Modular ACL2

Carl Eastlund
Northeastern University
Boston, MA, USA
cce@ccs.neu.edu

Matthias Felleisen
Northeastern University
Boston, MA, USA
matthias@ccs.neu.edu

In the early 1980s, Boyer and Moore decided to re-build their Nqthm theorem prover [1] for a first-order, functional subset of a standardized, industrial programming language: Common Lisp [8]. The resulting system, ACL2, was an attempt to piggy-back theorem proving on the expected success of Lisp and functional programming. Although Common Lisp didn't succeed, ACL2 became the most widely used theorem prover in industry. Over the past 20 years, numerous hardware and software companies turned to ACL2 to verify critical pieces of their products [5]; by 2006, their contributions to the ACL2 regression test suite exceeded one million lines of code. The ACL2 team received the 2005 ACM Systems Award for their achievement.[1]

During those 20 years, programming language theory and practice also evolved. In particular, programming language designers have designed, implemented, and experimented with numerous module systems for managing large functional programs [4]. One major goal of these design efforts has been to help programmers reason locally about their code. That is, a module should express its expectations about imports, and all verification efforts for definitions in a module should be conducted with respect to these expectations. Common Lisp and thus ACL2, however, lack a proper module system. Instead, ACL2 programmers use Common Lisp's package system and ad hoc tools for proof encapsulation and instantiation, plus usage patterns that mimic modular programming. Naturally, the manual maintenance of abstraction boundaries is difficult and error prone. Worse, it forces the programmer to choose between local reasoning and end-to-end execution, as functions defined in an encapsulated proof cannot be run.

Since manual programming patterns are laborious and error-prone, we have developed Modular ACL2, an extension of ACL2 with a module system. This new language is implemented on top of Dracula [2], our dialect of ACL2 [9]. The module system takes its inspiration from the PLT Scheme unit system [3, 7]; it separates modules from interfaces and introduces an external linking language to combine client modules with provider modules. Our language of interfaces allows hiding some functions for abstraction, exposing others to express new forms of inductive reasoning, and constraints to describe functions shared across multiple interfaces. Naturally we impose enough restrictions to ensure the soundness of ACL2, which assumes a first-order, terminating programming language. In the vein of previous extensions to the theorem prover [6], we supply a formal proof of correctness for Modular ACL2.

Our research includes a number of benchmarks, i.e., attempts to turn monolithic programs into modular systems and to measure the effect on theorem-proving time. With our module system, the theorem prover performs remarkably well; introducing modules reduces the amount of time spent by the theorem prover searching for a proof, sometimes by several orders of magnitude.

[1] `http://campus.acm.org/public/pressroom/press_releases/3_2006/software.cfm`

1. REFERENCES

[1] Boyer, R. S. and J S. Moore. Mechanized reasoning about programs and computing machines. In Veroff, R., editor, *Automated Reasoning and Its Applications: Essays in Honor of Larry Wos*, p. 146–176. MIT Press, 1996.

[2] Eastlund, C. and M. Felleisen. Toward a practical module system for ACL2. In *PADL*, p. 46–60, 2009.

[3] Flatt, M. and M. Felleisen. Units: Cool modules for HOT languages. In *PLDI*, p. 236–248, 1998.

[4] Harper, R. and B. C. Pierce. Design issues in advanced module systems. In Pierce, B. C., editor, *Advanced Topics in Types and Programming Languages*. MIT Press, 2004. 293–345.

[5] Kaufmann, M., P. Manolios and J. S. Moore. *Computer-Aided Reasoning: ACL2 Case Studies*. Kluwer, 2000.

[6] Kaufmann, M. and J. S. Moore. Structured theory development for a mechanized logic. *Journal of Automated Reasoning*, 26(2):161–203, February 2001.

[7] Owens, S. and M. Flatt. From structures and functors to modules and units. In *ICFP*, p. 87–98, 2006.

[8] Steele Jr., G. *Common Lisp—The Language*. Digital Press, 1984.

[9] Vaillancourt, D., R. Page and M. Felleisen. ACL2 in DrScheme. In *ACL2 Workshop*, p. 107–116, 2006.