Journal of Automated Reasoning manuscript No.
(will be inserted by the editor)

The Right Tools for the Job: Correctness of Cone of Influence Reduction Proved Using ACL2 and HOL4

Michael J. C. Gordon · Matt Kaufmann · Sandip Ray

Received: date / Accepted: date

Abstract We present a case study illustrating how to exploit the expressive power of higher-order logic to complete a proof whose main lemma is already proved in a first-order theorem prover. Our proof exploits a link between the HOL4 and ACL2 proof systems to show correctness of a cone of influence reduction algorithm, implemented in ACL2, with respect to the classical semantics of linear temporal logic, formalized in HOL4.

Keywords Theorem proving · Automated reasoning · Higher-order logic · Combining theorem provers · Linear temporal logic · Model checking

1 Introduction

We present a case study demonstrating the cooperative application of two interactive theorem provers, HOL4 [18,21] and ACL2 [15,14,11,10]. The linkage of different proof tools can benefit large-scale proof projects, because such tools often have complementary strengths and largely disjoint user communities. In our study, the main theorem is proved by reusing proof components from an existing ACL2 proof while exploiting the expressive logic of HOL4 to handle some key concepts that were problematic for ACL2's first order logic.

Our case study is the proof of correctness of cone of influence reduction. Cone of influence reduction, also referred to as *slicing* [7] or *localization reduction* [16], is a fundamental simplification technique employed in model checking [1]. Consider the use of model checking to determine if a design model (or *Kripke structure*) $\mathcal{M}$ satisfies

M. J. C. Gordon
Computer Laboratory
William Gates Building, 15 JJ Thomson Ave. Cambridge CB3 0FD, United Kingdom
Tel.: +44-1223-334627 Fax: +44-1223-334678
E-mail: Mike.Gordon@cl.cam.ac.uk

M. Kaufmann and S. Ray
Department of Computer Sciences
TAY 2.124, C0500, The Univ. of Texas at Austin, Austin, TX 78712-1188, USA
Tel.: 512-471-7316 Fax: 512-471-8885
E-mail: {kaufmann,sandip}@cs.utexas.edu

a temporal logic formula φ. Then roughly, cone of influence reduction constructs an abstraction $\mathcal{M}'$ of $\mathcal{M}$ by getting rid of state variables that are irrelevant to φ. Given the semantics of the temporal logic in which φ is specified, the correctness statement of cone of influence reduction is that $\mathcal{M}'$ satisfies φ if and only if M does so as well. Our work is a mechanically checked formalization of this statement where φ is a formula in a Linear Temporal Logic (LTL). A key aspect of the proof is the combination of formal tools employed: the semantics of LTL are formalized in higher-order logic (HOL) [4]; the cone of influence reduction algorithm is implemented in the logic of the ACL2 theorem prover [15, 14, 11, 10]; and the proof is completed in the HOL4 theorem prover using a main lemma proved with ACL2 and exported to HOL4, using a previously developed generic link between the two theorem provers.

Ray, Matthews, and Tuttle [19] carried out a formalization and verification of the cone of influence reduction algorithm in ACL2, as part of a project to develop high-assurance reduction algorithms for LTL model checking. The reduction algorithms were intended to be efficiently executable on concrete design models in addition to being formally verified. ACL2 was used in the project to exploit its strong support for efficient executability of formal definitions. However, the restricted expressive power of ACL2 injected significant complexity into the proof. In particular, the logic of ACL2 is a first-order logic of finite objects, but the standard semantics of LTL involve quantification over infinite paths of execution. Consequently, the ACL2 formulation of LTL semantics had to "cheat" by considering only finitely represented execution paths that are *eventually periodic*, *i.e.*, consist of an initial prefix followed by a cycle which is repeated forever. The equivalence between the standard semantics and the semantics based on eventually periodic paths is a classical result on LTL. Nevertheless, the use of eventually periodic paths in the formalization is unsatisfying for two reasons. First, it is a nonstandard formalization of LTL, and indeed the proof connecting this formalization with the standard LTL semantics is non-trivial (albeit well-known). Secondly, and perhaps more importantly, the use of eventually periodic paths significantly complicates the proofs of reduction algorithms. The standard proof of correctness of the cone of influence reduction algorithm proceeds by showing the following two propositions:

1. The reduced model $\mathcal{M}'$ is bisimilar to the original model $\mathcal{M}$.
2. If two models $\mathcal{M}$ and $\mathcal{M}'$ are bisimilar then $\mathcal{M}$ satisfies an LTL formula φ if and only if $\mathcal{M}'$ does as well.

The use of eventually periodic paths significantly complicates the proof of Proposition 2 which connects the notion of bisimulation of models with the semantics of LTL.

On the other hand, higher-order logic is well-suited to formalizing the standard semantics of LTL and carrying out the proof of Proposition 2. The connection between ACL2 and HOL permits the verification of the executable ACL2 definition of the algorithm based on the semantics defined in HOL, exploiting the existing ACL2 proof of Proposition 1 with a standard proof of Proposition 2 formalized in HOL4. The result thus demonstrates how a combination of different general-purpose theorem proving systems with complementary strengths can be gainfully used to derive an interesting formal proof.

The remainder of the paper is organized as follows. Section 2 begins with brief background on the ACL2 and HOL4 logics and systems, and on the link connecting them. Then Section 3 formalizes the main result, followed by an outline of the proof using the ACL2/HOL4 link in Section 4. We conclude with Section 5.

2 Background: ACL2, HOL4, and Their Connection

The ACL2 logic [12,13] is a first-order logic with built-in data types and axioms for numbers, characters, strings, symbols, and finite trees built from the `cons` constructor. The logic provides *extension principles* to extend a theory with new first-order axioms. The extension principles include (1) a *definitional principle* for introducing (recursive) total functions, (2) an *encapsulation principle* for introducing partially defined (or constrained) functions, and (3) a *defchoose principle* for introducing Skolem functions. The application of each extension principle generates proof obligations: for example, introducing a new recursive definition requires a proof that the recursion terminates. The proof obligations guarantee that the extended theory provides a conservative extension of the current theory.

The ACL2 system supports a syntax of terms, which however can be interpreted as formulas: when a term τ is used where a formula is expected, it stands for the formula $(\tau \neq \texttt{NIL})$, where the term `NIL` represents Boolean falsity. Thus, a term is considered to be a theorem if it is provably not `NIL`. We freely use `NIL` both in our ACL2 models and theorems to model Boolean falsity; following standard convention, we also use the symbol `T` to represent Boolean truth.

The Compactness Theorem of first-order logic provides *nonstandard models* of arithmetic, and hence of the ACL2 logic. (See any textbook on first-order logic, for instance Shoenfield [20], for an exposition of the Compactness Theorem.) We are interested in models whose universe is the *standard universe*, containing roughly those objects on which one can actually compute in the ACL2 system (and which thus excludes, among other objects, nonstandard integers).[1] By the Soundness Theorem of first-order logic, all theorems are true in the standard model of the current set of definitions.

The HOL4 system is an LCF-style implementation of a version of higher-order logic based on Church's simply typed lambda calculus augmented with with Hindley-Milner polymorphism [5]. It is sufficiently powerful to allow function definitions through recursion, analogous to the way that is allowed by ACL2.

The specifications and proofs shown in this paper make use of a previously established link between ACL2 and HOL4 [2,3]. The link is implemented by treating ACL2 as a trusted external oracle of HOL4. Logically, a theorem proved by ACL2 is treated as a theorem about the HOL datatype `sexp` shown below, which provides a model of the standard ACL2 universe. The definition uses previously defined HOL types `packagename`, `name`, `string`, `char` and `complex_rational`.

```
sexp = ACL2_SYMBOL    of packagename => name
     | ACL2_STRING    of string
     | ACL2_CHARACTER of char
     | ACL2_NUMBER    of complex_rational
     | ACL2_PAIR      of sexp => sexp
```

Each constructor introduced above returns an object of type `sexp`, with argument types as indicated after the keyword "`of`". For example, `ACL2_SYMBOL` is a (curried) constructor of type `packagename -> name -> sexp`. When function symbols of ACL2 are translated into HOL4, the resulting function symbols are curried, and all domain

[1] We say "roughly" since the model being referred to here contains certain objects formed by applying the free constructor for symbols to inappropriate arguments. These objects are referred to as *bad atoms* in ACL2, and cannot be used in computation. The reason for the standard model to include the bad atoms is technical and not relevant to this exposition.

and range types are `sexp`. Names are modified as necessary; for example, the hyphen character (`-`) — which is illegal in HOL4 identifier names — is replaced by underscore (_) , and `>` is replaced by `greater`. The logical soundness of the link above is based on the observation that any theorem proved in an ACL2 theory T holds in the standard model of T, and hence in the HOL theory obtained by extending `sexp` with the (suitably translated) definitions from T. The link operates by mechanically importing into HOL both built-in ACL2 axioms and user-supplied definitions and theorems, tagging each using HOL's oracle mechanism.

We end this section with a brief note on our notational conventions. Note that the formalization and proof discussed here involve two theorem provers with very different formula syntax: the syntax of ACL2 is essentially that of Lisp while HOL uses a higher-order ML-like syntax. Since our goal is to explain how the two systems work together, we adhere to the formal syntax of each system as much as possible in the presentation. In particular, we display ACL2 formulas in the concrete formal syntax, accompanied by paraphrasing and explanatory notes whenever appropriate. The presentation of HOL formulas employs the formal syntax of HOL4, but the formulas are lightly edited to improve readability (*e.g.*, by replacing the HOL logical symbols "`|=`", "`!`", "`?`", "`\`", "`/\`", "`\/`", "`==>`", "`~`" by more standard notations "$\models$", "$\forall$", "$\exists$", "λ", "$\wedge$", "$\vee$", "$\Rightarrow$", "$\neg$", respectively). Our full formalization is publicly available in the HOL repository [18] in the directory `examples/acl2/examples/acl2-hol-ltl-paper-example/`, and on the Web.[2] Theorems proved with ACL2 are shown with ACL2's `defthm` command; thus if `foo` is the name of the theorem that proves the formula τ, we will display it as follows.

```
(defthm foo
  τ)
```

We will subsequently refer to the theorem with the name `foo`. Correspondingly, in this paper we attach names to theorems proved in HOL4, as follows:

```
Theorem foo
    τ
```

We take the liberty of using theorem names in the paper that sometimes differ from the names in the proof scripts. In some cases, when we do not need to refer later by name to the formula τ, we only display the formula without labeling it with a theorem name. We ignore directives to both theorem provers (tactics, hints, rule classes, etc.), unless they are relevant for pedagogical purposes.

3 Problem formalization

We start by providing an informal overview of the concepts involved in the definition of cone of influence reduction. These concepts are formalized in ACL2 and are described more thoroughly in the previous paper by Ray, Matthews, and Tuttle [19]. In the descriptions below, we think of the *variables* as state elements of a circuit. If V_C is the set of variables of a circuit C, then a *state of* C is a mapping that associates each variable in V_C with a member of the set $\{\texttt{T}, \texttt{NIL}\}$.

– A *circuit* C is a data structure containing three fields: (1) a list V_C of *variables*, (2) a mapping associating each variable $v \in V_C$ with a *next-state equation*, and (3) a set

[2] See URL `http://hol.svn.sourceforge.net/viewvc/hol/HOL/examples/acl2/examples/acl2-hol-ltl-paper-example/`.

of states of C, called *initial states*. The next-state equation for v is a term which defines the value of v after one transition of the circuit in terms of the values of the variables in V_C in the current state. We define a predicate `circuitp` in ACL2, such that `(circuitp C)` returns `T` if and only if `C` is a syntactically well-formed circuit, and `NIL` otherwise.

- The *Kripke structure for* a circuit C is a Kripke structure with the same set of initial states as C, and whose transition relation is constructed in a straightforward manner from the next-state equations of C. In ACL2, we define a predicate `circuit-modelp` to recognize well-formed Kripke structures, and a transformation function `create-kripke` to transform a circuit into a Kripke structure.
- Let C be a circuit and V be a set of variables. Let $V' \triangleq V \cap V_C$. Then the *cone variables* of C with respect to V is the least set V_f such that (1) $V' \subseteq V_f$, (2) $V_f \subseteq V_C$, and (3) for each variable in V_f, all variables occurring in its associated equation are elements of V_f. Given a circuit `C` and a list of variables `vars`, the function `cone-vars` takes a circuit C and a set of variables V, and returns the cone variables of C with respect to V.
- By *cone of influence reduction* of a circuit C with respect to a set V of variables, we mean the following operation. First, obtain the cone variables V_f of C with respect to V. Then restrict each initial state of C to the variables in V_f, and restrict the equations of C to those defining a value for a variable in V_f. The circuit obtained by cone of influence reduction is called the *reduction of C with V*. The ACL2 function `cone-of-influence-reduction` takes a circuit C and a set V of variables, and generates the reduction of C with V.

We now sketch some of the HOL definitions necessary to state the correctness theorem for cone of influence reduction. First, we define a datatype for LTL formulas and translate the ACL2 definition of a Kripke structure into a definition suited naturally to higher-order logic. Below, the function denoted by $\models$ is used to map objects of type `sexp` to HOL Booleans: for an element x of `sexp`, $\models x$ holds if x is not `NIL`.[3] The recursive definition of the HOL logic datatype `formula` is shown below. This creates a type `('prop)formula` that is parameterized on a type variable `'prop` that can be instantiated to particular types of atomic propositions (it will be instantiated to `sexp`).

```
formula = TRUE
        | FALSE
        | ATOMIC     of 'prop
        | NOT        of formula
        | AND        of formula => formula
        | OR         of formula => formula
        | SOMETIMES  of formula
        | ALWAYS     of formula
        | NEXT       of formula
        | UNTIL      of formula => formula
        | WEAK_UNTIL of formula => formula
```

A key component in the formulation of the correctness theorem for cone of influence reduction is the definition of LTL semantics, which is formalized in HOL. The

[3] Here, by `NIL` we mean the element of the `sexp` datatype corresponding to the ACL2 object, `NIL`.

definition of LTL semantics is specified in terms of paths through a Kripke structure. Kripke structures are represented in the HOL logic as polymorphic terms of type `('prop,'state)model`, where the type variables `'prop` and `'state` can be instantiated to specific types for particular applications (for our application they are both instantiated to `sexp` — see the definition of `HOL_MODEL` below). Values of type `('prop,'state)model` are records with fields `S` (states), `S0` (initial states), `R` (transition relation) and `L` (state labelling function). The HOL4 syntax for the definition of the type `model` of Kripke structures is shown below.

```
model =
 <| S: 'state set;
    S0:'state set ;
    R: ('state # 'state) set;
    L: 'state -> 'prop set
 |>
```

If `M` is a term with type `('prop,'state)model` then `M.S`, `M.S0`, `M.R`, `M.L` are terms denoting the corresponding components; they have the types shown in the definition of `model`, e.g. `M.S` has type `'state set`, which represents a set of states, where sets are represented by their characteristic functions. We require a model `M` to satisfy two properties: the set of initial states is a subset of the set of all states (`M.S0 SUBSET M.S` in HOL notation), and if `s` is a state in `M.S` and `s'` is a successor state according to the transition relation `M.R`, then `s'` is also in `M.S`. These two requirements are represented by a predicate `MODEL` defined by:

```
MODEL M ⇔
M.S0 SUBSET M.S ∧ ∀s s'. s IN M.S ∧ (s,s') IN M.R ⇒ s' IN M.S
```

The function `HOL_MODEL` below maps an ACL2 representation of a Kripke structure to a HOL Kripke structure. In HOL, a term `<| S:=`t_1`; S0 :=`t_2`; R :=`t_3`; L :=`t_4 `|>` constructs a record value whose components are specified by the terms t_1, t_2, t_3 and t_4. The type of `HOL_MODEL` (as inferred by the HOL typechecker) is `sexp->(sexp,sexp)model`. The function `ksym` maps a string to a keyword of that name (i.e., an object of type `sexp` representing a symbol in the `"KEYWORD"` package), and the function `g` ("get") is defined by importing function `g` from ACL2, defined to return the binding of a key in a finite mapping.

```
HOL_MODEL sexp_model =
 <| S  := λs.     |= memberp s (g (ksym "STATES") sexp_model);
    S0 := λs.     |= memberp s (g (ksym "INITIAL-STATES") sexp_model);
    R  := λ(p,q). |= (next_statep p q sexp_model);
    L  := λs a.   |= memberp a (label_of s sexp_model)
 |>
```

Recall that the ACL2 function `create-kripke` translates a circuit description to a Kripke structure (in ACL2). This function is translated through the ACL2/HOL interface to the HOL function `create_kripke` of type `sexp->sexp`. An `sexp`-representation of a Kripke structure in HOL can then be converted to a HOL Kripke structure using `HOL_MODEL`. The composition of these functions (one imported from ACL2, one defined directly in HOL) is defined to be `CIRC_TO_MODEL`:

```
CIRC_TO_MODEL C = HOL_MODEL (create_kripke C)
```

```
(SEM M p TRUE = T)
∧
(SEM M p FALSE = F)
∧
(SEM M p (ATOMIC a) = M.L (p 0) a)
∧
(SEM M p (NOT f) = ¬(SEM M p f))
∧
(SEM M p (AND f1 f2) = SEM M p f1 ∧ SEM M p f2)
∧
(SEM M p (OR f1 f2) = SEM M p f1 ∨ SEM M p f2)
∧
(SEM M p (SOMETIMES f) = ∃i. SEM M (SUFFIX p i) f)
∧
(SEM M p (ALWAYS f) = ∀i. SEM M (SUFFIX p i) f)
∧
(SEM M p (NEXT f) = SEM M (SUFFIX p 1) f)
∧
(SEM M p (UNTIL f1 f2) =
   ∃i. SEM M (SUFFIX p i) f2 ∧ ∀j. j < i ⇒ SEM M (SUFFIX p j) f1)
∧
(SEM M p (WEAK_UNTIL f1 f2) =
  (∃i. SEM M (SUFFIX p i) f2 ∧ ∀j. j < i ⇒ SEM M (SUFFIX p j) f1)
  ∨
  ∀i. SEM M (SUFFIX p i) f1)
```

Fig. 1 HOL formalization of the semantics of LTL.

We now discuss the HOL formalization of the LTL semantics. For this, we need two notions, *e.g.*, (1) *path* through a Kripke structure, and (2) *suffix* of a path. These two definitions are shown below. The path through a Kripke structure M is simply modeled as a function defined on the natural numbers that respects the next-state relation of M. The predicate `PATH` tests that a path function `p` is a path starting from state `s` in a model `M` and for a general Kripke structure model has the rather complicated type: `('prop,'state)model -> 'state -> (num -> 'state) -> bool`.

```
PATH M s p = (p 0 = s) ∧ ∀i. M.R(p(i),p(i+1))
SUFFIX p i = λj. p(i+j)
```

The truth function `SEM` for a given model `M` and path `p` is shown in Fig. 1 and has the type: `('prop,'state)model -> (num -> 'state) -> 'prop formula -> bool`. The recursive definition is by cases on LTL formulas and is very similar to standard textbook definitions [1,9]. The corresponding notion `SAT`, that quantifies over all paths starting from initial states, has the type: `('prop,'state)model -> 'prop formula -> bool`.

```
SAT M f = ∀p. (p 0) IN M.S0 ∧ PATH M (p 0) p ⇒ SEM M p f
```

We now state our main theorem, which asserts that for a formula `f` using at most a set `FVars` of variables contained in the cone of influence of a set `Vars` of variables, the satisfaction (`SAT`) of that formula at any path is preserved under the cone of influence reduction with respect to `Vars`.

```
Theorem correctness-of-reduction

  ∀C Vars FVars.
  (⊨ circuitp C) ∧
  (⊨ subset FVars (cone_variables Vars C))
  ⇒
  ∀f. (Atoms f) SUBSET (SexpToSet FVars)
      ⇒
      (SAT (CIRC_TO_MODEL C) f =
       SAT (CIRC_TO_MODEL
             (cone_of_influence_reduction C Vars))
           f)
```

Here, `circuitp`, `cone_variables`, and `cone_of_influence_reduction` are first-order functions on the type `sexp`; their definitions are imported from the corresponding ACL2 functions. The HOL function `Atoms` is a straightforward recursive function that extracts the set of variables in an LTL formula (represented in `sexp`). The function `SexpToSet` maps an ACL2 list (represented in `sexp`) to a Boolean-valued function returning `T` exactly on members of that list. The functions `subset` and `SUBSET` are, respectively, the subset relation on lists in ACL2 (represented in `sexp`) and the subset relation in HOL.

4 The Proof

Our goal is to do the proof in two parts, which correspond to the two propositions discussed in Section 1.

1. Cone of influence reduction preserves bisimulation equivalence.
2. Bisimulation equivalence preserves LTL semantics.

Furthermore, we want the generic properties of bisimulation equivalence and LTL semantics that involve reasoning about infinite sequences to be handled by the HOL formalization, and discharge other properties (*e.g.*, properties of cone of influence reduction algorithm, circuits and corresponding Kripke structure models, etc.) by reusing the theorems already proven with ACL2. In this section, we explain how we achieve these objectives.

The formal rendition in HOL of the two parts above is given by the two lemmas shown in Fig. 2. We first discuss how to derive Theorem `correctness-of-reduction` from these two lemmas. We will then discuss the proofs of these lemmas.

By a bit of set-theoretic reasoning, Theorem `correctness-of-reduction` follows from Theorem `correctness-simplified` below.

```
Theorem correctness-simplified

  ∀C Vars.
   (⊨ circuitp C)
   ⇒
   ∀f. (Atoms f) SUBSET (SexpToSet (cone_variables Vars C))
       ⇒
       (SAT (CIRC_TO_MODEL C) f =
        SAT (CIRC_TO_MODEL
              (cone_of_influence_reduction C Vars)) f)
```

```
Theorem cone-bisim

  ∀C Vars.
   (⊨ circuitp C)
   ⇒
   BISIM_EQ (CIRC_TO_MODEL C)
            (CIRC_TO_MODEL (cone_of_influence_reduction C Vars))
            (SexpToSet (cone_variables Vars C))

Theorem bisim-sufficient

  ∀M M' Vars.
   MODEL M ∧ MODEL M' ∧ BISIM_EQ M M' Vars
   ⇒
   ∀f. (Atoms f SUBSET Vars) ⇒ (SAT M f = SAT M' f)
```

Fig. 2 Key lemmas for proving correctness of cone of influence reduction. Theorem `cone-bisim` says that cone of influence reduction produces a model that is bisimulation equivalent to the original, and Theorem `bisim-sufficient` says that bisimulation equivalence suffices for concluding preservation of LTL semantics.

```
  (defthm create-kripke-produces-circuit-model
    (implies (circuitp C)
             (circuit-modelp (create-kripke C))))
  (defthm cone-of-influence-reduction-is-circuit-p
    (implies (circuitp C)
             (circuitp (cone-of-influence-reduction C vars))))
```

Fig. 3 Some theorems about `circuitp`, `create-kripke`, and `cone-of-influence-reduction`. These theorems are proven in ACL2 and imported to HOL4.

How do we prove Theorem `correctness-simplified`? Consider the following instance of Theorem `bisim-sufficient`.

```
M    ↦ CIRC_TO_MODEL C
M'   ↦ CIRC_TO_MODEL (cone_of_influence_reduction C Vars)
Vars ↦ SexpToSet (cone_variables Vars C)
```

Theorem `correctness-simplified` follows from Theorem `cone-bisim` together with the above instance of Theorem `bisim-sufficient`, provided we can discharge the following obligations from the latter's hypothesis:

1. `MODEL (CIRC_TO_MODEL C)`
2. `MODEL (CIRC_TO_MODEL (cone_of_influence_reduction C vars))`

Note that the obligations 1 and 2 are, respectively, properties of the transformation function from circuits to Kripke structures, and the cone of influence reduction algorithm. These properties are already available as the ACL2 theorems shown in Fig. 3, which are imported to HOL4 to complete the proof.

We now turn to the two main lemmas, `cone-bisim` and `bisim-sufficient`. Theorem `bisim-sufficient` is the property that relates bisimulation equivalence with the semantics of LTL and is proven entirely in HOL4. Recall that the analogue of this

property given the nonstandard formalization of LTL semantics was the key source of complexity in the previous ACL2 proof of correctness of cone of influence reduction. On the other hand, the HOL4 proof is a straightforward induction, formalizing the standard textbook proof [1, §13]. The drastic reduction in complexity underlines the importance of employing a reasoning tool with the right expressive power to enable a natural formalization of the concepts involved in a verification project.

It remains to describe the proof of `cone-bisim`. This theorem is an interesting demonstration of the importance of transferring concepts smoothly between different reasoning tools involved in a verification. In particular, the predicate `BISIM_EQ` is a binary relation defined in HOL4, formalizing bisimulation equivalence. On the other hand, in ACL2 we formalize a specific bisimulation relation `c-bisim-equiv`, and show that (1) it is a bisimulation relation (Fig 4), and (2) if C is a circuit and C' is the circuit generated by cone of influence reduction of C, then the Kripke structures generated from C and C' are related by `c-bisim-equiv` (Fig. 5). The proof of Theorem `cone-bisim` involves stitching together these pieces.

To prove `cone-bisim`, assume `(⊨ circuitp C)`, so that by expanding the definition of `CIRC_TO_MODEL`, our goal is to prove the following formula.

```
BISIM_EQ (HOL_MODEL (create_kripke C))
         (HOL_MODEL
           (create_kripke
             (cone_of_influence_reduction C Vars)))
         (SexpToSet (cone_variables Vars C))
```

To discharge this goal, we use Theorem `Hol-bisim-eq` shown in Fig. 6. We will return to the proof of this theorem presently. For now, note that our goal follows from `Hol-bisim-eq` using the following instantiation, provided we can discharge the instantiated hypotheses.

```
m1   ↦ create_kripke C
m2   ↦ create_kripke (cone_of_influence_reduction C Vars)
Vars ↦ cone_variables Vars C
```

To finish the job, we need to discharge the following obligations (corresponding to the instantiated hypotheses of Theorem `Hol-bisim-eq`), from the assumption `(⊨ circuitp C)`.

1. `circuit_modelp (create_kripke C)`
2. `circuit_modelp (create_kripke`
   `(cone_of_influence_reduction C Vars))`
3. `c_bisim_equiv (create_kripke C)`
   `(create_kripke`
   `(cone_of_influence_reduction C Vars))`
   `(cone_variables Vars C)`

These obligations follow from the ACL2 theorems proven already (and imported to HOL): the first two follow from the theorems in Fig. 3, and the third from the key property of cone of influence reduction in Fig. 5.

Finally, Theorem `Hol-bisim-eq` is also an immediate consequence the ACL2 lemmas imported into HOL4! Fig. 7 includes the HOL4 proof; the reader is encouraged to skim this only lightly. Notice that we achieve significant automation through extensive use of the tactic `METIS_TAC`, which employs a resolution prover interface to HOL4 developed by Hurd [8]. The relevant ACL2 lemmas include properties of the bisimulation

```
(defthm c-bisimilar-witness-member-of-states-m->n
  (implies (and (circuit-bisim p m q n vars)
                (next-statep p r m)
                (memberp r (states m)))
           (memberp
             (c-bisimilar-transition-witness-m->n p r m q n vars)
             (states n))))
(defthm c-bisimilar-witness-matches-transition-m->n
  (implies (and (circuit-bisim p m q n vars)
                (next-statep p r m))
           (next-statep
             q
             (c-bisimilar-transition-witness-m->n p r m q n vars)
             n)))
(defthm c-bisimilar-witness-produces-bisimilar-states-m->n
  (implies (and (circuit-bisim p m q n vars)
                (next-statep p r m))
            (circuit-bisim
              r
              m
             (c-bisimilar-transition-witness-m->n p r m q n vars)
              n
              vars)))
(defthm c-bisimilar-equiv-implies-init->init-m->n
  (implies (and (c-bisim-equiv m n vars)
                (memberp s (initial-states m)))
           (memberp
             (c-bisimilar-initial-state-witness-m->n s m n vars)
             (initial-states n))))
(defthm c-bisimilar-equiv-implies-bisimilar-initial-states-m->n
   (implies (and (c-bisim-equiv m n vars)
                 (memberp s (initial-states m)))
            (circuit-bisim
              s
              m
             (c-bisimilar-initial-state-witness-m->n s m n vars)
              n
              vars)))
```

Fig. 4 Key requirements of bisimulation, as formalized in ACL2. It is convenient to think of m and n as Kripke structure. Here `circuit-bisim` relates a state p of m with a state q of n; the conditions stipulate that for each transition in m from state p, there is a matching transition in n from q. The relation `c-bisim-equiv` requires that for each initial state s of m there is some initial state of n that is related to s by `circuit-bisim`. Note that the theorems shown pertain to one direction of bisimulation, relating states of m with those of n; symmetric lemmas in the other direction are omitted here.

relation `c-bisim-equiv` (Fig. 4). The above lemmas were available from the previous ACL2 proof of correctness of cone of influence reduction, unlike the following, which was formulated and proved only after its need became evident during the HOL4 proof effort.

```
(defthm bisim-lemma-1
  (implies (and (memberp a vars)
                (circuit-bisim p m q n vars))
           (equal (memberp a (label-of p m))
```

```
(defthm cone-of-influence-is-c-bisimi-equiv
  (implies
   (circuitp C)
   (c-bisim-equiv (create-kripke C)
                  (create-kripke
                   (cone-of-influence-reduction C vars))
                  (cone-variables vars C))))
```

Fig. 5 ACL2 Theorem formalizing the concept that cone of influence reduction preserves bisimulation equivalence.

```
Theorem Hol-bisim-eq

  ∀m1 m2 Vars.
  (⊨ circuit_modelp m1) ∧
  (⊨ circuit_modelp m2) ∧
  (⊨ (c_bisim_equiv m1 m2 Vars))
  ⇒
  BISIM_EQ (HOL_MODEL m1) (HOL_MODEL m2) (SexpToSet Vars)
```

Fig. 6 Key HOL Lemma for for discharging Theorem `cone-bisim`.

```
 RW_TAC std_ss [BISIM_EQ_def, SPECIFICATION]
  THEN Q.EXISTS_TAC `\(s,s'). (|= circuit_bisim s m1 s' m2 Vars)`
  THEN RW_TAC std_ss []
  THENL
   [RW_TAC (srw_ss()) [HOL_MODEL_def, BISIM_def, SPECIFICATION, SexpToSet_def]
     THENL
      [METIS_TAC [bisim_lemma_1_thm,SPECIFICATION,equal_imp],
       METIS_TAC
        [c_bisimilar_witness_member_of_states_m_greater_n_thm,
         c_bisimilar_witness_matches_transition_m_greater_n_thm,
         c_bisimilar_witness_produces_bisimilar_states_m_greater_n_thm],
       METIS_TAC
        [c_bisimilar_witness_member_of_states_n_greater_m_thm,
         c_bisimilar_witness_matches_transition_n_greater_m_thm,
         c_bisimilar_witness_produces_bisimilar_states_n_greater_m_thm]],
    Q.EXISTS_TAC `c_bisimilar_initial_state_witness_m_greater_n s0 m1 m2 Vars`
     THEN FULL_SIMP_TAC (srw_ss()) [HOL_MODEL_def]
     THEN METIS_TAC
           [c_bisimilar_equiv_implies_init_greater_init_m_greater_n_thm,
    c_bisimilar_equiv_implies_bisimilar_initial_states_m_greater_n_thm],
    Q.EXISTS_TAC `c_bisimilar_initial_state_witness_n_greater_m m1 s0' m2 Vars`
     THEN FULL_SIMP_TAC (srw_ss()) [HOL_MODEL_def]
     THEN METIS_TAC
           [c_bisimilar_equiv_implies_init_greater_init_n_greater_m_thm,
    c_bisimilar_equiv_implies_bisimilar_initial_states_n_greater_m_thm]]
```

Fig. 7 Proof of Theorem `Hol-bisim-eq` in HOL4

```
∀p q. (⊨ equal p q) ⇒ ((⊨ p) = (⊨ q))
∀p q. (⊨ implies p q) = (⊨ p) ==> (⊨ q)
∀a b c. ((⊨ (if a then b else c))
      = (a ∧ (⊨ b)) ∨ (¬a ∧ (⊨ c))
```

Fig. 8 Some generic reusable lemmas in `sexp` theory.

```
                    (memberp a (label-of q n)))))
```

It was convenient to carry out the proof of this lemma using ACL2 rather than HOL4, which was easy given the lemmas already proved in ACL2.

Not surprisingly, a bit of HOL4 proof-hacking was required as well, introducing lemmas some of which are reusable in future proof efforts using the link between HOL4 and ACL2. Some examples are shown in Fig. 8. A dozen or so general lemmas similar to this were added to the infrastructure supporting the HOL theory of `sexp`.

5 Concluding Remarks

The ACL2 and HOL4 systems have nearly disjoint user communities, and are widely acknowledged to have different strengths. Furthermore, the mode of interaction is typically different for the two theorem provers. ACL2 users tend to rely on built-in proof automation (*e.g.*, efficient conditional rewriting, linear arithmetic, metatheoretic reasoning), fast evaluation of ground terms, and structuring mechanisms that support large parallel proof developments. HOL4 users often take advantage of direct programmability of proofs along with the powerful specification capability offered by higher-order logic. A linkage between the two systems, especially one which permits their complementary strengths to be deployed together smoothly and safely, is therefore of interest to both of their user communities. On the other hand, the utility of the combined system crucially depends on the ease with which the two systems can be used together.

This paper has demonstrated a successful application of the linkage between ACL2 and HOL4 in a proof that takes advantage of the strengths of the respective proof assistants. Specifically, the proof takes advantage of the expressive power of HOL and the work already completed in ACL2. Our entire effort took only a matter of days, and we expect that efforts of similar scope might take only a few hours now using our existing files for guidance. It is worth noting that developing a usable robust linkage between two mature formal reasoning tools is a non-trivial exercise involving significant attention to both logical and engineering issues. The experience suggests that the linkage between HOL4 and ACL2 is robust enough for non-trivial proof projects.

A key reason why the the combination of HOL4 and ACL2 is effective in this project is that the proof is logically broken into two relatively independent fragments that each exploits the strength of one prover. Recall that the theorem `bisim-sufficient` (proved with HOL4) is a property relating bisimulation equivalence with LTL semantics, and is not independent of the specific abstraction algorithm (*viz.*, cone of influence reduction) being verified; on the other hand, Theorem `cone-bisim` (exported from pieces developed with ACL2) is a property of cone of influence reduction and does not need reasoning about properties of bisimulation or their connection with LTL. Such a clear logical decomposition is important since it permits the insightful reasoning involved in

a non-trivial proof to be carried out within a single system without requiring the user to jump between different tools while in the middle of a serious reasoning effort; the job of the interface then is confined to assembling the results produced by individual tools to complete the proof. As an aside, an upshot of the decomposition for the current proof is that Theorem `bisim-sufficient` can be reused without modification in future work to prove correctness of other abstraction algorithms (*e.g.*, symmetry) that also preserve bisimulation equivalence.

Our project may be criticized on the ground that the validity of the proof depends on the soundness of several tools and interfaces (*viz.*, ACL2, HOL4, and the interface connecting the two), instead of a single reasoning tool. Since the original complexity in the ACL2 proof arose from the limitations in the expressive power of ACL2, it may be argued that a cleaner approach to the proof would simply be to do the proof from scratch in higher-order logic. However, ACL2 was chosen for the original proof for a reason: support for efficient executability of formal definitions was critical to its goal of reasoning about compositional model checking procedures which can then be used for practical hardware verification. Furthermore, the previous project resulted in successful proofs of several key properties of cone of influence reduction (*e.g.*, theorems in Fig. 4 and 5) which must be re-done in HOL if the correctness proof were to be done from scratch. But proofs of these theorems involve ACL2 scripts encompassing several thousand lines! Furthermore, since these properties did not involve reasoning about the nonstandard semantics of LTL, there is no reason to expect reduction in proof effort through HOL's superior expressive power. Instead, the path we chose exploits the strengths of HOL for parts of the proof that are difficult in ACL2 while reusing other parts from the previous effort.

In future work, we are investigating other applications of the link between ACL2 and HOL4. Some potential applications include properties of programs running on the Java Virtual Machine or the Rockwell Collins AAMP™ microprocessor ISA. Detailed ACL2 models of these machines are available [17,6]; ACL2 also has significant infrastructure developed over the years to reason about these models. On the other hand, properties of certain programs (*e.g.*, cryptographic algorithms) are more naturally stated in higher-order logic. The use of the interface allows reasoning about these higher-order properties while making use of the high-fidelity ACL2 models and the supporting infrastructures.

Acknowledgements We thank the anonymous referees for several insightful comments. This work has been funded in part by the National Science Foundation under Grants IIS-0417413, CNS-0429591 (with DARPA), CCF-0945316, CCF-0916772, and CNS-0910913, and by the Semiconductor Research Corporation under Grant 08-TJ-1849. Kaufmann also thanks the Texas-United Kingdom Collaborative for travel support to Cambridge, England, and the Computer Laboratory at the University of Cambridge for hosting him during the course of this research.

References

1. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model-Checking*. The MIT Press, Cambridge, MA, January 2000.
2. M. J. C. Gordon, W. A. Hunt, Jr., M. Kaufmann, and J. Reynolds. An Embedding of the ACL2 Logic in HOL. In *Proceedings of the 6th International Workshop on the ACL2 Theorem Prover and Its Applications (ACL2 2006)*, pages 40–46. ACM, August 2006.

3. M. J. C. Gordon, W. A. Hunt, Jr., M. Kaufmann, and J. Reynolds. An Integration of HOL and ACL2. In A. Gupta and P. Manolios, editors, *Proceedings on the 6th International Conference on Formal Methods in Computer-Aided Design (FMCAD-2006)*, pages 153–160. IEEE Computer Society Press, 2006.
4. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem-Proving Environment for Higher-Order Logic*. Cambridge University Press, 1993.
5. M. J. C. Gordon and A. M. Pitts. The HOL Logic and System. In J. Bowen, editor, *Towards Verified Systems*, volume 2 of *Real-Time Safety Critical Systems*, chapter 3, pages 49–70. Elsevier, 1994.
6. D. Greve, R. Richards, and M. Wilding. A Summary of Intrinsic Partitioning Verification. In M. Kaufmann and J S. Moore, editors, *5th International Workshop on the ACL2 Theorem Prover and Its Applications (ACL2 2004)*, Austin, TX, November 2004.
7. G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, November 2003.
8. J. Hurd. An LCF-Style Interface Between HOL and First-order Logic. In A. Voronkov, editor, *Proceedings of the 18th International Conference on Automated Deduction (CADE 2002)*, volume 2392 of *LNCS*, pages 134–138. Springer, July 2002.
9. M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2004.
10. M. Kaufmann, P. Manolios, and J S. Moore, editors. *Computer-Aided Reasoning: ACL2 Case Studies*. Kluwer Academic Press, Boston, MA., 2000.
11. M. Kaufmann, P. Manolios, and J S. Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Press, 2000.
12. M. Kaufmann and J S. Moore. A Precise Description of the ACL2 Logic. See URL `http://www.cs.utexas.edu/users/moore/publications/km97.ps.gz`, 1997.
13. M. Kaufmann and J S. Moore. Structured Theory Development for a Mechanized Logic. *Journal of Automated Reasoning*, 26(2):161–203, 2001.
14. M. Kaufmann and J S. Moore. An ACL2 Tutorial. In O. A. Mohamed, C. Muñoz, and S. Tahar, editors, *Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2008)*, volume 5170 of *LNCS*, pages 17–21. Springer, 2008.
15. M. Kaufmann and J S. Moore. The ACL2 Home Page, 2009. `http://www.cs.utexas.edu/users/moore/acl2/`.
16. R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton University Press, 1995.
17. H. Liu and J S. Moore. Executable JVM model for analytical reasoning: A study. *Science of Computer Programming*, 57(3):253–274, 2005.
18. M. Norrish and K. L. Slind. The HOL4 Home Page, 2009. `http://hol.sourceforge.net/`.
19. S. Ray, J. Matthews, and M. Tuttle. Certifying Compositional Model Checking Algorithms in ACL2. In W. A. Hunt, Jr., M. Kaufmann, and J S. Moore, editors, *4th International Workshop on the ACL2 Theorem Prover and Its Applications (ACL2 2003)*, Boulder, CO, July 2003.
20. J. R. Shoenfield. *Mathematical Logic*. Adison-Wesley, Reading, MA, 1967.
21. K. Slind and M. Norrish. A Brief Overview of HOL4. In O. A. Mohamed, C. Muñoz, and S. Tahar, editors, *Proceedings of the 21st International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2008)*, volume 5170 of *LNCS*, pages 28–32. Springer, 2008.