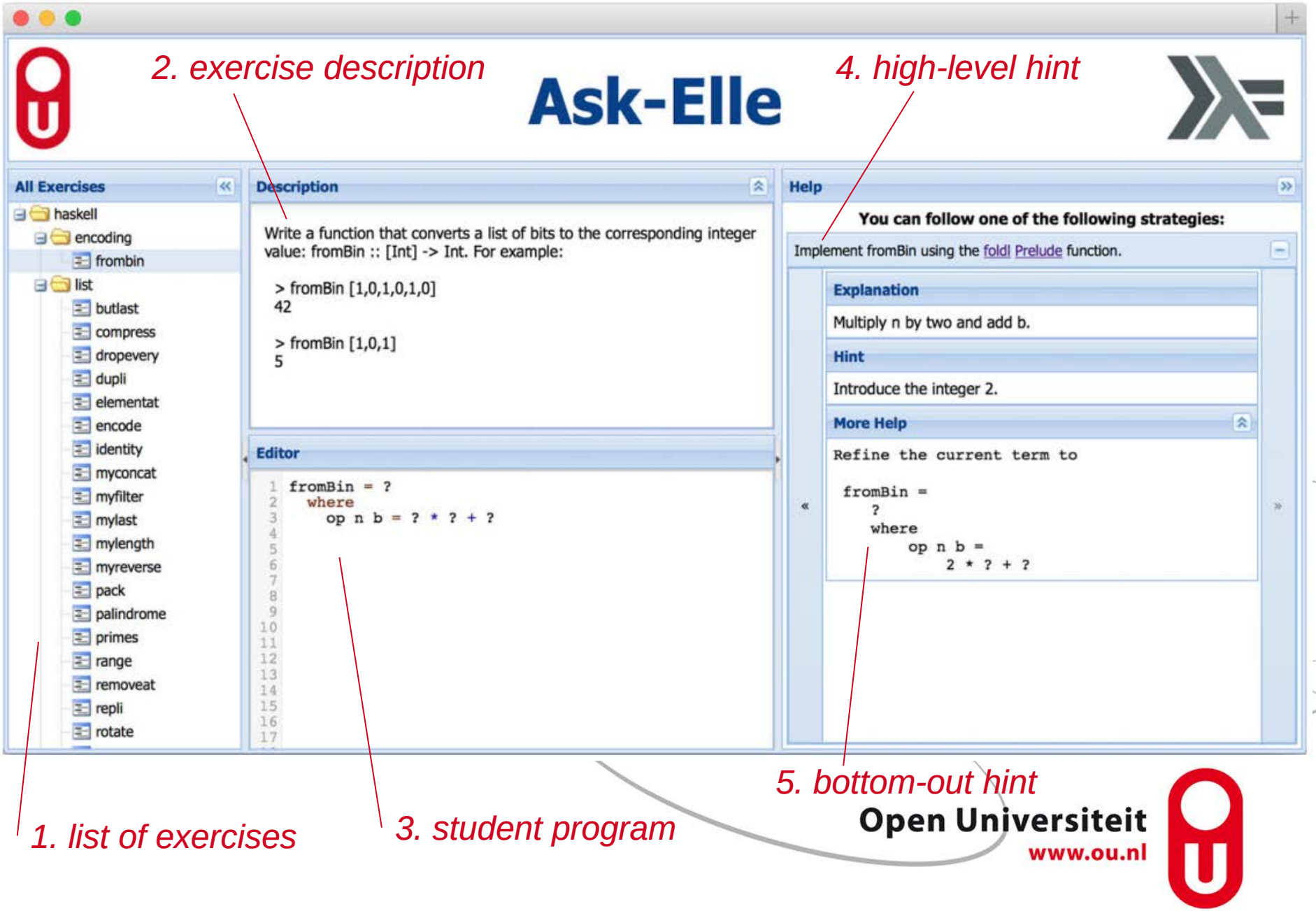
[1] Chalmers University of Technology, Gothenburg, Sweden

[2] Open Universiteit Nederland, Heerlen, Netherlands

[3] Utrecht University and Open Universiteit Nederland, Utrecht, Netherlands

[4] Royal Holloway, University of London, Egham, UK

Published online: 05 February 2016



Open Universiteit
www.ou.nl

Why use an ITS?

Evaluation studies have indicated that:

- ITS with stepwise development is almost as effective as a human tutor (VanLehn 2011)
- More effective when learning how to program than “on your own” with compiler, or pen and paper (Corbett et al. 1988)
- Requires less help from teacher while showing same performance on tests (Odekirk-Hash and Zachary 2001)
- Increases self-confidence of female students (Kumar 2008)
- Immediate feedback of ITS is preferred over delayed feedback common in classroom settings (Mory 2003)

Type of exercises

- Determines how difficult it is to generate feedback
- Classification by Le and Pinkwart (2014):
  - Class 1: single correct solution
  - Class 2: different implementation variants
  - Class 3: alternative solution strategies

- Ask-Elle offers class 3 exercises

Ask-Elle's contribution

The design of a programming tutor that:

1. offers class 3 exercises
2. supports incremental development of solutions
3. automatically calculates feedback and hints
4. allows teachers to add exercises and adapt feedback

Our approach:

- strategy-based model tracing
- property-based testing
- compiler technology for FP languages

Open Universiteit
www.ou.nl

Overview

- Session: student & teacher
- Design
- Experiment 1: assessment
- Experiment 2: questionnaire
- Experiment 3: student program analysis
- Conclusions

Example

Write a function that converts a list of bits to the corresponding decimal value:

$$fromBin :: [Int] \rightarrow Int$$

For example:

$> fromBin\ [1,0,1,0,1,0]$
42
$> fromBin\ [1,0,1]$
5

32 + 8 + 2 = 42

- Available hints:

we follow the foldl approach

You can proceed in several ways:
- Implement *fromBin* using the *foldl* prelude[2] function.
- Take the inner product with a list of factors of two.
- Implement *fromBin* with a helper function using an extra parameter.

Session

a hole (expression)

$fromBin = \bullet$

Define the *fromBin* function using *foldl*. The operator should multiply the intermediate result with two and add the value of the bit.

$$fromBin = foldl\ op\ \bullet$$
where
$$op\ \bullet\ \bullet = \bullet$$

$$fromBin = foldl\ op\ \bullet$$
where
$$op\ n\ b = \bullet + \bullet$$

Multiply n by two and then add b.

Session (continued)

```
fromBin = foldl op •
  where
    op n b = 2 * n + c
```

standard compiler error by Helium

```
Error: undefined variable c
```

```
fromBin = foldl op 1
  where
    op n b = 2 * n + b
```

```
Your implementation is incorrect for the following input: []
We expected 0, but we got 1
```

```
fromBin = foldl op 0
  where
    op n b = 2 * n + b
```

Model solutions

- Teachers can supply model solutions

```
   -- 1. Solution with foldl
fromBin = foldl op 0
      where
         op n b = 2 * n + b
   -- 2. Inner product with powers of two
fromBin = sum ∘ zipWith (*) (iterate (*2) 1) ∘ reverse
   -- 3. Tupling, passing around the length as extra argument
fromBin bs = fromBin' (length bs − 1) bs
   where
      fromBin' _ []       = 0
      fromBin' n (b : bs) = b * 2^n + fromBin' (n − 1) bs
```

Fig. 2 Three model solutions for the *fromBin* programming exercise

Recognising solutions

- Aggressive normalisation
- Semantic equality of programs is undecidable
- For example:

$$
\begin{aligned}
fromBin\ xs = \mathbf{let}\ f\ z\ [\,] &= z \\
f\ z\ (x : xs) &= f\ (base * z + x)\ xs \\
base &= 2 \\
start &= 0 \\
\mathbf{in}\ f\ start\ xs &
\end{aligned}
$$

can be recognised by:

```
-- 1. Solution with foldl
fromBin = foldl op 0
    where
      op n b = 2 * n + b
```

Adapting feedback

description of the solution

```
{-# DESC Implement fromBin using the foldl prelude function. #-}
```

textual feedback annotations

```
fromBin =
  {-# FB Define the fromBin function using foldl. The op... #-}
  foldl op 0
    where
      op n b = {-# FB Multiply n by two and add b. #-}2 * n + b
```

enforce use of library function

```
fromBin = sum ∘ zipWith (*) ({-# MUSTUSE #-}iterate (*2) 1) ∘ reverse
```

alternative definition

```
{-# ALT foldl op b = foldr (flip op) b ∘ reverse #-}
```

Properties

- Used for reporting counter-examples

f is the student program

```
propModel f bs = feedback msg (output == model)
  where
    output = f bs
    model  = foldl (λn b → 2 * n + b) 0 bs
    msg    = "Your implementation is incorrect for the " ++
             "following input: " ++ show bs ++ "\nWe expected " ++
             show model ++ ", but we got " ++ show output
```

round-trip property

```
propSound f bs = feedback msg (bs == toBin (f bs))
  where
    msg = "Converting back results in a different list of bits"
```

Ask-Elle's design

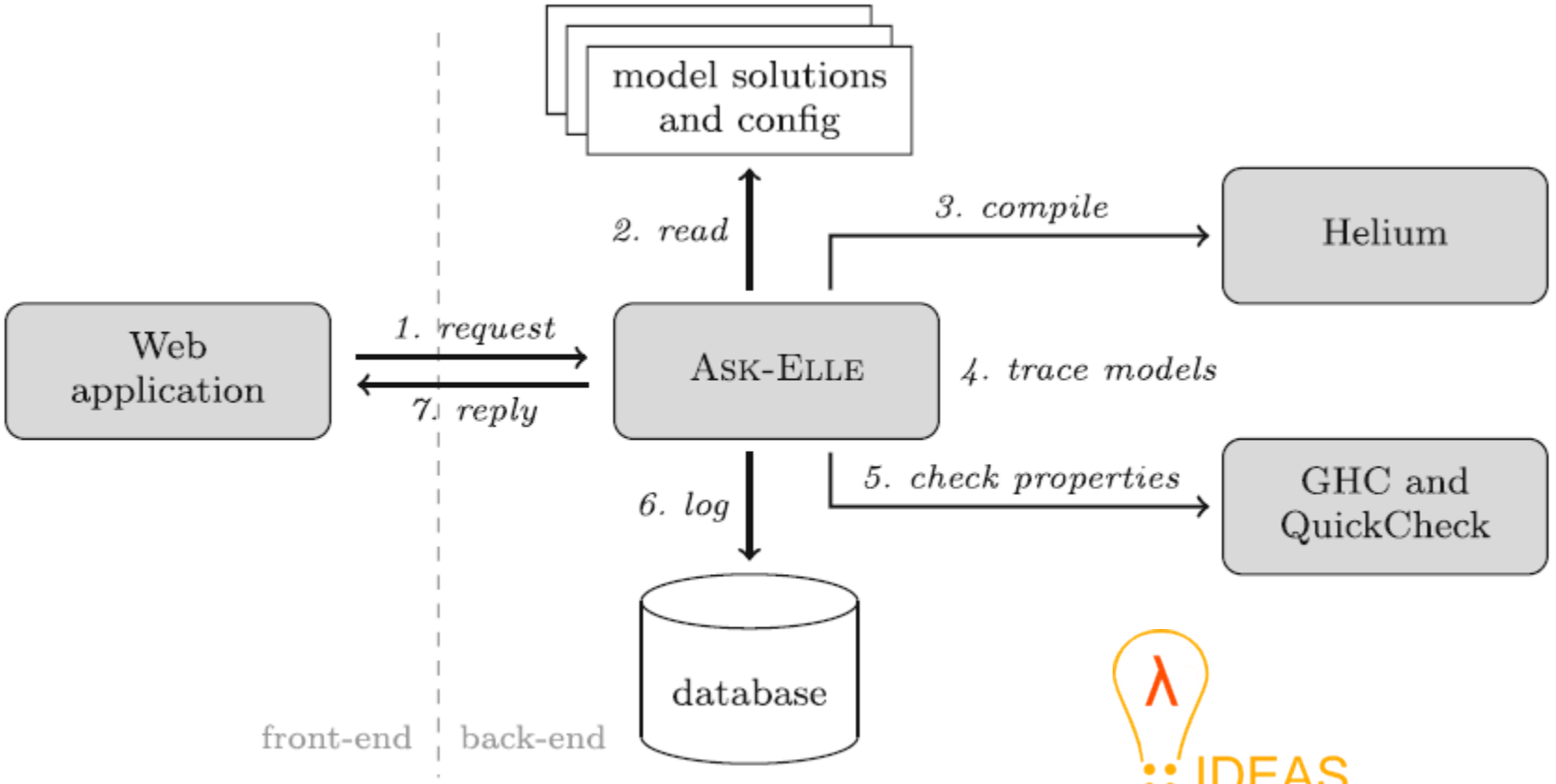


Fig. 3 Ask-Elle's web-based architecture

Experiment 1:

Assessing Student Programs

Automated assessment

- Many tools use some form of testing
- Problems with testing: how do you know …
  1. you have tested enough (coverage)?
  2. that good programming techniques are used?
  3. which algorithm was used?
  4. the executed code has no malicious features?

- Strategy-based assessment solves these problems

Classification (by hand)

- Good: proper solution (correctness and design)
- Good with modifications: solutions augmented with sanity checks (e.g. input checks)
- Imperfect: program contains imperfections: e.g. superfluous cases, length (x:xs) - 1

- First-year FP course at UU (2008)
  - 94 submissions for fromBin
  - 64 are good, 8 good with modifications (total: 72)

Results

- 62 of 72 (86%) are recognized based on 4 model solutions
- No false positives
- Model solutions: foldl (18), tupling (2), inner product (2)
- Explicit recursion (40), which is simple but inefficient

$$\begin{aligned} fromBin\ [\,] &= 0 \\ fromBin\ (b : bs) &= b * 2\,\hat{}\,length\ bs + fromBin\ bs \end{aligned}$$

- Example of program that was not recognized:

$$\begin{aligned} fromBin\ [\,] &= 0 \\ fromBin\ [b] &= b \\ fromBin\ (b : c : rest) &= fromBin\ ((2 * b + c) : rest) \end{aligned}$$

Experiment 2:

Questionnaire

Open Universiteit
www.ou.nl

Questionnaire

- FP bachelor course at UU (September 2011) with 200 students
- Approx. 100 students used the tutor in two sessions (week 2)
- Forty filled out the questionnaire (Likert scale, 1-5)

- Experiment was repeated for:
  - FP experts from the IFIP WG 2.1 group
  - Student participants of the CEFP 2011 summer school

Results

Table 1 Questionnaire: questions and scores

#	Question	Score
1	The tutor helped me to understand how to write simple functional programs	3.15
2	I found the high-level hints about how to solve a programming problem useful	3.43
3	I found the hints about the next step to take useful	3.05
4	The step-size of the tutor corresponded to my intuition	2.85
5	I found the possibility to see the complete solution useful	4.25
6	The worked-out solutions helped me to understand how to construct programs	3.55
7	The feedback texts are easy to understand	3.25
8	The kind of exercises offered are suitable for a first functional programming course	3.90

Evaluation of open questions

Remarks that appear most:

- Some solutions are not recognised by the tutor
  - Incorrect solution? Give counterexample
- The response of the tutor is sometimes too slow
  - Special ‘search mode’

Experiment 3:

Student Program Analysis

Classification (by Ask-Elle)

Correctness:

- For full program: expected input-output behaviour
- For partial program: can be refined to correct, full program

Categories:

- Compiler error (Error)
- Matches model solution (Model)
- Counterexample (Counter)
- Undecided, separated into Tests passed and Discarded

Questions related to feedback quality

- How many programs are classified as undecided?
- How often would adding a program transformation help?
- How often would adding a model solution help?
- How often do students add irrelevant parts?
- How many of the programs with correct input–output behaviour contain imperfections (hard to remove)?
- How often does QuickCheck not find a counterexample, although the student program is incorrect?

(precise answers in paper)

Correct (but no match)

Cases:

1. The student has come up with a way to solve the exercise that significantly differs from the model solutions
2. Ask-Elle misses some transformations
3. The student has solved more than just the programming exercise (e.g. extra checks)
4. The student implementation does not use good programming practices or contains imperfections

Incorrect (but no counterexample)

Cases:

1. Tests passed. All test cases passed. By default, 100 test cases are run with random values for each property.
2. Discarded. Too many test cases are discarded. By default, more than 90% is considered to be too many.

Results

- September 2013 at UU: 5950 log entries from 116 students
- Exercise attempts (last program) and interactions
- Recognized: Model / (Model + Passed + Discarded)
- Classified: (Model + Error + Counter) / Total

Category	Attempts	Interactions
Compiler error	142 (21.8%)	1920 (55.4%)
Model	221 (33.9%)	754 (21.8%)
Counter	33 (5.1%)	201 (5.8%)
Tests passed	235 (36.0%)	436 (12.6%)
Discarded	21 (3.2%)	155 (4.5%)
total	652	3466
recognised	221/477 (46.3%)	754/1345 (56.1%)
classified	396/652 (60.7%)	2875/3466 (82.9%)

Missing program transformations

Analysis (by hand) of 436 interactions in ‘Tests passed’:

- Remove type signature (94)
- Recognise more prelude functions and alternative definitions (37); followed by beta-reduction (39)
- Formal parameters versus lambda’s, eta-conversion (75)
- Alpha-conversion bug (48), wildcard (19)
- Better inlining (26)
- Substituting equalities a==b (26)
- Removing syntactic sugar (22)
- (…)

Updated results

Category		Interactions
Compiler error	1920 (55.4%)	1920 (55.4%)
Model	754 (21.8%)	1095 (31.6%)
Counter	201 (5.8%)	206 (5.9%)
Tests passed	436 (12.6%)	87 (2.5%)
Discarded	155 (4.5%)	158 (4.6%)
total	3466	3466
recognised	754/1345 (56.1%)	1095/1340 (81.7%)
classified	2875/3466 (82.9%)	3221/3466 (92.9%)

original results

Conclusions

- Ask-Elle supports the incremental development of programs for class 3 programming exercises
- Feedback and hints are automatically calculated from teacher-specified annotated model solutions and properties
- Main technologies: strategy-based model tracing and property-based testing.
- With improvements from last experiment:
  - recognise nearly 82% of (correct) interactions
  - classify nearly 93% of interactions

Future work

- Other programming languages and paradigms
- Measure learning effects and effectiveness
- Draw up a feedback benchmark
- Abstract model solutions (recursion patterns)
- Contracts for blame assignment
- Systematic literature review on feedback in learning environments for programming
  - Part 1 to be presented at ITiCSE 2016 (69 tools)