LCF-style for Secure Verification Platform based on Multiway Decision Graphs

Sa'ed Abed[1] and Otmane Ait Mohamed[2]

[1]Department of Computer Engineering,
Hashemite University, Zarqa, Jordan
sabed@hu.edu.jo
[2]Department of Electrical and Computer Engineering,
Concordia University, Montreal, Canada
ait@ece.concordia.ca

Abstract. Formal verification of digital systems is achieved, today, using one of two main approaches: states exploration (mainly model checking and equivalence checking) or deductive reasoning (theorem proving). Indeed, the combination of the two approaches, states exploration and deductive reasoning promises to overcome the limitation and to enhance the capabilities of each. A comparison between both categories is discussed in details. In this paper, we are interested in presenting as an example a platform for Multiway Decision Graphs (MDGs) in LCF-style theorem prover. Based on this platform, many conversions such as the reachability analysis and reduction techniques can be implemented that uses the MDG theory within the HOL theorem prover. The paper also questions the best formalization principle of decision graphs to build such a platform in theorem proving since a set of basic operations are used to efficiently manipulate the decision graphs which constitute the kernel of the model checking algorithms, by describing two alternatives to formalize these decision graphs. Then we contrast between them according to their efficiency, complexity and feasibility. Finally, we hope this paper to serve as an adequate introduction to the concepts involved in formalization and a survey of relevant work.

1 Introduction

With the increasing complexity of the design of digital systems and the size of the circuits in VLSI technology, the role of design verification has gained a lot of importance. Serious design errors and bugs take a lot of time and effort to be detected and corrected especially when they are discovered late in the verification process. This will increase the total cost of the chip. In order to overcome these limitations, *formal verification techniques* arose as a complement to simulation for detecting errors as early as possible, thus ensuring the correctness of the design.

Formal verification of digital systems is achieved, today, using one of two main approaches: states exploration [29] (mainly model checking and equivalence checking) or deductive reasoning (theorem proving). It is accepted that both approaches have complementary strengths and weaknesses.

Model checking algorithms can automatically decide if a temporal property holds for a finite-state system. Furthermore, they can produce a counterexample when the property does not hold, which can be very important for correcting the corresponding error in the implementation under verification or in the specification itself. However, model checking suffers from the state explosion problem: the number of the explored states grows exponentially in the size of the system description.

In deductive reasoning, the correctness of a design is formulated as a theorem in a mathematical logic and the proof is checked using a general-purpose theorem-prover. Based on first-order and high-order logic, these theorem provers are known for their abilities to express relationships over unbounded data structures. Therefore, theorem proving tools are not sensitive to the state explosion problem when used to reason formally about such data and relationships. Unfortunately, if the property fails to hold, deductive methods do not give a counterexample. Furthermore, this frequent situation requires skilled manual guidance for verification and human insight for debugging. Yet theorem provers, today, provide feedback, but only expert user can track the proof trace and determine whether the fault lies within the system, the property being verified, or within the failed proof tactic.

Indeed, the combination of the two approaches, states exploration and deductive reasoning promises to overcome the limitations and to enhance the capabilities of each. This combination can be performed either by adding a layer of deduction theorems and rules on top of the model checking tool (hybrid approach) or by embedding model checking algorithms inside theorem provers (deep embedding approach). Our work is motivated by using the deep embedding approach to blend the best of model checker and theorem prover.

The paper is organized as follows: In Section 2, we briefly introduce the formal verification techniques. The combination of model checkers and theorem provers is described in Section 3. We then survey the literature and present the related work in Section 4. Section 5 overviews the formalization of MDG-HOL platform and the proof of the correctness methodology. Finally, Section 6 concludes the paper and gives some future research directions.

2 Formal Verification Techniques

Formal verification problem consists of mathematically establishing that an implementation behaves according to a given set of requirements or specification. To classify the various approaches, we first look at the three main aspects of the verification process: the system under investigation (implementation), the set of requirements to obey (specification) and the formal verification tool to verify the process (relationship between implementation and specification).

The implementation refers to the description of the design that is to be verified. It can be described at different levels of abstraction which results in different verification methods. Another important issue with the implementation is the class of the system or circuit to be verified, i.e., whether it is combinational/sequential, synchronous/asynchronous, pipelined or parameterized hardware. These variations may require different approaches. The specification refers to the property with respect to which the correctness is to be determined. In practice, one needs to model both the implementation and the specification in the tool, and then uses one of the formal verification algorithms of the tool to check the correctness of the system or in some cases give a trace of error (counter-example).

Formal techniques have long been developed within the computing research community as they provide sound mathematical foundation for the specification, implementation and verification of computer system. Thus, formal verification is proposed as a method to help certify hardware and software, and consequently, to increase confidence in new designs. A correctness proof cannot guarantee that the real device will never malfunction; the design of the device may be proved correct, but the hardware actually built can still behave in a way unintended by the designer. Wrong specification can play a major rule in this, because it has been verified that the system will function as specified, but it has not been verified that it will work correctly. Defects in physical fabrication can cause this problem too. In formal verification a model of the design is verified, not the real physical implementation. Therefore, a fault in the modeling process can give false negatives (errors in the design which do not exist). Although sometimes, the fault covers some real errors.

Formal verification approaches can be generally divided into two main categories: theorem proving methods and state exploration methods such as model checkers as described in the following subsections.

2.1 Theorem Proving

Theorem proving is an approach where the specification and the implementation are usually expressed in first-order or higher-order logic. Their relationship is formed as a theorem to be proved within the logic system. This logic is a set of axioms and a set of inference rules. Steps in the proof appeal to the axioms and rules, and possibly derived definitions and intermediate lemmas. The axioms are usually "elementary" in the sense that they capture the basic properties of the logic's operators [17].

Theorem proving utilizes the proof inference technique. The problem itself is transformed into a sequent, a working representation for the theorem proving problem. Then a sequent holds if the formula f holds in any model:

$$\models f$$

Theorem proving methods have been in use in hardware and software verification for a number of years in various research projects. Some of the well-known theorem provers are HOL (Higher-Order Logic), ISABELLE, PVS (Prototype Verification System), Coq and ACL2 [21, 41, 14, 24, 27]. These systems are distinguished by, among other aspects, the underlying mathematical logic, the way automatic decision procedures are integrated into the system, and the user interface. Even though they are powerful, they require expertise in using a theorem prover. User is expected to know the whole design leading to a white box verification approach. It is not fully automated and requires a large amount of time to verify the system. Another shortcoming is the inability to produce counterexamples in the event of a failed proof, because the user does not know whether the required property is not derivable or whether the person conducting the derivation is not ingenious enough. The advantage of the deductive verification approach is that it can handle very complex systems because the logics of theorem provers are more expressive. In the next subsection, we will overview the HOL theorem proving system, which we intend to use in this work.

The HOL Theorem Prover: The HOL system is an LCF [18] (Logic of Computable Functions) style proof system. Originally intended for hardware verification, HOL uses higher-order logic to model and verify variety of applications in different areas; serving as a general purpose proof system. We cite for example: reasoning about security, verification of fault-tolerant computers, compiler verification, program refinement calculus, software verification, modeling, and automation theory.

HOL provides a wide range of proof commands, rewriting tools and decision procedures. The system is user-programmable which allows proof tools to be developed for specific applications; without compromising reliability [21].

The basic interface to the system is a Standard Meta Language (SML) interpreter. SML is both the implementation language of the system and the Meta Language in which proofs are written. Proofs are input to the system as calls to SML functions. The HOL system supports two main different proof methods: forward and backward proofs in a natural-deduction style calculus.

Theorems in HOL are represented by values of the ML abstract type **thm**. There is no way to construct a theorem except by carrying out a proof based on the primitive inference rules and axioms. HOL has many built-in inference rules and ultimately all theorems are proved in terms of the axioms and basic inferences of the calculus. By applying a set of primitive inference rules, a theorem can be created. Once a theorem is proved, it can be used in further proofs without recomputation of its own proof.

HOL also has a rudimentary library facility which enables theories to be shared. This provides a file structure and documentation format for self contained HOL developments. Many basic reasoners are given as libraries such as **mesonLib**, **bossLib**, and **simpLib**. These libraries integrate rewriting, conversion and decision procedures to free the user from performing low-level proof.

2.2 State Exploration Methods

State exploration methods use states space traversal algorithms on finite-state models to check if the implementation satisfies its specification. They are focused mostly on automatic decision procedures for solving the verification problem.

Model checking is a state exploration based verification technique developed in the 1980s by Clarke and Emerson [12] and independently by Quielle and Sifakis [43]. In model checking, a state of the system under consideration is a snapshot of the system at certain time, given by the set of the variables values of that system at that time. The system is then modeled as a set of states together with a set of transitions between states that describe how the system moves from one state to another in response to internal or external stimulus. Model checking tools are then used to verify that desired properties (expressed in some temporal logic) hold in the system.

State exploration approach has two important advantages. First, once the correct design of the system and the required properties has been fed in, the verification process is fully automatic. Second, in the event of a property not holding, the verification process is able to produce a counter-example (i.e. an instance of the behavior of the system that violates the property) which is extremely useful in helping the human designers pinpoint and fix the flaw. On the other hand, model checkers are unable to handle very large designs due to the state space explosion problem [12]. Another drawback is the problematic description of specifications as properties, this description sometimes may not give full system coverage.

Model checkers such as SPIN [23], COSPAN [31], SMV [33], and MDG [53] take as input, essentially, a finite-state system and temporal property in some variety or subset of CTL*, and automatically check that the system satisfies the property. Moreover, the model is often restricted to a finite-state transition system, for which finite-state model checking is known to be decidable. The design or model M is formalized in terms of a state machine (*Transition System*), or a Kripke structure and the property ϕ is formalized as a logical formula that the machine should satisfy. The verification problem is stated as checking the formula ϕ in the model M:

$$M \vDash \phi$$

If the model M is represented explicitly as a transition relation, then the size of the model is limited to the number of states that can be stored in the computer memory, which are a few million states with the current technology. To increase the size of the model, more efficient state representations can be used to manipulate these formulae using BDDs or SAT solving techniques.

Binary Decision Diagrams (BDDs) [10] are data structures used as a compact representation for the Boolean function which improves the capacity of the model checker. Different representations of ROBDDs (Reduced Order Binary Decision Diagrams) [11] are used to manipulate the state transition relations as diagrams and this allows model checkers to verify larger systems. Still, most model checkers face the state space explosion problems [12] even using Symbolic Model Checking. To be able to apply model checking to larger designs, state reduction techniques are used that exploit some features of the model, the properties, or the problem domain to reduce the state space to a tractable size. Examples include partitioned transition relation, dynamic variable reordering, cone of influence reduction, abstraction, problem-specific techniques, e.g. when the original design is rewritten in a simpler way, omitting the irrelevant details, but preserving the important behavior for the property being verified.

An alternative for decision graphs is to represent the transition relation in CNF and use Satisfiability Checking (SAT) [15, 49] with several properties that make them attractive compared to BDDs. SAT solvers can decide satisfiability of very large Boolean formulae in reasonable time, but they are not canonical and require additional efforts to check for equivalence of formulas. As a result, various researchers have developed routines for performing Bounded Model Checking (BMC) [9, 16, 3] using SAT. The common theme is to convert the problem of interest into a SAT problem, by devising the appropriate propositional Boolean formula, and to utilize other non-canonical representations of state sets. However, they all exploit the known ability of SAT solvers to find a single satisfying solution when it exists. Moreover, SAT solver technology has improved significantly in recent years with a number of sophisticated packages now available. Well known state-of-the-art SAT solvers include CHAFF [38], GRASP [32] and SATO [54]. Since state sets can be represented as Boolean formulae, and since most model checking techniques manipulate state sets, SAT solvers have enormously boosted their speed and applicability.

3 Combining Model Checking based Decision Diagrams and Theorem Proving

Model checking is automatic while theorem proving is not. On the other hand, theorem proving can handle complex systems while model checking can not. Today, there exist a number of integration tools of theorem proving and model checking. The motivation is to achieve the benefits of both tools and to make the verification simpler and more effective. In this section, we explore two approaches of linking proof systems to external automated verification tools. The approaches can be divided in two kinds:

1. Hybrid approach: adding a layer of deduction theorems and rules on top of Decision Diagrams tool, i.e. combining theorem provers with other powerful model checking tool.
2. Deep embedding approach: adding Decision Diagrams algorithms to theorem provers.

We first review the most related work to every approach and then, we contrast between them according to their efficiency, complexity and feasibility.

3.1 Hybrid Approach

The hybrid approach implements a tool linking model checking and theorem proving. During the verification procedure, the user deals mainly with the theorem proving tool. Verification using hybrid approach proceeds as shown in Figure 1. The user starts by providing the theorem proving with the design (specification or implementation), the property and the goal to be proven. If the goal fits the required pattern, the theorem proving tool generates the required model checking files (sub-goals). The latter are sent to the model checking tool for verification. If the property holds, a theorem is created (Make-Theorem). Otherwise, the proof is performed interactively.

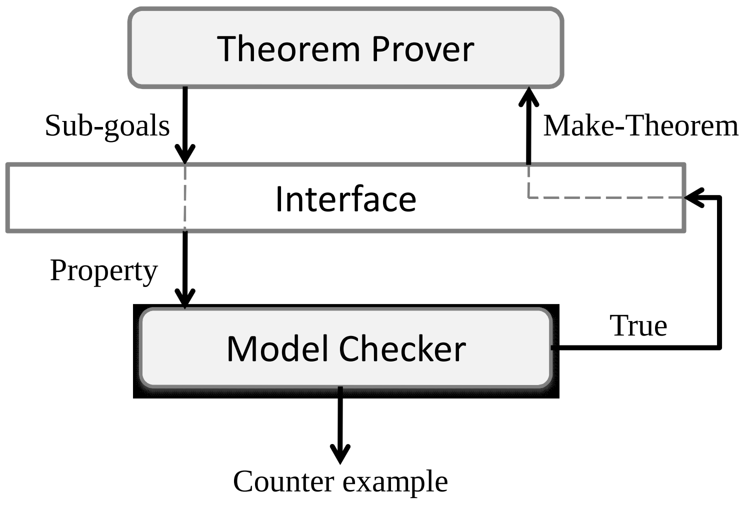

Fig. 1. Theorem Proving and Model Checking Interface

The linkage between both tools is carried out using scripting languages (translators) to be able to automatically verify small subgoals generated by the theorem prover from a large system. The disadvantage of this approach lies in achieving an efficient and correct translation from theorem prover logic to a model checker and from model checker to theorem prover (import the result or give a counter-example). Successful combinations of this kind have been achieved in [25, 2, 46, 26, 47, 28, 37].

3.2 Deep Embedding Approach

In this approach, the emphasis is to establish a secure platform for new verification algorithms. The performance penalty will be compensated by the secure infrastructure. The approach implements a model checking inside a theorem proving tool. As shown in Figure 2, the design and the property are fed to the model checking to check if the property holds and create a theorem. Otherwise, the proof cannot be performed.

The result of the model checker is correct by construction, since both of the theory and the implementation are proved correct in the theorem prover. Thus a high assurance of soundness is guaranteed because more work is backed up by

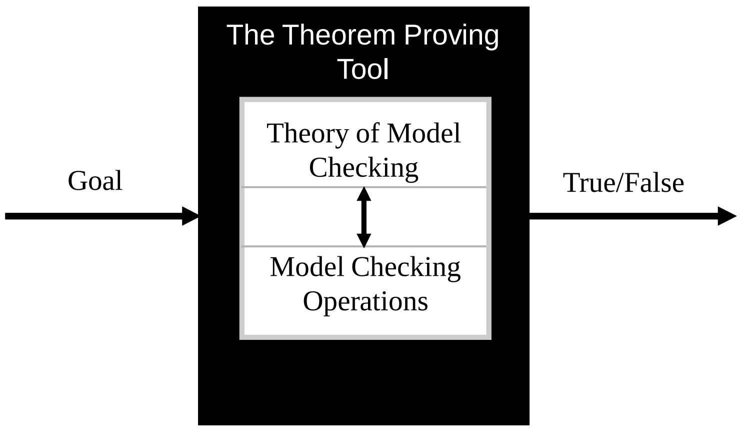

Fig. 2. Embedding Model Checking inside Theorem Proving Tool

mechanized fully-expansive proof. The price for the extra proof and flexibility is in increased development effort. This approach differs from the hybrid approach in the way the verification is performed. In fact, we do not use an external checking tool, instead we deeply embed the model checker algorithms inside the theorem prover. Thus the criteria of correctness by construction, efficiency, flexibility and expressiveness can be met. Successful works have been achieved in [19, 20, 22, 8, 36].

The "deep embedding" approach [44] introduce the model checker syntax as a new higher order logic type and then define the operations and algorithms based on this syntax within the theorem prover. This contrasts within a "shallow embedding" where the syntax is not formally represented in the logic, only in the meta-language. In general, a deep embedding allows one to reason about the language itself rather than just the semantics of programs in the language.

4 Related Work

We consider two categories of related work: embedding of model checking algorithms in theorem provers and correctness proof of these algorithms.

4.1 Embedding of Model Checking Algorithms in Theorem Provers

Model checkers [34] are usually built on top of BDDs [10], or some other set of efficiently implemented algorithms for representing and manipulating Boolean formulae. The closest work, in approach to our own is that of Joyce and Seger [48], Gordon [20, 19] and later Amjad [8].

The Voss system [48], an implementation of Symbolic Trajectory Evaluation (STE), was implemented in a lazy Functional Language (FL). In [25] Voss was interfaced to HOL and the verification using a combination of deduction and

STE was demonstrated. The HOL-Voss system integrates HOL88 deduction with BDD computations. A system based on this idea, called Voss-ThmTac, was later developed by Aagaard: combination of the ThmTac theorem prover with the Voss system. Then the development of HOL-Voss evolved into a new system called Forte [1]. Recently, Forte [35] is mostly used in Intel as an industrial integrated formal verification tool.

Gordon [20] integrated the BDD based verification system BuDDy (BDD package implemented in C) into HOL. The aim of using BuDDy is to get near the performance of C-based model checker, whilst remaining fully expansive, though with a radically extended set of inference rules.

In [22], Harrison implemented BDDs inside the HOL system without making use of external oracle. The BDD algorithms were used by a tautology-checker. However, the performance was about thousand times slower than with a BDD engine implemented in C. Harrison argued that by re-implementing some of HOL's primitive rules, the performance could be improved by around ten times.

Amjad [8] demonstrated how BDD based symbolic model checking algorithms for the propositional *μ-calculus* (L_μ) can be embedded in HOL theorem prover. This approach allows results returned from the model checker to be treated as theorems in HOL. By representing primitive BDD operations as inference rules added to the core of the theorem prover, the execution of a model checker for a given property is modeled as a formal derivation tree rooted at the required property. These inference rules are hooked to a high performance BDD engine [20] which is external to the theorem prover. Thus, the HOL logic is extended with these extra primitives. Empirical evidence suggests that the efficiency loss in this approach is within reasonable bounds. The approach still leaves results reliant on the soundness of the underlying BDD tools. A high assurance of soundness is obtained at the expenses of some efficiency.

Our work, deals with embedding MDGs rather than BDDs. In fact, while BDDs are widely used in state-exploration methods, they can only represent Boolean formulae. By contrast, MDGs represent a subset of first-order terms allowing the abstract representation of data and hence raising the level of abstraction. Another major difference is that it implements the related inference rules for BDD operators in the core of HOL as untrusted code, whereas we implement the MDG operations as a trusted code inside HOL itself.

Several works in [28], [42] and [37] provide strategies to link the HOL theorem prover and the MDG tool. The verification is performed within HOL and MDG separately, where the MDG tool is considered as an oracle. That makes the global proof informal, and this method does not extend readily to the fully expansive approach (theorem prover). Thus, it achieves the integration goal at the expense of higher assurance of correctness.

Mhamdi and Tahar [36] follow a similar approach to the BuDDy work [20]. The work builds on the MDG-HOL [28] project, but uses a tightly integrated system with the MDG primitives written in ML rather than two tools communicating as in MDG-HOL system.

Haiyan et al. [52] verified formally the linkage between a simplified version of MDG tool and HOL theorem prover. The verification is based on importing MDG results to HOL theorems. Then, they combine translator correctness theorems with the linkage theorems in order to allow low level MDG verification results to be imported into HOL in terms of the semantics of MDG-HDL. The work didn't give a formal proof of the soundness of the MDG structure and operators.

4.2 Correctness Proof of Model Checking Algorithms

Verification of BDD algorithms has been a subject of active research using different proof assistants such that HOL, PVS, Coq, and ACL2 [21, 14, 24, 27]. A common goal of these papers is to extend the prover with a certified BDD package to enhance the BDD performance, while still inside a formal proof system. Moreover, there is a general consensus in the formal verification community that correctness proofs should be checked, partly or wholly, by computers. Some efforts have been made to verify model checkers and theorem provers.

In [45], the authors successfully carried out the verification task of the RAVEN model checker. RAVEN is a real-time model checker which uses time-extended finite state machines (interval structure) to represent the system and a timed version of CTL (CCTL) to describe its properties. The specification and the correctness proof were carried out using an interactive specification and verification system KIV.

In [39], the author showed a mechanism of how certifying model checker can be constructed. The idea is that, a model checker can produce a deductive proof on either success or failure. The proof acts as a certificate of the result, since it can be checked independently. A certifying model checker thus provides a bridge from the model-theoretic to the proof-theoretic approach to verification. The author developed a deductive proof system for verifying branching time properties expressed in the μ-calculus, and showed it to be sound and relatively complete. Then, a proof generation in this system from a model checking run is presented. This is done by storing and analyzing sets of states that are generated by the fixpoint computations performed during model checking.

Krstic and Matthews [30] provided a technique for proving correctness of high performance BDD packages. In their work, they adopted an abstraction method called monadic interpretation for verifying an abstraction of the BDD programs with the primitives specified axiomatically. The method is suitable for higher order logic theorem provers such as Isabelle/HOL. The monadic interpreter translates source programs of input type A and output type B into function of type $A \Rightarrow MB$ in the target functional language, where the type constructors M is a suitable monad that encapsulate the notion of computation used by the source language to describe BDD programs. At this level, they modeled the BDD programs as a function in higher order logic in the style of monadic interpreters. Then the correctness proof was carried out on the BDD abstract model.

Wright [51] described an embedding of higher order proof theory within the logic of the HOL theorem proving system. Types, terms and inferences were represented as new types in the logic of the HOL system, and notions of proof and provability were defined. Using this formalization, it was possible to reason about the correctness of derived inference rules and about the relations between different notions of proofs: a Boolean term is provable if and only if there exists a proof for it. The formalization is also intended him to make it possible to reason about programs that handle proofs as their data (e.g., proof checker).

The authors in [50] implemented and proved the correctness of BDD algorithms using Coq. One of their goals was to extract a certified algorithm manipulating BDDs in Caml (the implementation language of Coq). BDDs were represented as DAGs and maps were used to model a state of the memory in which all the BDDs are stored. The authors used reflection to prove a given property P applied to some term t where the program is described and proved in Coq. This means that writing a program π that takes t as an input and returns true exactly when $P(t)$ holds. Then, to show π is correct with respect to P they needed to be sure that whenever $\pi(t)$ returns true $P(t)$ holds and this is done inside the Coq proof assistant itself (i.e. the proof of P has been replaced by the computation of π and reflect this by allowing the system to accept meta-level computation as actual proof).

Another concept to prove the program correctness using Hoare logic as described by Ortner and Schirmer [40]. The principle of this logic is to annotate the program with pre- and post-conditions and to to observe the changes made by each statement of the program. Ortner and Schirmer modeled the graph structure of the BDD as a kind of heap and presented the verification of BDD normalization. They follow the original algorithm presented by Bryant in [10]: transforming an ordered BDD into a reduced, ordered and shared BDD. The work is based on Schirmer's research on the Verification Condition Generator (VCG) to generate the proof obligations for Hoare Logic. The proofs are carried out in the theorem prover Isabelle/HOL.

Our work follows the verification of the Boolean manipulating package, but using MDG instead. We provided a complete formalization of the MDG logic and its well-formedness conditions as DFs in HOL mechanically. Based on this infrastructure we formalized the basic MDG operations in HOL following a deep embedding approach and proved their correctness. Our work focuses more on how one can raise the level of assurance by embedding and proving formally the correctness of those operators in HOL to use them as an infrastructure for MDG model checker.

5 MDG-HOL Platform Methodology

The intention of our work is to provide a secure platform that combines an automatic high level MDGs model checking tool within the HOL theorem prover. While related work has tackled the same problem by representing primitive Binary Decision Diagrams (BDD) operations [10] as inference rules added to the core of the theorem prover [20], we have based our approach on the Multiway Decision Graphs (MDGs) [13]. MDG generalizes ROBDD to represent and manipulate a subset of first-order logic formulae which is more suitable for defining model checking inside a theorem prover. With MDGs, a data value is represented by a single variable of an abstract type and operations on data are represented in terms of an uninterpreted functions. Considering MDG instead of BDD will rise the abstraction level of what can be verified using a state exploration within a theorem prover. Furthermore, an MDG structure in HOL allows better proof automation for larger datapaths systems. The work consists of two main phases:

1. provide all the necessary infrastructure (data structure + algorithms) to define a high level state exploration in the HOL theorem prover named as MDG-HOL platform.

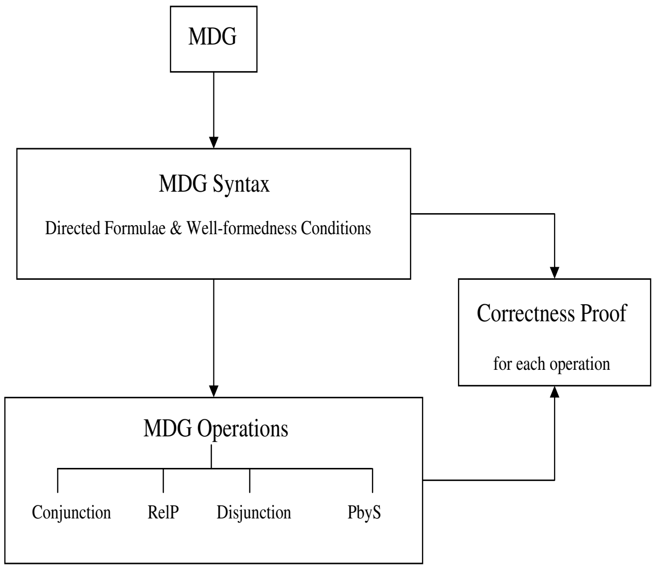

Fig. 3. Overview of the Embedding Methodology in HOL

As shown in Figure 3, we firstly define the MDG structure inside the HOL theorem prover to be able to construct and manipulate MDGs as formulae in HOL. This step implies a formal logic representation for the *MDG Syntax.* It is based on the Directed Formulae DF: an alternative vision for MDG in terms of logic and set theory [7]. Secondly, HOL tactic is defined to check the satisfaction of the well-formedness conditions of any directed formula [4]. This step is important to guarantee the canonical representation of the MDG in terms of DF. Then, the definition of the MDG operations, is associated naturally with a proof of their correctness [5, 6]. Finally, the MDG based reachability analysis is defined in HOL as a conversion that uses the MDG theory (syntax and operations).

2. Based on this MDG-HOL platform, we propose an automatic methodology to verify the soundness of model checking reduction techniques. The idea is to use the consistency of the specifications to verify if the reduced model is faithful to the original one. The user provides the reduction technique, the specifications and the system under verification. Then, we verify automatically using High Order Logic if the reduction technique is soundly applied. The methodology verifies the soundness of the verification output and not the reduction algorithm itself (non-decidable problem).

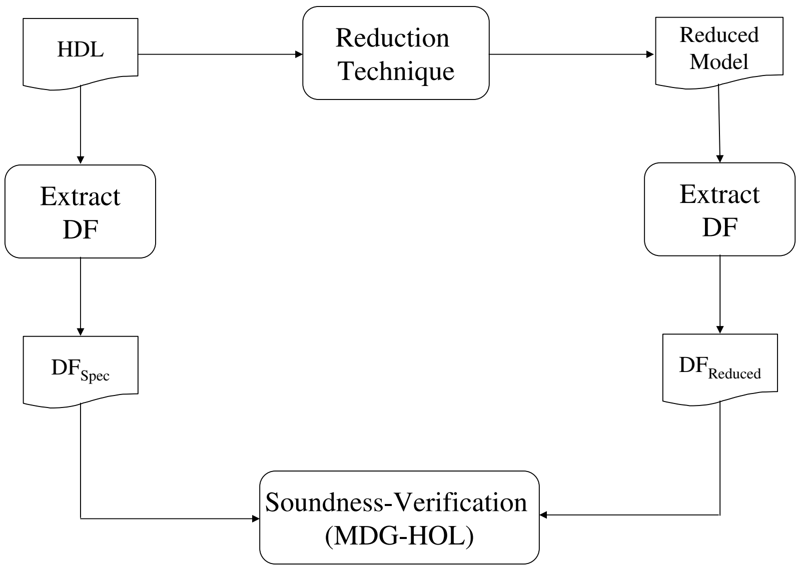

Fig. 4. Overview of the Soundness-Verification Methodology

As shown in Figure 4, we start with a specification of a circuit design written in Hardware Description Language (HDL) and extract a mathematical model in terms of Directed Formulae (DF_{Spec}). The reduction technique itself could be applied on the HDL description either using HOL or an external tool. Similarly, the reduced model is expressed in terms of Directed Formulae ($DF_{Reduced}$).

Then, both DFs should be fed to the MDG-HOL platform where the soundness verification is checked. If the the reduction is proved sound then the formal verification can be performed on the obtained reduced model.

The powerful of our methodology is that it can be used with any verification tool. All what we need is to translate in a sound manner both the model and its reduction in order to embed them thereafter as DFs in HOL and then prove that the reduced model is derived correctly using high order logic.

6 Conclusion and Future Work

BDD based symbolic model checking has proved to be a successful automatic verification technique that can be applied to real designs. However, the state space explosion problem caused by large datapaths is often the bottleneck in applying the symbolic model checking technique. Theorem provers are based on expressive formalisms that are capable of modeling complex systems but requires expertise to verify most properties of practical interest. It has been shown through several research papers that model checking can be efficiently combined with theorem proving in a way that sacrifices neither efficiency of the former, nor the expressiveness of the latter.

In this paper we discussed two alternatives used for the formalization in the theorem proving of decision diagrams, namely BBDs. We compare both of them according to their efficiency, complexity and feasibility.

Moreover, we distinguished two approaches to extend theorem provers. The first approach consists of adding a layer of deduction theorems and rules on top of model checking tool leading to a combination of theorem provers with powerful model checking tool. This approach suffers from two drawbacks: first, very often model checking is used to prove the current sub-goal, and many reduction techniques developed in model checking are simply omitted in the hope that can be done with the existing theorem proving techniques. Second, the translator and the model checking algorithms must be verified or trusted. The second approach consists of deeply embedding the model checker algorithms inside theorem prover. In this approach, the correctness of the result is correct by construction. The drawback is the performance penalty due to the theorem proving itself. Much work is promising in this area to enhance the performance by developing techniques such as reduction and abstraction.

Also, the paper served as an introduction to formalize model checking in high order theorem provers and an extended survey of relevant work. Finally, we have created a new formal theory for MDGs (data structure + operations) inside the HOL theorem prover which provides us with several theoretical advantages without too high performance penalty. We used this secure theory or platform to verify the soundness of model checking reduction techniques. We thus hope that this work will be of interest to the research community and also be of use to industrial practitioners.

References

1. M. Aagaard, R. B. Jones, R. Kaivola, K. R. Kohatsu, and C. H. Seger. Formal Verification of Iterative Algorithms in Microprocessors. In *DAC '00: Proceedings*

of the 37th conference on Design automation, pages 201–206, New York, NY, USA, 2000. ACM.

2. M. D. Aagaard, R. B. Jones, and C. H. Seger. Combining Theorem Proving and Trajectory Evaluation in an Industrial Environment. In *Proceedings of the 1998 Conference on Design Automation (DAC-98)*, pages 538–541, Los Alamitos, CA, June 1998. ACM/IEEE.
3. P. Aziz Abdulla, P. Bjesse, and N. Eén. Symbolic Reachability Analysis based on SAT-Solvers. In *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2000)*, 2000.
4. S. Abed and O. Ait Mohamed. Embedding of MDG Directed Formulae in HOL Theorem Prover. In *Proc. 9th Maghrebian Conference on Software Engineering and Artificial Intelligence MCSEAI'06*, pages 659–664, Agadir, Morocco, December 2006.
5. S. Abed, O. Ait Mohamed, and G. Al Sammane. A High Level Reachability Analysis using Multiway Decision Graph in the HOL Theorem Prover. In *International Conference on Theorem Proving in Higher Order Logics (TPHOLs'07): B-Track Proceedings*, pages 1–17, Kaiserslautern, Germany, September 2007.
6. S. Abed, O. Ait Mohamed, and G. Al Sammane. Reachability Analysis using Multiway Decision Graphs in the HOL Theorem Prover. In *Proc. of ACM SAC '08*, pages 333–338, Brazil, 2008. ACM Press.
7. O. Ait-Mohamed, X. Song, and E. Cerny. On the Non-termination of MDG-based Abstract State Enumeration. *Theoretical Computer Science*, 300:161–179, 2003.
8. H. Amjad. Programming a Symbolic Model Checker in a Fully Expansive Theorem Prover. In *Proceedings of the 16th International Conference on Theorem Proving in Higher Order Logics*, volume 2758 of *Lecture Notes in Computer Science*, pages 171–187. Springer-Verlag, 2003.
9. P. Bjesse and K. Claessen. SAT-based Verification without State Space Traversal. In *Formal Methods in Computer-Aided Design*, pages 372–389, 2000.
10. R. E. Bryant. Graph-based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986.
11. R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.
12. E. M. Clarke, O. Grumberg, and D. E. Long. Model Checking. In *Nato ASI*, volume 152 of F. Springer-Verlag, 1996.
13. F. Corella, Z. Zhou, X. Song, M. Langevin, and E. Cerny. Multiway Decision Graphs for Automated Hardware Verification. In *Formal Methods in System Design*, volume 10, pages 7–46, February 1997.
14. J. Crow, S. Owre, J. Rushby, N. Shankar, and M. Srivas. *A Tutorial Introduction to PVS*. http://www.dcs.gla.ac.uk/proper/papers.html.
15. M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem-Proving. *Commun. ACM*, 5(7):394–397, 1962.
16. M. K. Ganai and A. Aziz. Improved SAT-based Bounded Reachability Analysis. In *ASP-DAC '02: Proceedings of the 2002 conference on Asia South Pacific design automation/VLSI Design*, page 729, Washington, DC, USA, 2002. IEEE Computer Society.
17. M. Gordon. From LCF to HOL: A Short History. pages 169–185, 2000.
18. M. Gordon, R. Milner, and C. Wadsworth. *Edinburgh LCF*, volume 78 of *Lecture Notes in Computer Science*. Springer, 1979.
19. M. J. C. Gordon. Reachability Programming in HOL98 using BDDs. In *International Conference on Theorem Proving in Higher Order Logics TPHOLs*, Lecture Notes in Computer Science, pages 179–196, 2000.

20. M. J. C. Gordon. Programming Combinations of Deduction and BDD-based Symbolic Calculation. *LMS Journal of Computation and Mathematics*, 5:56–76, August 2002.
21. M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic.* Cambridge University Press, New York, NY, USA, 1993.
22. J. Harrison. Binary Decision Diagrams as a HOL Derived Rule. *The Computer Journal*, 38:162–170, 1995.
23. Gerard J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, 1990.
24. G. Huet, G. Kahn, and C. Paulin-Mohring. *The Coq Proof Assistant : A Tutorial.* http://coq.inria.fr/doc/tutorial.html.
25. J. J. Joyce and C. H. Seger, editors. *The HOL-Voss System: Model Checking inside a General-Purpose Theorem Prover*, volume 780 of *Lecture Notes in Computer Science.* Springer, 1994.
26. K. Schneider and T. Kropf. Verifying Hardware Correctness by Combining Theorem Proving and Model Checking. Technical Report SFB358-C2-5/95, Institut für Rechnerentwurf und Fehlertoleranz, 1995.
27. M. Kaufmann, P. Manolios, and J. S. Moore. *Computer-Aided Reasoning: An Approach.* Kluwer Academic Publishers, June 2000.
28. S. Kort, S. Tahar, and P. Curzon. Hierarchal Verification using an MDG-HOL Hybrid Tool. *International Journal on Software Tools for Technology Transfer*, 4(3):313–322, May 2003.
29. T. Kropf. *Introduction to Formal Hardware Verification.* Springer Verlag, 1999.
30. S. Krstic and J. Matthews. Verifying BDD Algorithms through Monadic Interpretation. In *VMCAI '02: Revised Papers from the Third International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 182–195, London, UK, 2002. Springer-Verlag.
31. R. P. Kurshan and L. Lamport. Verification of a Multiplier: 64 Bits and Beyond. In C. Courcoubetis, editor, *Proceedings of the 5th International Conference on Computer Aided Verification*, volume 697, pages 166–179, Elounda, Greece, 1993. Springer-Verlag.
32. J. P. Marques-Silva and K. A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, 1996.
33. K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, Boston, Massachusetts, 1993.
34. K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, Boston, Massachusetts, 1993.
35. T. Melham. Integrating Model Checking and Theorem Proving in a Reflective Functional Language. In Eerke A. Boiten, John Derrick, and Graeme Smith, editors, *Integrated Formal Methods: 4th International Conference, IFM 2004: Canterbury, UK, April 4–7, 2004: Proceedings*, volume 2999 of *Lecture Notes in Computer Science*, pages 36–39. Springer-Verlag, 2004.
36. T. Mhamdi and S. Tahar. Providing Automated Verification in HOL using MDGs. In *Automated Technology for Verification and Analysis*, pages 278–293, 2004.
37. R. Mizouni, S. Tahar, and P. Curzon. Hybrid Verification Incorporating HOL Theorem Proving and MDG Model Checking. *Microelectronics Journal*, 2006.
38. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

39. K. S. Namjoshi. Certifying Model Checkers. In *CAV '01: Proceedings of the 13th International Conference on Computer Aided Verification*, pages 2–13, London, UK, 2001. Springer-Verlag.
40. V. Ortner and N. Schirmer. Verification of BDD Normalization. In *TPHOLs*, pages 261–277, 2005.
41. L. C. Paulson. *Isabelle: A Generic Theorem Prover.* Springer Verlag, 1994.
42. V. Pisini, S. Tahar, P. Curzon, O. Ait-Mohamed, and X. Song. Formal Hardware Verification by Integrating HOL and MDG. In *Proc. of IEEE GLS-VLSI'00, Chicago, USA*, Chicago, Illinois, USA, 2000.
43. J. Quille and J. Sifakis. Specification and Verification of Concurrent Systems in CESAR. In M. Dezani-Ciancaglini and Ogo Montanari, editors, *Proceedings of the 5th International Symposium on Programming*, volume 137, pages 337–351. Springer-Verlag, 1982.
44. R. Boulton, A. Gordon, M.J.C. Gordon, J. Herbert, and J. van Tassel. Experience with Embedding Hardware Description Languages in HOL. In *Proc. of the International Conference on Theorem Provers in Circuit Design: Theory, Practice and Experience*, pages 129–156, Nijmegen, 1992. North-Holland.
45. W. Reif, J. Ruf, G. Schellhorn, and T. Vollmer. Do you Trust your Model Checker? In Warren A. Hunt Jr. and Steven D. Johnson, editors, *Formal Methods in Computer Aided Design (FMCAD).* Springer LNCS 1954, November 2000.
46. S. Rajan, N. Shankar, and M. K. Srivas. An Integration of Model Checking with Automated Proof Checking. In P. Wolper, editor, *Proceedings of the 7th International Conference On Computer Aided Verification*, volume 939, pages 84–97, Liege, Belgium, 1995. Springer Verlag.
47. K. Schneider and D. Hoffmann. A HOL Conversion for Translating Linear Time Temporal Logic to ω-automata. In *TPHOLs*, volume 1690, pages 255–272, Nice, France, 1999. Springer-Verlag.
48. C. H. Seger. Voss – A Formal Hardware Verification System, User's Guide. Technical report, Nortel Networks, Ottawa, Canada, The University of British Columbia, December 1993. ftp ftp.cs.ubc.ca:ftplocaltechreports1993TR-93-45.ps.
49. M. Sheeran, S. Singh, and G. Staalmarck. Checking Safety Properties using Induction and a SAT-Solver. In *FMCAD '00: Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*, pages 108–125, London, UK, 2000. Springer-Verlag.
50. K. N. Verma, J. Goubault-Larrecq, S. Prasad, and S. Arun-Kumar. Reflecting BDDs in Coq. In *Proc. 6th Asian Computing Science Conference (ASIAN'2000), Penang, Malaysia, Nov. 2000*, volume 1961, pages 162–181. Springer, 2000.
51. J. Wright. Representing Higher-Order Logic Proofs in HOL. In *Proceedings of the 7th International Workshop on Higher Order Logic Theorem Proving and Its Applications*, pages 456–470, London, UK, 1994. Springer-Verlag.
52. H. Xiong, P. Curzon, S. Tahar, and A. Blandford. Providing a Formal Linkage between MDG and HOL. *Formal Methods in Systems Design*, 29(3):1–36, 2006.
53. Y. Xu, X. Song, E. Cerny, and O. Ait Mohamed. Model Checking for A First-Order Temporal Logic using Multiway Decision Graphs (MDGs). *The Computer Journal*, 47(1):71–84, 2004.
54. H. Zhang. SATO: An Efficient Propositional Prover. In *Proceedings of the International Conference on Automated Deduction (CADE'97), volume 1249 of LNAI*, pages 272–275, 1997.