

Master Thesis Research Proposal: Embedding Contract Inferencing for Functional Programs In Ask-Elle

Beerend Lauwers
Utrecht University, The Netherlands

February 8, 2013

1 Introduction

Ask-Elle is a programming tutor created by Alex Gerdes [1] for his PhD Thesis. It utilizes (transformations of) model solutions to provide feedback to students on their progress. However, if a student's program cannot be reduced to such a model solution, providing helpful feedback becomes hard to do. The Master Thesis of Jurriën Stutterheim [2] focuses on the development of a contract inferencing system for functional programs, with the goal to embed this functionality in the Ask-Elle programming tutor, so as to provide a new source of meaningful feedback to students using the tutor. In the "Future Work" section of the thesis, Stutterheim notes that the actual integration of the contract inferencing system in Ask-Elle still remains to be done. This research proposal explores what needs to be done in order to attain that goal, as well as the effects it would have upon the tutor's capability to provide meaningful feedback.

2 Research Question

Having embedded contract inferencing in Ask-Elle, does this new functionality provide students with more detailed feedback than without?

3 Main Research Goals

Each goal is expanded upon in the subsections below.

Goal one Embed the contract inferencing system as described in Stutterheim's thesis.

Goal two Perform tests with students to check if the newly-implemented functionality adds value to the provided feedback.

Goal three Analyze the testing results and report our findings.

3.1 Goal One: Embedding the contract inferencing system

Figure 1 depicts an overview of the steps involved in using the contract inferencing system. The blue dashed-dotted arrows and boxes represent functionality that is not yet present in Ask-Elle, but is required for the contract inferencing system to be of use. Let us step through the system and inspect the goals and requirements of each element in more detail.

Currently, if conventional 'diagnosis' of the student program fails, a QuickCheck example is generated, after which it is displayed to the user (this arrow is not shown). Alongside the counterexample, the AST of the student program must be generated such that it can be

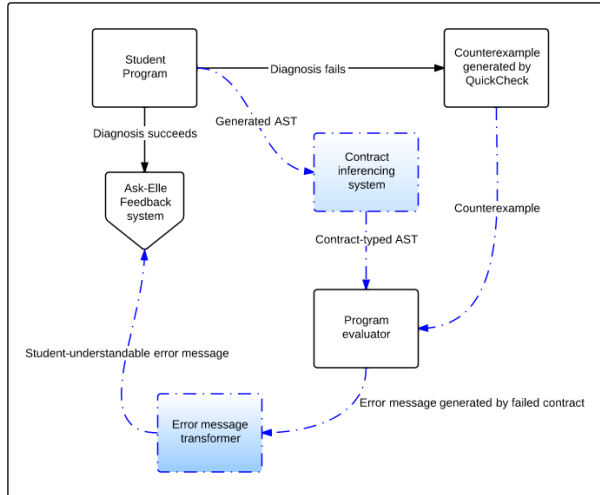


Figure 1: Overview of steps involved in the contract inferencing system.

used by the contract inferencing system. The ASTs used by the Ask-Elle programming tutor and the contract inferencing system differ; Ask-Elle uses Helium as its back-end for Haskell, and Algorithm CW is based upon a small lambda language. So the first step towards embedding the system will be:

Step 1 Modify and extend Algorithm CW to accept the AST generated by Ask-Elle, and embed it in Ask-Elle.

Initial work of this step has already been undertaken by research assistants (**TODO: Who?**). After completing Step 1, it should be possible to infer a contract for any arbitrary AST generated by Ask-Elle.

The next step is to use the contract-typed AST to generate an error message. There are two ways of doing this:

1. Use the QuickCheck-provided counterexample to transform the contract-typed AST directly, and then convert it back to an executable program and run it. We suspect that this functionality will be required for dependent function contracts.

2. Convert the contract-typed AST back to an executable program immediately, and apply the counterexample to it.

Replacing parts of the contract-typed AST with a counterexample was touched upon in Stutterheim’s thesis, but an automated way of determining what should be replaced and where does not seem to be available. Hence, this is the next step:

Step 2 Investigate an automated way of modifying the contract-typed AST with a counterexample, and implement this functionality.

After (not) modifying the contract-typed AST, it must be converted back into an executable program. We assume the Helium-compiler has a way of feeding it an AST and returning an executable program. However, keeping the well-known saying about assumptions in mind, we will have to investigate further:

Step 3 Investigate if it is possible to feed a contract-typed AST to the Helium compiler. If not, investigate if this functionality can be implemented.

If all has gone well, we should end up with an error message generated by the Helium compiler, caused by a contract violation. However, returning this error message to the student will not result in useful feedback. Therefore, we must transform the error message by replacing the details of the contract violation with those of the function wrapped by the violated contract. This transformed error message will be returned to the student. The final step of this subsection, then, is as follows:

Step 4 Implement an error message transformer that "cleans up" the generated error message so it is understandable by the student, and feed it to the Ask-Elle feedback mechanism.

3.2 Goal Two: Perform testing with students

In August, Utrecht University hosts an Applied Functional Programming summer school. This summer school course specifically targets students and participants who are not yet very knowledgeable about functional programming in general, and Haskell in particular. This is an excellent opportunity to test the functionality described in Goal One. Let us again go through each step that must be taken to complete this goal.

The first step is relatively simple, but nonetheless crucial:

Step 1 Organize the practical aspects of the testing on time. Contact the coordinators of the summer school and request a time slot during which testing can take place.

After that, we must provide two versions of the Ask-Elle tutor: one with the new contract inferencing system embedded, and the original, unmodified one. We require two versions to perform a simple A/B test: for each exercise, a version of Ask-Elle is randomly selected and presented to the student. Thus, we require a way to randomly serve one of the versions to a student:

Step 2 Prepare two versions of the Ask-Elle tutor, and set up access in such a way that, for each exercise, one of the versions is randomly served to the student.

After completing an exercise in Ask-Elle (there are around fifteen exercises in total), the student will fill out a very small survey if he or she encountered feedback containing a counterexample. The feedback will be shown above the survey. The following questions will then be visible below the feedback:

For each question, please mark the circle that corresponds with your opinion.

1. The feedback above provided me with sufficient information to help me find the mistake in my code.
[Strongly disagree] 0 - 0 - 0 - 0 - 0 [Strongly agree]
2. The feedback above should indicate a more specific part of the code.
[Strongly disagree] 0 - 0 - 0 - 0 - 0 [Strongly agree]

As this survey is quite small, it could be made multi-lingual if this makes students more inclined to take part in the survey.

With the results of these surveys, we are able to discern which version of Ask-Elle students prefer when they encounter feedback containing a counterexample.

So, our final steps for this goal are:

Step 3 Build a small survey system that acquires any feedback containing a counterexample from the Ask-Elle tutor and presents the survey mentioned above, then writes the answers back to a database.

Step 4 Test the entire system beforehand, and ensure it is operational for use by the students.

Step 5 Collect survey results.

3.3 Goal Three: Analyze the testing results and report findings

Having collected the survey results, all that remains to be done is to analyze them and determine if the embedding of the contract inferencing system in Ask-Elle has increased the usefulness of feedback containing a counterexample provided to students, or not:

Step 1 Analyze the survey results to answer the research question.

After processing the results, we will present our findings in a presentation, as is customary:

Step 2 Report findings and problems encountered / lessons learned during this thesis in a presentation.

4 Stretch Goals

In case the goals described in section 3 require less time than was planned, there are a few stretch goals available:

4.1 Goal One Stretch Goals

- Ask-Elle’s normal analysis may fail because of a multitude of reasons. Eta-expansion, commutativity, associativity and many other transformations that do not change the meaning of the program are currently unsupported by Ask-Elle. Collecting and categorizing these problematic transformations would be great source of examples for future work in supporting these transformations.
- At the moment, there are only counterexamples available for Haskell, generated by QuickCheck. However, Ask-Elle focuses on functional languages in general. Generalizing the counterexample and contract inferencing interfaces may not be too big of a hassle and prove to be an ideal foundation for further future work.
- Building further upon the previous stretch goal, a toy implementation of another functional language in the tutor is a final ambitious stretch goal that could help lay bare any existing issues with Ask-Elle’s architecture. There are multiple functional candidates available, such as Scheme, Erlang and Clojure, all of which are quite different from Haskell. As an added bonus,

there exist QuickCheck ports for all of these languages.

4.2 Goal Two Stretch Goals

- It may be of interest to extend the survey presented to students (or present a larger survey at the end of the exercise set) to learn more about how the students perceived Ask-Elle’s performance. However, this could increase the time needed for testing, and it remains to be seen if this extra time can be made available, if at all. A starting point for a larger survey could be the original survey in Alex Gerdes’ thesis [1].
- In case testing during the summer school course provides insufficient data (or data was unable to be collected for whatever reason), it may be desirable to perform additional testing. As Ask-Elle is designed to work from a web browser, testing and survey taking can be done remotely; identifying and contacting students interested in functional programming (inside and outside the university) may prove to be a valuable source of potential survey takers.

5 Acknowledgements

I would like to thank Johan Jeuring and Wouter Swierstra for their master thesis suggestions and for helping me delineate the initial research goals of this thesis proposal.

References

- [1] Alex Gerdes. *Ask-Elle: a Haskell Tutor*. PhD thesis, Open Universiteit, November 2012.
- [2] Jurriën Stutterheim. Contract inferencing

for functional programs. Master's thesis,
Utrecht University, January 2013.