Ask-Elle: A Haskell tutor
— Demonstration —

Johan Jeuring

Alex Gerdes

Bastiaan Heeren

Technical Report UU-CS-2012-010

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

ASK-ELLE: A Haskell tutor — Demonstration —

Johan Jeuring[1,2], Alex Gerdes[1], and Bastiaan Heeren[1]

[1]School of Computer Science, Open Universiteit Nederland
P.O.Box 2960, 6401 DL Heerlen, The Netherlands
{jje, age, bhr}@ou.nl
[2] Department of Information and Computing Sciences, Universiteit Utrecht

Abstract. In this demonstration we will introduce ASK-ELLE, a Haskell tutor. ASK-ELLE supports the incremental development of Haskell programs. It can give hints on how to proceed with solving a programming exercise, and feedback on incomplete student programs. We will show ASK-ELLE in action, and discuss how a teacher can configure its behaviour.

1 Introduction

ASK-ELLE[1] is a programming tutor for the Haskell [5] programming language, targeting bachelor students at the university level starting to learn Haskell. Using ASK-ELLE, a student can:

- develop a program incrementally,
- receive feedback about whether or not she is on the right track,
- ask for a hint when she is stuck,
- can see how a complete program is stepwise constructed.

ASK-ELLE is an example of an intelligent tutoring system [6] for the domain of functional programming.

In this demonstration we will show ASK-ELLE in action, by means of some interactions of a hypothetical student with the tutor. Furthermore, we will show how a teacher can configure the behaviour of ASK-ELLE. This demonstration accompanies our paper introducing the technologies we use to offer flexibility to teachers and students in our tutor [1,2]. Jeuring et al. [4] give more background information about ASK-ELLE.

2 ASK-ELLE in action

We will start our demonstration of ASK-ELLE with showing some interactions of a hypothetical student with the functional programming tutor. A screenshot of ASK-ELLE is shown in Figure 1. It sets small functional programming tasks,

[1] http://ideas.cs.uu.nl/ProgTutor/

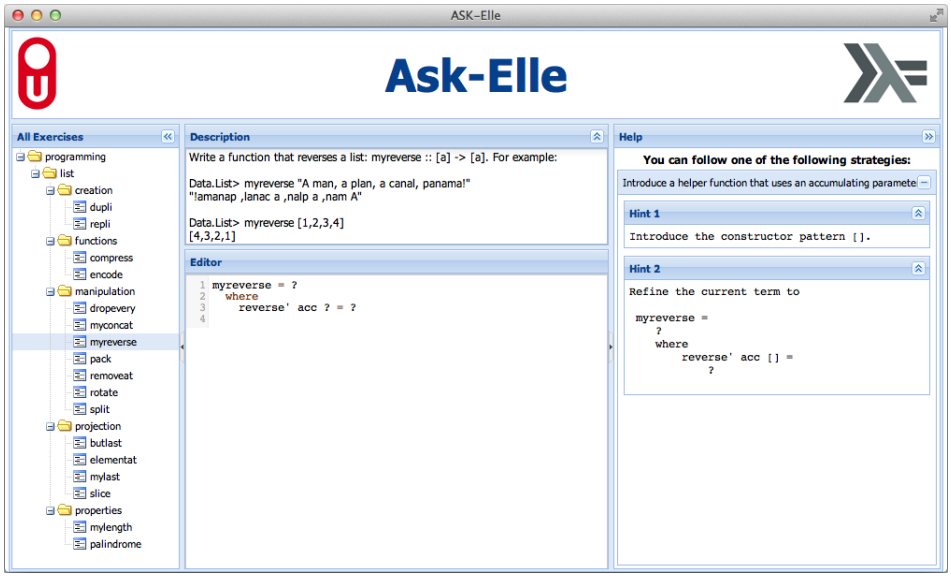

Fig. 1. The web-based functional programming tutor

and gives feedback in interactions with the student. We assume that the student has attended lectures on how to write simple functional programs on lists.

At the start of a tutoring session the tutor gives a problem description. Here the student has to write a program to construct a list containing all integers within a given range.

```
Write a function that creates a list with all integers between
a given range:
```

$$range :: Int \to Int \to [Int]$$

```
For example:
```

$$> range\ 4\ 9$$
$$[4,5,6,7,8,9]$$

and displays the name of the function to be defined, along with its parameters:

$$range\ x\ y = \bullet$$

The task of a student is to refine the holes, denoted by $\bullet$, of the program. After each refinement, a student can ask the tutor whether or not the refinement is bringing him or her closer to a correct solution. If a student doesn't know how to proceed, she can ask the tutor for a hint. A student can also introduce new declarations, function bindings, and alternatives.

Suppose the student has no idea where to start and asks the tutor for help. The tutor offers several ways to help. For example, it can list all possible ways to proceed solving an exercise. In this case, the tutor would respond with:

```
You can proceed in several ways:
- Implement range using the unfoldr function.
- Use the enumeration function from the prelude[2].
- Use the prelude functions take and iterate.
```

We assume a student has some means to obtain information about functions and concepts that are mentioned in the feedback given by the tutor. This information might be obtained via lectures, an assistant, lecture notes, or even via the tutor at some later stage. The tutor can make a choice between the different possibilities, so if the student doesn't want to choose, and just wants a single hint, she gets:

```
Implement range using the unfoldr function.
```

Here we assume that the teacher has set up the tutor to prefer the solution that uses *unfoldr*, defined by:

$$\begin{aligned} &\mathit{unfoldr} :: (b \to \mathit{Maybe}\ (a, b)) \to b \to [a] \\ &\mathit{unfoldr}\ f\ b = \mathbf{case}\ f\ b\ \mathbf{of}\ \mathit{Just}\ (a, b') \to a : \mathit{unfoldr}\ f\ b' \\ &\qquad\qquad\qquad\qquad\qquad\quad \mathit{Nothing} \quad\ \to [\,] \end{aligned}$$

The higher-order function *unfoldr* builds a list from a seed value b. The argument f is a producer function that takes the seed element and returns *Nothing* if it is done producing the list, or $\mathit{Just}\ (a, b')$, in which case a is prepended to the output list and b' is used as the seed value in the recursive call.

The student can ask for more detailed information at this point, and the tutor responds with increasing detail:

```
Define function range in terms of unfoldr, which takes two
arguments: a seed value, and a function that produces a new
value.
```

with the final bottom-out hint:

Define: $\mathit{range}\ x\ y = \mathit{unfoldr}\ \bullet\ \bullet$

At this point, the student can refine the function at two positions. We do not impose an order on the sequence of refinements. Suppose that the student chooses to first implement the producer function:

$$\mathit{range}\ x\ y = \mathit{unfoldr}\ f\ \bullet \quad \mathbf{where}\ f\ i \mid \bullet = \bullet$$

Note that the student has started to define the producer function in a **where** clause. She continues with the introduction of the stop criterion:

[2] The prelude is the standard library for Haskell containing many useful functions.

$$\mathit{range}\ x\ y = \mathit{unfoldr}\ f\ \bullet \quad \textbf{where}\ f\ i \mid i == y + 1 = \bullet$$

There are several ways in Haskell to implement a condition. Here the student has chosen to define the function f with a so-called guarded expression; the predicate after the vertical bar acts as a guard. The student continues with:

$$\mathit{range}\ x\ y = \mathit{unfoldr}\ f\ \bullet \quad \textbf{where}\ f\ i \mid i == y + 1 = \mathit{Just}\ \bullet$$

The tutor responds with:

```
Wrong solution: range 4 6 provides a counterexample.
```

The partial definition of f does not match any of the correct solutions, and by means of random testing the tutor can find an example where the result of the student program differs from a model solution. Correcting the error, the student enters:

$$\mathit{range}\ x\ y = \mathit{unfoldr}\ f\ \bullet \quad \textbf{where}\ f\ i \mid i == y + 1 = \mathit{Nothing}$$

which is accepted by the tutor. If the student now asks for a hint, the tutor responds with:

```
Introduce a guarded expression that gives the output value and
the value for the next iteration.
```

She continues with

$$\begin{aligned}\mathit{range}\ x\ y = \mathit{unfoldr}\ f\ \bullet \quad \textbf{where}\ f\ i \mid\ & i == y + 1 = \mathit{Nothing} \\ \mid\ & \mathit{otherwise} = \mathit{Just}\ \bullet\end{aligned}$$

which is accepted, and then

$$\begin{aligned}\mathit{range}\ x\ y = \mathit{unfoldr}\ f\ \bullet \quad \textbf{where}\ f\ i \mid\ & i == y + 1 = \mathit{Nothing} \\ \mid\ & \mathit{otherwise} = \mathit{Just}\ (n, i + 1)\end{aligned}$$

which gives:

```
Error: undefined variable n
```

This is a syntax-error message generated by the Helium [3] compiler, which we use in our tutor. The student continues with:

$$\begin{aligned}\mathit{range}\ x\ y = \mathit{unfoldr}\ f\ x \quad \textbf{where}\ f\ i \mid\ & i == y + 1 = \mathit{Nothing} \\ \mid\ & \mathit{otherwise} = \mathit{Just}\ (i, i + 1)\end{aligned}$$

which completes the exercise.

A student can develop a program in any order, as long as all variables are bound. For example, a student can write

$$\mathit{range}\ x\ y = \bullet \quad \textbf{where}\ f\ i \mid \bullet = \bullet$$

and then proceed with defining f. This way, bottom-up developing a program is supported to some extent.

These interactions show that our tutor can give hints about which step to take next, in various levels of detail, list all possible ways in which to proceed, point out errors, and pinpoint where the error appears to be, and show a complete worked-out example.

3 Configuring the behaviour of ASK-ELLE

In this part of the demonstration we show how a teacher adds a programming exercise to the tutor by specifying model solutions for the exercise, and how a teacher adapts the feedback given by the tutor.

3.1 Adding an exercise

The interactions of the tutor are based on *model solutions* to programming problems. A model solution is a program that an expert writes, using good programming practices. We have specified three model solutions for *range*. The first model solution uses the enumeration notation from Haskell's prelude:

$$range\ x\ y = [x\,..\,y]$$

The second model solution uses the prelude functions *take*, which given a number n and a list xs returns the first n elements of xs, and *iterate*, which takes a function and a start value, and returns an infinite list in which the next element is calculated by applying the function to the previous element:

$$range\ x\ y = take\ (y - x + 1)\ (iterate\ (+1)\ x)$$

The last model solution uses the higher-order function *unfoldr*:

$$\begin{aligned} range\ x\ y = unfoldr\ f\ x \quad \textbf{where}\ f\ i \mid\ & i == y + 1 = Nothing \\ \mid\ & otherwise = Just\ (i, i + 1) \end{aligned}$$

The tutor uses these model solutions to generate feedback. It recognises many variants of a model solution. For example, the following solution:

$$\begin{aligned} range\ x\ y = \textbf{let}\ & f = \lambda a \to \textbf{if}\ a == y + 1\ \textbf{then}\ Nothing\ \textbf{else}\ Just\ (a, a + 1) \\ & g = \lambda f\ x \to \textbf{case}\ f\ x\ \textbf{of}\ Just\ (r, b) \to r : g\ f\ b \\ & \qquad\qquad\qquad\qquad\quad Nothing \quad \to [\,] \\ \textbf{in}\ & g\ f\ x \end{aligned}$$

is recognised from the third model solution. To achieve this, we not only recognise the usage of a prelude function, such as *unfoldr*, but also its definition. Furthermore, we apply a number of program transformations to transform a program to a normal form.

Using a *class* a teacher groups together exercises, for example for practicing list problems, collecting exercises of the same difficulty, or exercises from a particular textbook.

3.2 Adapting feedback

A teacher adapts the feedback given to a student by *annotating* model solutions. The description of the entire exercise is given together with the model solutions

in a configuration file for the exercise. Using the following construction we add a *description* to a model solution:

$$\{-\#\ \mathit{DESC}\ \ \mathit{Implement\ range\ using\ the\ unfoldr...}\ \ \#-\}$$

The first hint in Section 2 gives the descriptions for the three model solutions for the range exercise.

A teacher allows an *alternative* implementation for a prelude function by:

$$\{-\#\ \mathit{ALT}\ \ \mathit{iterate}\ f = \mathit{unfoldr}\ (\lambda x \rightarrow \mathit{Just}\ (x, f\ x))\ \ \#-\}$$

Using this annotation we not only recognise the prelude definition ($\mathit{iterate}\ f\ x = x : \mathit{iterate}\ f\ (f\ x)$), but also the alternative implementation given here. Alternatives give the teacher partial control over which program variants are allowed.

A teacher may want to enforce a particular implementation method, for example, use higher-order functions and forbid their explicit recursive definitions, for which we use the *MUSTUSE* construction:

$$\mathit{range}\ x\ y = \{-\#\ \mathit{MUSTUSE}\ \ \#-\}\ \mathit{unfoldr}\ f\ x$$

Specific feedback messages can be attached to particular locations in the source code. For example:

$$\mathit{range}\ x\ y = \{-\#\ \mathit{FEEDBACK}\ \ \mathit{Note...}\ \ \#-\}\ \mathit{take}\ (y - x + 1)\ \$\ \mathit{iterate}\ (+1)\ x$$

Thus we give a detailed description of the *take* function. These feedback messages are organised in a hierarchy based on the abstract syntax tree of the model solution. This enables the teacher to configure the tutor to give feedback messages with an increasing level of detail.

References

1. A. Gerdes, B. Heeren, and J. Jeuring. Teachers and students in charge — using annotated model solutions in a functional programming tutor. In *Proceedings of ECTEL 2012*, 2012.
2. A. Gerdes, B. Heeren, and J. Jeuring. Teachers and students in charge — using annotated model solutions in a functional programming tutor. Technical Report UU-CS-2012-007, Utrecht University, Department of Computer Science, 2012.
3. B. Heeren, D. Leijen, and A. van IJzendoorn. Helium, for learning Haskell. In *Haskell 2003: Proceedings of the 2003 ACM SIGPLAN workshop on Haskell*, pages 62 – 71. ACM, 2003.
4. J. Jeuring, A. Gerdes, and B. Heeren. A programming tutor for Haskell. In *Proceedings of CEFP 2011: Lecture Notes of the Central European School on Functional Programming*, LNCS. Springer, 2012. To appear.
5. S. Peyton Jones et al. *Haskell 98, Language and Libraries. The Revised Report*. Cambridge University Press, 2003. A special issue of the Journal of Functional Programming, see also `http://www.haskell.org/`.
6. K. VanLehn. The behavior of tutoring systems. *International Journal on Artificial Intelligence in Education*, 16(3):227–265, 2006.