Reasoning About DrScheme Programs in ACL2

Melissa Weiderrecht and
Christopher MacLellan

Advisor: Ruben Gamboa

Department of Computer Science
University of Wyoming
Laramie, WY
April 24, 2010

Background: The “How to Design Programs” (HTDP) Curriculum

- Includes several languages built into Dr. Scheme
- Minimalist programming language philosophy
- Progressively more expressive languages that match programmer experience
- “Cookie cutter” code templates

Background: The Problem

- Sometimes a student needs even more help than this.
  - Need to learn to reason about programs
  - Need to know why a program in correct or incorrect
- Maybe we can help?

Background: Dr. Scheme the solution?

- Dr. Scheme is designed specifically for beginning programmers.
- However, it does not possess any tools for reasoning about correctness of programs.

Background: ACL2 the solution?

- ACL2 is a mechanical theorem prover which reasons about the correctness of programs.
- Very complex and difficult for a beginning programmer to use
- One simple reason (of many):
  - ACL2 accepts only total functions

Background: Factorial Example

- (fact n) = (if (= n 0)

  1

  (∗ n (fact (− n 1)))))

- Negative values of n?
- Strings values of n?

Background: The Ultimate Goal

- The ultimate goal is to build a pedagogical theorem prover from scratch in Scheme
- Built into Dr. Scheme
- This theorem prover should be fully automated (i.e. Not requiring any help from the students)
  - Possible because of ”cookie cutter” approach

Our Project: A First Step

- Currently no theorem prover exists in the Scheme language
- A theorem prover in Scheme would require a different way of reasoning than, say, ACL2
  - Rewriting instead of Idealized Functions

Our Project: A First Step (cont'd)

- We merged ACL2 with the Beginning Student Language (BSL) from Dr. Scheme
- i.e. we wrote an interpreter for BSL in ACL2
- Proved theorems about BSL programs in ACL2
  - e.g. (bsl '(factorial 4)) ≠ error

First Iteration

- Consisted of several mutually recursive functions
- bsl: evaluate a scheme expression
- bsl-list: evaluate a list of scheme expressions (e.g. a list of function arguments)
- bsl-builtin: evaluate an expression that matches a form built into BSL
- bsl-userdefined: evaluate an expression that is defined in the current BSL environment

First Iteration Problems

- ACL2 generally does not open up recursive definitions, because it would never know when to stop expanding.
- We happened to want to expand the recursive definitions, which became tedious when working with the builtin functions
- None of our theorems were being used as rewrite rules because they were written in the negative form (i.e. bsl does not return error).

Second Iteration

- Rewrote all of our theorems in the positive form so they would be used as rewrite rules.
  - bsl returns *success* or *timeout*
  - when successful, the result is correct
- Proved theorems about the builtin functions
- Moved on to proving theorems about bsl programs (e.g. Sum_up_to(n) and factorial(n)).

Second Iteration Problems

- With our rewrite rules being used we no longer had to expand every single recursive function, but we still had to expand many of them.
- ACL2 struggled with mutually recursive functions, essentially hindering further progress in our theorems.

Third Iteration

- Started over from scratch
- Combined our mutually recursive functions into one singlely recursive function named bsl (which took an additional flag to set whether it evaluates an expression or a program).
- We were able to prove partial correctness of programs such as sum_up_to(n).

Conclusions

- As it stands, ACL2 (our intrepreter) can be used to reason about simple functions written in BSL, but it is still quite difficult.
- We are convinced that the inference rules and strategies required to perform such reasoning can be fully automated, at least for simple programs, such as the ones beginners are likely to write in BSL.

The Icing

- We submitted a paper describing our work to the conference ”Trends in Functional Programming 2010” and we got accepted.
- We plan to attend the conference in May and present our paper.