Available online at www.sciencedirect.com
ScienceDirect
Journal of Applied Logic 7 (2009) 3–25
JOURNAL OF APPLIED LOGIC
www.elsevier.com/locate/jal

Integrating external deduction tools with ACL2 ☆,☆☆

Matt Kaufmann *, J Strother Moore, Sandip Ray *, Erik Reeber *

Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712-0233, USA

Available online 3 August 2007

Abstract

We present an interface connecting the ACL2 theorem prover with external deduction tools. The ACL2 logic contains several mechanisms for proof structuring, which are important to the construction of industrial-scale proofs. The complexity induced by these mechanisms makes the design of the interface challenging. We discuss some of the challenges, and develop a precise specification of the requirements on the external tools for a sound connection with ACL2. We also develop constructs within ACL2 to enable the developers of external tools to satisfy our specifications. The interface is available with the ACL2 theorem prover starting from Version 3.2, and we describe several applications of the interface.

Keywords: Automated reasoning; Decision procedures; First-order logic; Interfaces; Theorem proving

1. Introduction

Recent years have seen rapid advancement in the capacity of automatic reasoning tools, in particular for decidable theories such as Boolean logic and Presburger arithmetic. For instance, modern BDD packages and satisfiability solvers can automatically solve problems with tens of thousands of variables and have been successfully used to reason about commercial hardware system implementations [2,3]. This advancement has sparked significant interest in the general-purpose mechanized theorem proving community, to improve the efficiency of theorem provers by developing connections with automatic reasoning tools. In this paper, we present a general interface for connecting the ACL2 theorem prover [4,5] with tools that are external to ACL2's built-in reasoning routines.

ACL2 consists of a functional programming interface based on Common Lisp [6], along with a first-order interactive theorem prover. The ACL2 theorem prover supports several deduction mechanisms such as congruence-based

☆ A preliminary version of this paper [M. Kaufmann, J S. Moore, S. Ray, E. Reeber, Integrating external deduction tools with ACL2, in: C. Benzmüller, B. Fischer, G. Sutcliffe (Eds.), Proceedings of the 6th International Workshop on Implementation of Logics (IWIL 2006), CEUR Workshop Proceedings, vol. 212, 2006, pp. 7–26. [1]] appeared in the 6th International Workshop for Implementation of Logics (Phnom Penh, Cambodia, November 2006).

☆☆ This material is based upon work supported by DARPA and the National Science Foundation under Grant No. CNS-0429591, by the National Science Foundation under Grant No. ISS-0417413, and by DARPA under Contract No. NBCH30390004.

* Corresponding authors.
E-mail addresses: kaufmann@cs.utexas.edu (M. Kaufmann), moore@cs.utexas.edu (J S. Moore), sandip@cs.utexas.edu (S. Ray), reeber@cs.utexas.edu (E. Reeber).
URLs: http://www.cs.utexas.edu/users/kaufmann (M. Kaufmann), http://www.cs.utexas.edu/users/moore (J S. Moore), http://www.cs.utexas.edu/users/sandip (S. Ray), http://www.cs.utexas.edu/users/reeber (E. Reeber).

1570-8683/$ – see front matter
doi:10.1016/j.jal.2007.07.002

conditional rewriting, well-founded induction, several integrated decision procedures, and generalization. ACL2 has been particularly successful in the verification of microprocessors and hardware designs, such as the floating point multiplication, division, and square root algorithms of AMD processors [7–10], microcode for the Motorola CAP DSP [11], and separation properties of the Rockwell Collins AAMP7™ processor [12]. However, the applicability of ACL2 (as that of any theorem prover) is often limited by the amount of user expertise necessary to drive the theorem prover. Indeed, each of the above projects represents many man-years of effort. Yet, many of the necessary lemmas, for instance those establishing hardware invariants, can be expressed in a decidable theory and dispatched by a decision procedure.

On the other hand, it is non-trivial to establish a sound connection between ACL2 and other tools. ACL2 contains several logical constructs intended to facilitate effective proof structuring [13]. These constructs are crucial to the applicability of ACL2 in large-scale verification projects; however, they complicate the logical foundations of the theorem prover. To facilitate connection between another tool and ACL2, it is therefore imperative (i) to determine the conditions under which a conjecture certified by a combination of the theorem prover and the tool is indeed a theorem, and (ii) to provide mechanisms that enable a tool implementor to meet these conditions.

The interface described in this paper enables the connection of ACL2 with other reasoning tools. In particular, it permits an ACL2 user to invoke an external tool to reduce a goal formula C to a list of formulas L_C during a proof attempt. Correctness of the tool involves showing that the provability of each formula in L_C (in the logic of ACL2) implies the provability of C. We present a sufficient condition (expressible in ACL2) that guarantees such provability claims, and discuss the logical requirements on the implementor of external tools for sound connection with ACL2. The interface design illustrates some of the subtleties and corner cases that need to be considered in augmenting an industrial-strength formal tool with a non-trivial feature.

We distinguish between two classes of external tools, namely (i) tools verified by the ACL2 theorem prover, and (ii) unverified but trusted tools. A verified tool must be formalized in the logic of ACL2 and the sufficient condition alluded to above must be formally established by the theorem prover. An unverified tool can be defined using the ACL2 programming interface, and can invoke arbitrary executable programs via a system call interface. An unverified tool is introduced with a *trust tag* acknowledging that the validity of the formulas proven using the tool depends on the correctness of the tool.

The connection with unverified tools enables us to invoke external SAT solvers, BDD packages, and so on, for simplifying ACL2 subgoals. Why might one use verified tools? The formal language of ACL2 is a programming language, based on an applicative subset of Common Lisp. The close relation between ACL2 and Lisp makes it possible to write efficient programs in the ACL2 logic [6]. Indeed, most of the source code implementing the theorem prover is written in this language. It can therefore be handy for the ACL2 user to control proofs by (i) implementing customized reasoning code, (ii) verifying such code with ACL2, and (iii) invoking the code for proving theorems in a specific domain. In fact, ACL2 currently provides a way for users to augment its built-in term simplifier with their own customized reasoning code, via the so-called "meta rules" [14]. However, such rules essentially augment ACL2's term simplifier without providing a way to manipulate directly the entirety of a subgoal generated during a proof. Furthermore, meta rules can only simplify a term to one that is provably equivalent; that is, they do not allow generalization. The connection with verified tools supports direct invocation of customized, provably correct, reasoning code for reducing a conjecture to a collection of (possibly more general) subgoals.

The rest of the paper is organized as follows. In Section 2 we provide a brief review of ACL2; this section is intended to provide an overview of the facets of ACL2 that are relevant to the subsequent discussions, and can be skipped by readers familiar with the theorem prover without loss of continuity. In Sections 3–5 we present the interface for connecting external tools to ACL2, the logical requirements for the developer of such connections, and the necessary augmentations required to support the interface. In Section 6, we provide a few remarks on our implementation. We discuss related work in Section 7, and conclude in Section 8.

The interface described in this paper is available with ACL2 Version 3.2, and ACL2's hypertext documentation includes a topic, clause-processor, which provides further details for many of its features. In addition, the ACL2 distribution contains a directory `books/clause-processors/`, with proof scripts demonstrating many applications of the interface.