Formally Linking MDG and HOL Based on a Verified MDG System

Haiyan Xiong[1], Paul Curzon[1], Sofiène Tahar[2], and Ann Blandford[3]

[1] School of Computing Science, Middlesex University, London, UK.
{h.xiong, p.curzon}@mdx.ac.uk
[2] ECE Department, Concordia University, Montreal, Canada.
tahar@ece.concordia.ca
[3] UCL Interaction Centre, University College of London, London, UK.
a.blandford@ucl.ac.uk

Abstract. We describe an approach for formally linking a symbolic state enumeration system and a theorem proving system based on a verified version of the former. It has been realized using a simplified version of the MDG system and the HOL system. Firstly, we have verified aspects of correctness of a simplified version of the MDG system. We have made certain that the semantics of a program is preserved in those of its translated form. Secondly, we have provided a formal linkage between the MDG system and the HOL system based on importing theorems. The MDG verification results can be formally imported into HOL to form a HOL theorem. Thirdly, we have combined the translator correctness theorems and importing theorems. This allows the MDG verification results to be imported in terms of a high level language (MDG-HDL) rather than a low level language. We also summarize a general method to prove `existential theorems` for the design. The feasibility of this approach is demonstrated in a case study that integrates two applications: hardware verification (in MDG) and usability verification (in HOL). A single HOL theorem is proved that integrates the two results.

Keywords: hardware verification, hybrid verification systems, deductive theorem proving, symbolic state enumeration, usability verification.

1 Introduction

Deductive theorem proving and symbolic state enumeration are complementary approaches to formal verification. In the former, the correctness condition for a design is represented as a theorem in a mathematical logic, and a mechanically checked proof of this theorem is generated using a general-purpose theorem prover. In symbolic state enumeration systems, the design being verified is represented as a decision diagram. Techniques such as reachability analysis are used to automatically verify given properties of the design or machine equivalence. Much of this work is based on Binary Decision Diagrams (BDD) [2].

Deductive theorem proving systems often use interactive proof methods. The user interactively constructs a formal proof which proves a theorem stating the correctness of an implementation. Theorem proving systems allow a hierarchical verification method to be used to model the overall functionality of designs with complex datapaths. They are very general in their application. Theorems cannot only be used to formalize a specific design but also can be abstracted as a general situation of this class of design. Theorem proving systems are semi-automated. To complete a verification, experts with good knowledge of the internal structure of the design are required to guide the proof searching process. This enables the designer to gain greater insight into the system and thus achieve better designs. However, the learning curve is very steep and modeling and verifying a system is very time-consuming. This is a major problem for applying theorem proving systems in industry.

In contrast, symbolic state enumeration systems are automated decision diagram approaches. In this kind of approach, an implementation and its behavioral specification are represented as decision diagrams. A set of algorithms is used to efficiently manipulate the decision diagrams so as to get the correctness results. The symbolic state enumeration verification approach can be viewed as a black-box approach. During the verification, the user does not need to understand the internal structure of the design. The strength of this approach is its speed and ease of use. However, it does not scale well to complex designs since it uses non-hierarchy state-based descriptions of the design. An increase in the number of design components can result in the state space growing exponentially.

Recently, there has been a great deal of work concerned with combining theorem proving and symbolic state enumeration systems to gain the advantages of both. A common approach to combining proof tools is to use a symbolic state enumeration system as an oracle to provide results to the theorem proving system. The issue in such work is to guarantee that the results provided by external tools are theorems within the theory of the proof system. In other words, an oracle is used to receive problems and return answers. For example, the HOL system provides approaches for tagging theorems that are dependent on the correctness of external verification tools. An oracle can be built in the HOL system and viewed as a plug-in. This process brings about two questions.

1. Can we ensure the automated verification system produces correct results?
2. Have the verification results from an automated verification system been correctly converted into a valid theorem in the theorem proving system?

We investigate the answers to the above two questions. Some symbolic state enumeration based systems such as MDG [6] consist of a series of translators and a set of algorithms. Higher level languages such as hardware description languages are used to describe the specification and implementation of the design. The specification and implementation are then translated into the decision diagrams via intermediate languages. The algorithms in the system are used to efficiently and automatically deal with the decision diagrams so as to obtain the correctness results. We need to verify the translators and algorithms in order to get the answer to the first question. To solve the second question, we can formally justify the correctness results obtained from the symbolic state enumeration system in a theorem prover.

The main contribution of this paper is that we describe an approach that provides a formal linkage between a theorem proving system and a symbolic state enumeration system based on a verified symbolic state enumeration system, to ensure the correctness of the theorem creation process. We partly realize the methodology with the HOL system and a simplified version of the MDG system. We prove the correctness of aspects of the simplified MDG system. We also provide a formal linkage between the HOL system and the simplified MDG system based on importing theorems [22]. Most importantly, we combine the translator correctness theorems with the importing theorems. This combination allows the low level MDG verification results to be imported into HOL in terms of the semantics of a high level language (MDG-HDL). Lessons from the work are applicable to other related systems. We chose HOL and MDG because this work is part of a large project in collaboration with the Hardware Verification group at Concordia University. They are developing a hybrid system (MDG-HOL) [17] which combines the MDG system and the HOL system.

The structure of the rest of this paper is as follows: in Section 2, we review related work. In Section 3 we briefly introduce the MDG and HOL systems. The main body of the paper is described in Section 4. Finally, our conclusion and ideas for further work are presented in Section 5.

2 Related Work

Many different technologies have been used to link systems. We will briefly review some related techniques here.

Joyce and Seger [16] presented a hybrid verification system: HOL-Voss. Several predicates were defined in the HOL system, which presented a mathematical link between the specification language of the Voss system (symbolic trajectory evaluation) [14] and that of the HOL system.

Aagaard *et al.* developed the Forte verification system [1]. Forte is a combined model checking (in Voss via symbolic trajectory evaluation) and theorem proving system (ThmTac)[1]. Both specification and implementation language are `fl` which has been deeply embedded in itself so as to be lifted. In other words, the system can execute `fl` functions in Voss and reason about the behavior of `fl` functions in ThmTac.

Rajan *et al.* [20] proposed an approach for the integration of model checking with PVS [7]: the Prototype Verification System. The μ-calculus, which consists of quantified boolean formula and predicates defined by means of the least and greatest fixpoint operators, was used as a medium for communicating between PVS and a model checker.

The PROSPER toolkit [10] provides a uniform way of linking HOL with external proof tools. The specification of its integration interface has been implemented in several languages allowing components written in these languages to be used together. A range of different external proof tools can access the

[1] ThmTac is written in `fl` and is an LCF style implementation of a higher order classical logic.

toolkit and act as servers to a HOL client. It also tags theorems produced by its plug-in with a label which can be used in the HOL system. The MDG-HOL system [17] used the PROSPER/Harness Plug-in Interface to link the HOL system and the MDG system.

Gordon [12] integrated the BDD based verification system BuDDY into HOL by implementing BDD-based verification algorithms inside HOL building on top of primitives provided. Since "LCF-Style" general infrastructure was provided, by implementing BDD primitives in HOL — as long as they are correct — not only could the standard state algorithms be efficiently and safely programmed in HOL, but it also made it possible to achieve the advantages of both theorem proving tools and state algorithms.

Hurd [15] used a different method to combine the strengths of two theorem-prover systems — Gandalf and HOL. He wrote functions to simulate the Gandalf proof according to the Gandalf logged file so reconstructing the proof in HOL to form the HOL theorems. As a result, the Gandalf proof results need not be tagged into HOL and the degree of trust is high.

We take a different approach. We focus on the verification of a symbolic state enumeration system (the MDG system) and provide a theoretical underpinning to the formal linkage of a symbolic state enumeration system and a theorem proving system (MDG and HOL). We verify the correctness of translators of the MDG system by using the HOL system and prove theorems that formally convert the MDG verification results of MDG's different applications into the traditional HOL hardware verification theorems. By combining the translator correctness theorems with the importing theorems, the MDG verification results can be imported into HOL in terms of the MDG input language (MDG-HDL). The feasibility of this approach is demonstrated in a case study that integrates two distinct applications: hardware verification (in MDG) and usability verification (in HOL). A single HOL theorem is proved that integrates the two results.

3 The MDG and HOL Systems

The MDG system is a hardware verification system based on Multiway Decision Graphs (MDGs). MDGs subsume the class of Bryant's Reduced Ordered Binary Decision Diagrams (ROBDD) [3] while accommodating abstract sorts and uninterpreted function symbols. The system combines a variety of different hardware verification applications implemented using MDGs [25]. The applications developed include: combinational verification, sequential verification, invariant checking and model checking.

The input language of MDG is MDG-HDL [25], which is a Prolog-style hardware description language that allows the use of abstract variables for representing data signals. In MDG, a circuit description file declares signals and their sort assignment, components network, outputs, initial values for sequential verification and the mapping between state variables and next state variables. In the components network, there is a large set of predefined components such as logic gates, flip-flops, registers, constants, etc. Among the predefined components there is a special component constructor known as a table which is used to describe a functional block in an implementation or specification. The TABLE constructor is similar to a truth table but allows first-order terms in rows. It also allows the description of high-level constructs as ITE (If-Then-Else) formulas and CASE formulas. A table is essentially a series of lists, together with a single final default value. The first list contains variables and cross-terms. The last element of this first list is the output of the table which must be a variable (either concrete or abstract). The other variables in the list must be concrete variables. For example, a two input `AND` gate can be described as

```
table([[x1, x2, y], [0, *, 0], [1, 0, 0]|1])        (1)
```

where "*" means "`don't care`". It states that if `x1` is `false` and `x2` is `don't care` then the output `y` is equal to `false`, if `x1` is equal to `true` and `x2` is equal to `false` then the output `y` is equal to `false`, otherwise the output `y` is equal to `true`.

HOL [13] uses higher-order logic to model and verify a system. There are two main proof methods used: forward and backward proof. In forward proof, the steps of a proof are implemented by applying inference rules chosen by the user, and HOL checks that the steps are safe. It is an LCF [11] style proof system: all derived inference rules are built on top of a small number of primitive inference rules. In backward proof, the user sets the desired theorem as a goal. Small programs written in SML [19] called tactics and tacticals are applied that break the goal into a list of subgoals. Tactics and tacticals are repeatedly applied to the subgoals until they can be proved. A justification function is also created mapping a list of theorems corresponding to subgoals to a theorem that solves the goal. In practice, forward proof is often used within backward proof to convert a goal's assumptions to a suitable form.

Theorems in the HOL system are represented by values of the ML abstract type `thm`. In a pure system (where `mk_thm` which creates arbitrary theorems is not used), a theorem is only obtained by carrying out a proof based on the primitive inference rules and axioms. More complex inference rules and tactics must ultimately call a series of primitive rules to do the job. In this way, the ML type system protects the HOL logic from the arbitrary construction of a theorem, so that every computed value of the type representing theorems is a theorem. The user can have a great deal of confidence in the results of the system provided `mk_thm` and new axioms are not used.

HOL 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, simpLib, decisionLib and bossLib`. These libraries integrate rewriting, conversion and decision procedures that automate a proof. They free the user from performing low-level proof.

4 Formally Linking a Verified MDG and HOL System

The intention of our work is to explore a way of increasing the degree of trust of the MDG system and provide a formal linkage between the HOL system and the

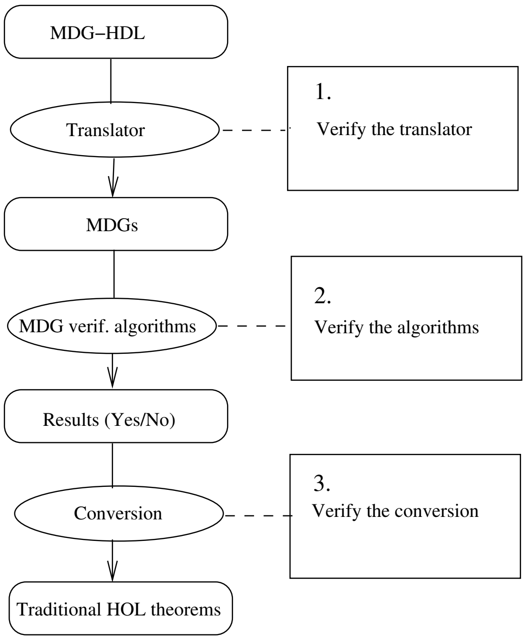

Fig. 1. Overview of the Verification Project

MDG system in terms of the MDG input language as shown in Figure 1. This work can be divided into three steps. (a) We must verify the correctness of the MDG system using the HOL system. It consists of two phases: (1) verification of the translators [23] and (2) verification of the algorithms. (b) We then must prove theorems (step 3), which formally convert the verification results of the MDG applications into traditional HOL hardware verification theorems [22]. (c) By combining the correctness theorems (theorems obtained from step 1 and step 2) of the verification of the MDG system with the importing theorems (obtained from step 3), the MDG verification results can be formalized in terms of MDG-HDL.

During this study, we concentrate on the verification of the translation phase of the MDG system (see step 1 in Figure 1) using the HOL theorem prover and importing the MDG results into HOL to form the HOL theorems (see step 3 in Figure 1) [22]. Step 2 is similar to Chou and Peled's work [5] which verifies a partial-order reduction technique for model checking. Verifying the algorithms is beyond the scope of this paper: we are primarily concerned with the linkage and how it could be combined with the correctness theorems and importing theorems. We outline the overall methodology and emphasize the importation process of the hybrid system. We not only verify the correctness of aspects of the MDG system in HOL, but also formally import the MDG results into HOL to form HOL theorems based on the semantics of the high level MDG input language (MDG-HDL) [25] rather than the semantics of the low level MDG results. Since we use a

$$\text{MDG–HDL} \xrightarrow{(1)} \text{core MDG–HDL} \xrightarrow{(2)} \text{MDGs}$$

Fig. 2. Overview of the MDG Translation Phases

deep embedding semantics, the separate translator correctness theorems can be combined with themselves and the importing theorems. The combination allows the low level MDG results to be converted into a form that can be easily reasoned about in HOL based on the semantics of MDG-HDL. We also summarize the general method about proving the `existential theorem`. This theorem is needed for importing sequential verification result into the theorem proving system.

In the remainder of this section, we will briefly introduce the individual steps that we have undertaken: verifying the translator correctness theorems, proving the general importing theorems, combining the translator correctness theorems with the importing theorems in terms of deep embedding semantics, proving the `existential theorem` and implementing our method in a case study that integrates two different applications.

4.1 Verifying the MDG Translators

In the MDG system, most of the components in the MDG-HDL library are compiled into their own core MDG-HDL tabular code first. The core MDG-HDL program can then be compiled into an internal MDG. Some components, such as registers, are implemented directly in terms of an MDG. However, in theory these components also could be implemented as tables which provides a general specification mechanism. We assume the MDG-HDL program is firstly translated into a core MDG-HDL program. The core MDG-HDL program is then translated into the MDG. In this situation, the MDG system could be specified as in Figure 2.

Adopting this approach makes the translation phase more amenable to verification. We are not verifying the actual MDG implementation. Rather our formalization of the translator is a specification of it. Once combined with a translator from core MDG-HDL to the MDG, it would be specifying the output required from the implementation. This would be used as the basis for verifying such an implementation. We thus split the problem of verifying the translator into the two problems of verifying that the implementation meets a functional specification, and that the functional specification then meets the requirement of preserving semantics. We are concerned with the latter step here. This split between implementation and specification correctness was advocated by Chirica and Martin [4] with respect to compiler correctness.

The MDG system is based on Multiway Decision Graphs which extend ROBDDs with concrete sorts, abstract sorts and uninterpreted function symbols. We define a deep embedding semantics for a subset of the MDG-HDL language. This subset is all of MDG except three predefined components (Multiplexer, Driver and constant) and the Transform construct used to apply functions. These components are omitted from our subset as they have non-boolean inputs or outputs.

We consider this subset since our aim is to explore the feasibility of this method. However, this subset allows a program to contain concrete sorts. In other words, the inputs and outputs of a `table` could be boolean sorts and concrete sorts. The concrete sort of boolean values is treated separately as it is predefined in MDG and used with most components. It is therefore treated as a special case. To cope with different types in one list, we define a new type `Mdg_Basic` in HOL. The value of the type can be either a boolean value or a string. In the rest of this paper, we will refer to the simplified version of the MDG system as "the MDG system".

We verified the first translation step of the MDG system (see phase 1 in Figure 2) based on the syntax and semantics of the MDG input language and the core MDG-HDL language using the HOL theorem prover. The syntax and the semantics of the subset MDG-HDL and core MDG-HDL are defined. A set of functions, which translate the program from MDG-HDL to core MDG-HDL is then defined. For each program in MDG-HDL, the compilation operators are defined as functions, which return their core MDG-HDL code. Translation function `TransProgMC` is applied to each MDG-HDL program so that the corresponding core MDG-HDL program is established. In other words, the relations of the translations can be represented as below:

$\forall$ `p. TransProgMC p =` *Corresponding core MDG-HDL program*

The standard approach to prove a translator between two languages is in terms of the semantics of the languages. Essentially the translation should preserve the semantics of the source language. This has the traditional form of compiler specification correctness used in the verification of a compiler [4]. An analogous method has been used in the specification and verification of the MDG system. For the translation to core MDG-HDL, the correctness theorem has been proved:

$$\vdash_{thm} \forall\ \texttt{p. SemProgram (p) = SemProgram\_Core (TransProgMC p)} \tag{2}$$

where `SemProgram` and `SemProgram_Core` are semantic functions for the MDG-HDL program and core MDG-HDL program. This theorem states that the semantics of the low level core MDG-HDL program is equal to the semantics of the high level MDG-HDL (the MDG input language). For a subset of MDG without concrete sorts we have also verified translation to a lower level formula language of decision diagrams. More details can be found in [21].

4.2 The Importing Theorems

Generally, when we use HOL to verify a design, the design is modeled as a hierarchy structure with modules divided into submodules. The submodules are repeatedly subdivided until the logic gate level is eventually reached. Both the structural and the behavioral specifications of each module are given as relations in higher-order logic. The verification of each module is carried out by proving a theorem asserting that the implementation (its structure) implements (implies) the specification (its behavior). They have the very general form:

$$\texttt{implementation} \supset \texttt{specification} \tag{3}$$

The correctness theorem for each module states that its implementation down to the logic gate level satisfies the specification. The correctness theorem for each module can be established using the correctness theorems of its submodules. In this sense the submodule is treated as a black-box. A consequence of this is that different technologies can be used to address the correctness theorem for the submodules. In particular, we can use the MDG system instead of HOL to prove the correctness of submodules.

In order to convert the MDG verification results into HOL, we formalized the results of the MDG verification applications in HOL. These formalizations have different forms for the different verification applications, i.e., combinational verification gives a theorem of one form, sequential verification gives a different form and so on. However, the most natural and obvious way to formalize the MDG results does not give theorems of the form that HOL needs if we are to use traditional HOL hardware verification techniques. Therefore, we have to convert the MDG results into a form that can be used. In other words, we proved a series of translation theorems (one for combinational verification and one for sequential verification, etc.) that state how an MDG result can be converted into the traditional HOL form:

$$\begin{array}{l}\texttt{Formalized MDG result} \supset \\ \qquad\qquad\qquad \texttt{implementation} \supset \texttt{specification}\end{array} \tag{4}$$

We have formally specified the correctness results produced by several different MDG verification applications and given general importing theorems. These theorems do not explicitly deal with the MDG-HDL semantics or multiway decision graphs. Rather they are given in terms of general relations on inputs and outputs. The theorems proved could be applied to other verification systems with similar architectures based on reachability analysis or equivalence checking.

For example, the behavioral equivalence of two abstract state machines (Figure 3) is verified by checking that the machines produce the same sequence of outputs for every sequence of inputs. The same inputs are fed to the two machines `M` and `M'` and then reachability analysis is performed on their product machine using an invariant asserting the equality of the corresponding outputs in all reachable states. This effectively introduces new "hardware" (see Figure 3) which we refer to here as `PSEQ` (the Product machine for SEQuential verification). `PSEQ` has the same inputs as `M` and `M'`, but has as output a single boolean signal (`flag`). The outputs `op` and `op'` of `M` and `M'` are input into an equality checker. On each cycle, `PSEQ` outputs true if `op` and `op'` are identical at that time, and false otherwise. The result that MDG proves about `PSEQ` is that the `flag` output is always true. This can be formalized as

$$\forall\ \texttt{ip op op' flag. PSEQ ip flag op op' M M'} \supset (\forall\ \texttt{t. flag t = T}) \tag{5}$$

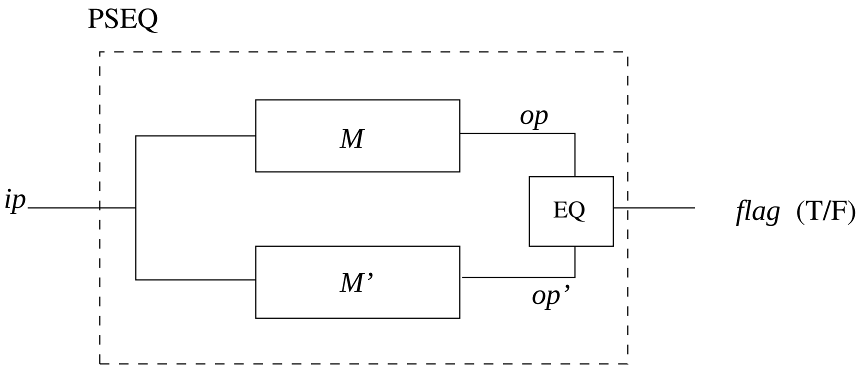

Fig. 3. The Product Machine used in MDG Sequential Verification

The corresponding importing theorem which converts MDG results to the appropriate HOL form has been obtained:

$$
\begin{array}{l}
\vdash_{thm} \forall \texttt{M M'}. \\
\quad ((\forall \texttt{ ip op op' flag}. \\
\quad\quad \texttt{PSEQ ip flag op op' M M'} \supset \forall \texttt{ t. flag t = T}) \wedge \\
\quad (\forall \texttt{ ip}. \exists \texttt{ op'. M' ip op'})) \supset \\
\quad\quad\quad (\forall \texttt{ ip op. M ip op} \supset \texttt{M' ip op})
\end{array}
\tag{6}
$$

However, the MDG results can be imported into HOL when an additional assumption ($\forall$ `ip.` $\exists$ `op'. M' ip op'`) is proved. We summarize a general method to prove the additional assumption of the design in Section 4.4.

4.3 Combining the Translator Correctness Theorems with the Importing Theorems

In this section, we introduce the basic idea about how to combine the translator correctness theorems with importing theorems based on a deep embedding semantics. This combination allows MDG results to be reasoned about in HOL in terms of the MDG input language (MDG-HDL). Ultimately in HOL we want a theorem about input language artifacts. However, the MDG verification result is obtained based on a low level data structure — an MDG representation: that is what the algorithms apply to. Therefore, the formalization of the MDG verification results in the importing theorems ought to be based on the semantics of the MDG representations. Moreover, the theorem about the translator's correctness can be used to convert the result MDG proves about the low level representation to one about the input language (MDG-HDL). By combining the translator correctness theorems with the importation theorems, we obtain the new importing theorems which convert the low level MDG verification results into HOL to form the HOL theorems in terms of the semantics of MDG-HDL. In other words, we are not only able to import the MDG result into HOL based on a verified MDG system, but also the MDG verification results can be converted directly from the MDG input files to the theorems of HOL naturally.

For example, if we check that three `NOT` gates are equivalent to a single `NOT gate`, the whole MDG verification process and the importing process can be illustrated in Figure 4. In Figure 4, step (1) gives the main part of the two circuit description files (the MDG-HDL input language), which are translated into the core MDG-HDL (tabular representations) language as shown in step (2). The core MDG-HDL languages are then translated into the MDG language (step (3)). The MDG algorithm is then applied to the MDG in order to obtain two canonical MDGs and the MDG tool checks whether the two canonical MDGs are identical and returns true or false (step (4)).

In our example the MDG tool returns true. The MDG verification results are obtained based on the low level MDGs rather than the high level language MDG-HDL. However, the translator correctness theorems state that the semantics of the low level MDG is equal to the semantics of the high level MDG-HDL (the MDG input language). By combining the low-level MDG result with the translator correctness theorems, the MDG verification results can be imported into HOL based on the semantics of the MDG input language (MDG-HDL). Therefore, the traditional HOL theorem can be obtained in terms of the semantics of the MDG input language.

In our work, we have verified the first translator. In order to demonstrate the combination of the translator correctness theorems and the importing theorems, the formalization of the MDG results will be in terms of the core MDG-HDL (see Figure 5). In fact, the principle is the same. Similar conversion can be done for further verified translators. By combining the translator correctness theorem with the importation theorems, we obtain the new importing theorems which convert the low level MDG verification results into HOL to form the HOL theorems in terms of the semantics of MDG-HDL. The combination also allows the additional assumption for sequential verification to be proved in terms of the semantics of MDG-HDL and the conversion theorem to be obtained in terms of the semantics of MDG-HDL.

Therefore, the different MDG verification applications are formalized in a way that corresponds to the semantics of the low level program (core MDG-HDL) and converted into HOL to form the HOL theorem in terms of the semantics of MDG-HDL. We have obtained theorems for both combinational verification and sequential verification. They state that a verification result about a circuit based on a low level program is equivalent to a HOL theorem based on the semantics of the high level language (MDG-HDL), i.e., that the structural specification implements the behavioral specification. For example, the new sequential verification importing theorem is obtained by instantiating (`SemProgram_Core (TransProgMC SPEC)`) and (`SemProgram_Core (TransProgMC IMP)`) for the machine `M` and `M'` in the general importing theorem (6) and using the translator correctness theorem (2).

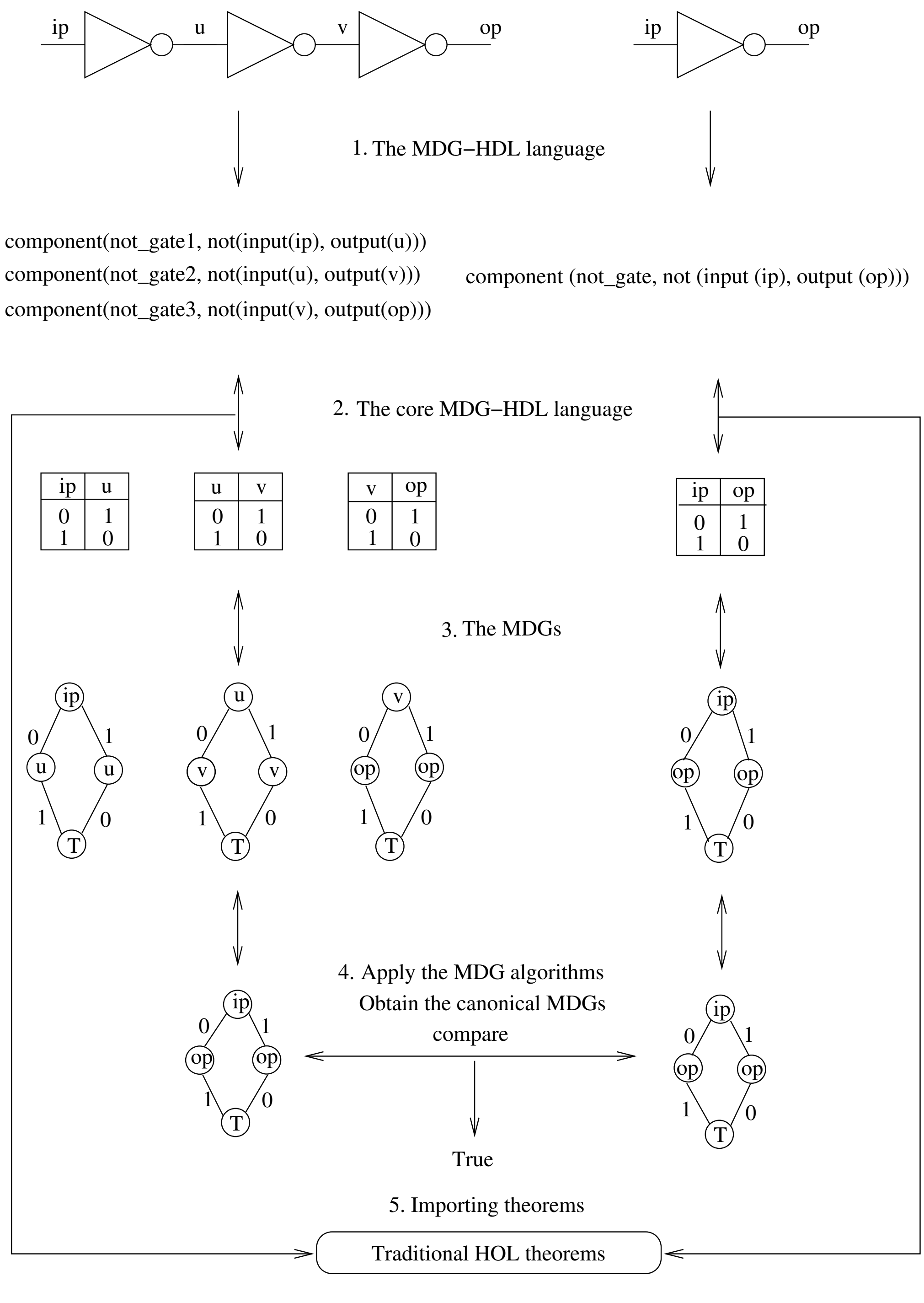

Fig. 4. The MDG Verification Process

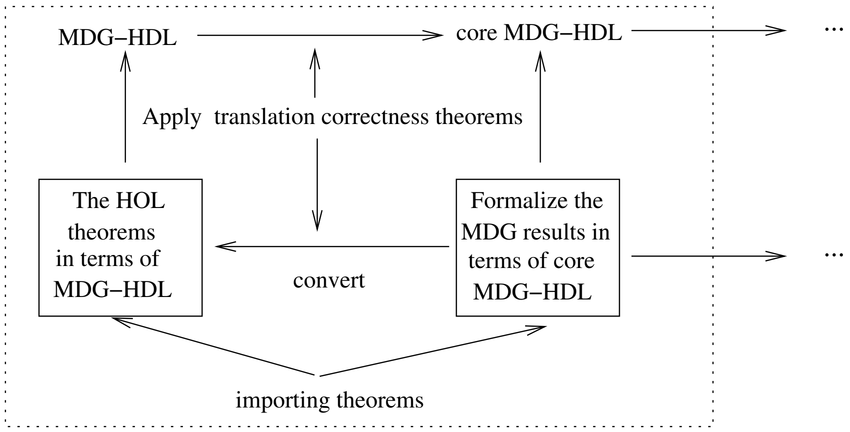

Fig. 5. Combining the Translator Correctness Theorems with Importing Theorems for a Subset Language

```
⊢thm ∀ IMP SPEC.
      (∀ ip op op' flag.
        PSEQ ip op op' flag
          (SemProgram_Core (TransProgMC SPEC))
          (SemProgram_Core (TransProgMC IMP))
                    ⊃ (∀ t. (flag t = T))) ∧
        (∀ ip. ∃  op'. SemProgram SPEC ip op') ⊃
        (∀ ip  op. SemProgram IMP ip op ⊃
                    SemProgram SPEC ip op)
```
(7)

4.4 Proving the Existential Theorem

We proved the importing theorem for sequential verification. It has the form:

```
⊢thm Formalized MDG  result ∧
        ∀ ip.  ∃ op. SPECIFICATION ip op ⊃
      (∀ ip op. (IMPLEMENTION ip op ⊃  SPECIFICATION ip op))
```

where `SPECIFICATION` represents the behavioral specification and `IMPLEMENTATION` represents the structural specification. The first assumption is discharged by the MDG verification. However, for importing the sequential verification results into HOL, a user of the hybrid system strictly needs to prove the additional assumption (an existential theorem) to ensure the correct HOL theorem can be made. This theorem states that for all possible input traces, the behavioral specification `SPECIFICATION` can be satisfied for some outputs:

$$\vdash_{thm} \forall\ \mathtt{ip}.\ \exists\ \mathtt{op}.\ \mathtt{SPECIFICATION\ ip\ op} \tag{8}$$

When we convert the MDG results into HOL to form the HOL theorems, the theorems actually state that the implementation of the design implements its specification as shown in (9).

$$\vdash_{thm} \forall\ \texttt{ip op. IMPLEMENTATION ip op} \supset\ \texttt{SPECIFICATION ip op} \tag{9}$$

This representation might meet an inconsistent model that trivially satisfies any specification. We need to verify a stronger consistency theorem against the implementation as suggested in [18], which has the form:

$$\vdash_{thm} \forall\ \texttt{ip.}\ \exists\ \texttt{op. IMPLEMENTATION ip op} \tag{10}$$

This means that for any set of input values `ip` there is a set of output values `op` which is consistent with it. This shows that the model does not satisfy a specification merely because it is inconsistent.

We have investigated a way of proving the additional assumption and the stronger consistency theorem based on the syntax and semantics of the MDG input language [23]. As we mentioned above, we prove the additional assumption because we want to make the linking process easier and remove the burden from the user of the hybrid system. We prove the stronger consistency theorem because we want to avoid an inconsistent model occurring. The above two theorems actually have the same form. We call them `existential theorems`. If we use `C` to represent any specification or implementation of a circuit, `ip` and `op` to represent the external inputs and outputs, the `existential theorem` should have the form:

$$\vdash_{thm} \forall\ \texttt{ip.}\ \exists\ \texttt{op. C ip op} \tag{11}$$

In fact, the stronger consistency theorem (10) is an `existential theorem` for the structural specification, whereas the additional assumption (8) for the importing theorem is an `existential theorem` for the behavioral specification.

The `existential theorem` is existentially quantified. We can remove hidden lines in goals of this form using the HOL tactic `EXISTS_TAC`, which strips away the leading existentially quantified variable and substitutes `term` for each free occurrence in the body. This `term` is called the `existential term`. An `existential term` of a variable is determined by one or several `output representations` of the corresponding MDG-HDL components. An `output representation` of a component represents an output function of this component, which depends on its input value and output value at the current time or an earlier time instance.

We prove the existential theorems based on the syntax and semantics of MDG-HDL [23] [9]. We provide the `output representation` for each component (mainly logic gates and flip-flops). The `existential term` of a design, which reduces the goal $\exists$ `x. t` to `t[u/x]`, is determined in terms of the corresponding `output representations`. This is very important for verifying the existential theorem, since as long as we find the `existential term` of the design, the corresponding theorem will be proved. We also provide HOL tactics for expanding the semantics of the circuit and proving the `existential theorem`. More detail can be found in [21] [24].

4.5 Case Study: Integrated Hardware and Usability Verification

So far, we have discussed how to prove translator correctness theorems and importing theorems. We have combined the translator correctness theorems with the importing theorems. The combination allows the MDG verification results to be formalized and reasoned about in HOL in terms of the semantics of MDG-HDL. We now consider a simple case study to show that this method works in practice. We show how an actual MDG verification result can be imported into HOL to form a traditional HOL theorem. Moreover, we show that the importing theorems can be practically used in HOL. In particular, we apply our approach to a simple example, integrating MDG hardware verification and HOL usability verification for a vending machine. This example was originally used to verify the absence of a common class of user errors known as post-completion errors (users missing completion tasks such as taking change after their goal, such as getting chocolate, is achieved) within the framework of a traditional hardware verification by Curzon and Blandford [8]. In this work, it was proved that the implementation of the vending machine meets its specification. A usability theorem that a usability property holds about post-completion errors based on its `specification` was then proved. By combining the above two theorems, the usability theorem based on its `implementation` was proved. In the original work all the verification was done in HOL.

We closely followed their steps. However, we used the MDG system to verify the correctness of the vending machine and imported it into HOL using the theorem (7). We then proved the `specification` based usability theorem in the HOL system. By combining those two theorems, first the correctness theorem of the vending machine which is verified in MDG (the importing theorem), second the `specification` based usability theorem which is proved in HOL, we obtained the `implementation` based usability theorem. Therefore, the importing theorem (the correctness theorem) cannot only be imported into HOL but also can be used in HOL.

We first did a hardware verification of the vending machine in MDG. The theorem about the formalization of the MDG verification result can be tagged into HOL in terms of the semantics of core MDG-HDL.

$$
\begin{array}{l}
\vdash_{thm} \forall\ \texttt{ip flag op op'}. \\
\qquad \texttt{PSEQ ip flag op op'} \\
\qquad\quad \texttt{(SemProgram\_Core (TransProgMC } \mathit{Vend\_Imp\_Syn}\texttt{))} \\
\qquad\quad \texttt{(SemProgram\_Core (TransProgMC } \mathit{Vend\_Spe\_Syn}\texttt{))} \\
\qquad\qquad \supset (\forall\ \texttt{t. (flag t = T))}
\end{array}
\tag{12}
$$

where *Vend_Imp_Syn* and *Vend_Spe_Syn* stand for syntax of the implementation and specification of the vending machine in terms of MDG-HDL. As stated in Section 4.3, the importing theorem for the vending machine can be obtained by instantiating theorem (7) with the syntax of its implementation and specification (*Vend_Spe_Syn* and *Vend_Imp_Syn*). We obtain theorem `Import_Vend_Thm`

$$\vdash_{thm} (\forall\ \texttt{ip flag op op'}.\ \texttt{PSEQ ip flag op op'}\ (\texttt{SemProgram\_Core}\ (\texttt{TransProgMC}\ \mathit{Vend\_Imp\_Syn}))\ (\texttt{SemProgram\_Core}\ (\texttt{TransProgMC}\ \mathit{Vend\_Spe\_Syn}))\ \supset (\forall\ \texttt{t}.\ (\texttt{flag t = T}))) \wedge \forall\ \texttt{ip}.\ \exists\ \texttt{op'}.\ \texttt{SemProgram}\ \mathit{Vend\_Spe\_Syn}\ \texttt{ip op'} \supset (\forall\ \texttt{ip op}.\ \texttt{SemProgram}\ \mathit{Vend\_Imp\_Syn}\ \texttt{ip op} \supset \texttt{SemProgram}\ \mathit{Vend\_Spe\_Syn}\ \texttt{ip op}) \tag{13}$$

We then prove the `existential theorem` for the behavioral specification in terms of the semantics of MDG-HDL.

$$\vdash_{thm}\ \forall\ \texttt{ip}.\ \exists\ \texttt{op'}.\ (\texttt{SemProgram}\ \mathit{Vend\_Spe\_Syn}\ \texttt{ip op'}) \tag{14}$$

Finally, the conversion theorem can be obtained by discharging the formalization theorem (12) and the existential theorem (14) from the importing theorem (13). This theorem states that the implementation implies the specification.

$$\vdash_{thm}\ \forall\ \texttt{ip op}.\ \texttt{SemProgram}\ \mathit{Vend\_Imp\_Syn}\ \texttt{ip op} \supset \texttt{SemProgram}\ \mathit{Vend\_Spe\_Syn}\ \texttt{ip op} \tag{15}$$

We then prove the `specification` based usability theorem in the HOL system. The general user model for a vending machine is defined as `CHOC_MACHINE_USER ustate op ip`. This specifies a simple form of rational user behaviour. We prove that if a user behaves in this rational way, they will not make post completion errors with the particular vending machine in question (though they may with other poorer designs). It specifies concrete types for the machine and user state, a list of pairs of lights and the actions associated with them, history functions that represent the possessions of the user, functions that extract the part of the user state that indicates when the user has finished and has achieved their main goal and an invariant that indicates the part of the state that the user intends to be preserved after the interaction. The details of the user model are not important for our main argument here about integrating the results: the interested reader should refer to [8].

```
⊢def CHOC_MACHINE_USER ustate op ip =
  USER
   [(CoinLight,InsertCoin); (ChocLight,PushChoc);
    (ChangeLight,PushChange)]
  (CHOC_POSSESSIONS UserHasChoc GiveChoc CountChoc UserHasChange
    GiveChange CountChange UserHasCoin InsertCoin CountCoin)
  UserFinished
  UserHasChoc
  (VALUE_INVARIANT (CHOC_POSSESSIONS UserHasChoc GiveChoc CountChoc
    UserHasChange GiveChange CountChange
    UserHasCoin InsertCoin CountCoin))
  ustate op ip
```

The usability of a vending machine is defined as `CHOC_MACHINE_USABLE ustate op ip` in terms of a user-centric property. It states that if at any time, t, a user approaches the machine when its coin light is on, then they will at some time, t1, have both chocolate and change: they will not make post-completion errors.

```
⊢def CHOC_MACHINE_USABLE ustate op ip =
  ∀ t. ∼ (UserHasChoc ustate t) ∧
       ∼ (UserHasChange ustate t) ∧
       (UserHasCoin ustate t) ∧
       (VALUE_INVARIANT (CHOC_POSSESSIONS UserHasChoc GiveChoc
            CountChoc UserHasChange GiveChange CountChange
            UserHasCoin InsertCoin CountCoin) ustate t) ∧
        ((CoinLight op t)= BOOL T) ⊃
           ∃ t1. (UserHasChoc ustate t1) ∧
                        (UserHasChange ustate t1)
```

The `specification` based usability theorem states that if all the external inputs and outputs are boolean, a user acts rationally and the machine behaves according to its specification, then the usability property will hold.

```
⊢thm ∀ ustate op ip.
   Boolean ip op ∧
      CHOC_MACHINE_USER ustate op ip ∧
            CHOC_MACHINE_SPEC ip op ⊃
                   CHOC_MACHINE_USABLE ustate op ip       (16)
```

where predicate `Boolean` checks if all the external wires are boolean values. This is because the inputs of a `TABLE` could be either a concrete type variable or a boolean variable. This predicate ensures the external wires have proper values.

The `implementation` based usability theorem can be proved in terms of the above theorems (15) and (16). It (17) states that if the inputs and outputs are boolean, a user acts rationally according to the user model and the machine behaves according to its `implementation`, then the usability property will hold.

```
⊢thm ∀ ustate op ip.
   Boolean ip op ∧
      CHOC_MACHINE_USER ustate op ip ∧
            CHOC_MACHINE_IMPL ip op ⊃
                   CHOC_MACHINE_USABLE ustate op ip       (17)
```

From this example, we have shown that a system can be verified in two parts. One part of the proof can be done in MDG and the other part of the proof can be done in HOL. The division allows MDG to be used when it would be easier than obtaining the result directly in HOL. We have provided a formal linkage between the MDG system and the HOL system, which allows the MDG verification results to be formally imported into HOL to form the HOL theorem. We do not simply assume that the results proved by MDG are directly equivalent to the result that would have been proved in HOL. The linkage is based on the importing theorems giving a greater degree of trust. We have made use of the importing theorem. In other words, the MDG verification result not only can be imported into HOL to form the HOL theorem, it also can be used as part of a compositional verification in HOL. We have also shown that two different applications (hardware verification and usability verification) suited to two different tools can be combined together.

5 Conclusions

We have described a methodology which provides a formal linkage between a symbolic state enumeration system and a theorem proving system based on a verified symbolic state enumeration system. The methodology involves the following three steps. The first step is to verify correctness of the symbolic state enumeration system in an interactive theorem proving system. Some symbolic state enumeration based systems such as MDG consist of a series of translators and a set of algorithms. We need to prove the translators and algorithms to ensure the correctness of the system. We have not verified the algorithms in this work, but concentrated on the translators. For verifying the translators, we need to define the deep embedding semantics and translation functions. We have to make certain that the semantics of a program is preserved in those of its translated form. This work increases our trust in the results of the symbolic state enumeration system.

The second step of the methodology is to prove importing theorems in the proof system about the results from the symbolic state enumeration system. We need to formalize the correctness results produced by different hardware verification applications using the theorem proving system. The formalization is based on semantics of the low level language (decision graph). We need to prove a theorem in each case that translates them into a form usable in the theorem proving system. In other words, we need to provide the theoretical justification for linking two systems.

The third step is to combine the translator correctness theorems with the importing theorems. This combination allows the verification results from the state enumeration system to be formalized in terms of the semantics of a low level language that the algorithms manipulate, and the result is strictly about, and imported in terms of, the semantics of a high level language (HDL). Therefore, we are able to import the result into the theorem proving system based on the semantics of the input language of a verified symbolic state enumeration system.

We have implemented this methodology on a simplified version of the MDG system and the HOL system, and provided a formal linkage by using the above mentioned steps. We have verified aspects of correctness of a simplified version of the MDG system. We have provided a formal linkage between the MDG system and the HOL system based on importing theorems [22]. Most importantly, we have combined the translator correctness theorems with the importing theorems. This combination allows the low level MDG verification results to be imported into HOL in terms of the semantics of a high level language (MDG-HDL). We have also summarized a general method which is used to prove the `existential theorem` for the specification and implementation of the design [24]. This work makes the linking process easier and removes the burden from the user of the hybrid system. The feasibility of this approach has been demonstrated in a case study: integrating hardware verification and usability verification for a vending machine. However, for importing the MDG verification result into HOL, we have to prove the `existential theorem` for the specification of the design. The behaviour specifications must be in the form of a finite state machine or table description.

In ongoing work we are verifying the translator from core MDG-HDL to MDGs. This has already been done, and integrated with the importing theorems, for a smaller subset of the language (only using boolean sorts). We will also prove importing theorems for other MDG applications, verify more complex examples and use our method in a combined system. In the longer term we will verify the MDG algorithms and so integrate them with the existing work.

Acknowledgments This work is funded by EPSRC grant GR/M45221, and a studentship from the School of Computing Science, Middlesex University. Travel funding was provided by the British Council, Canada.

References

1. M. D. Aagaard, R. B. Jones, R. Kaivola, and C. J. H. Seger. Formal verification of iterative algorithms in microprocessors. *DAC*, June 2000.
2. R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions in Computers*, 35(8):677–691, August 1986.
3. R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computer Surveys*, 24(3), September 1992.
4. L. M. Chirica and D. F. Martin. Toward compiler implementation correctness proofs. *ACM Transactions on Programming Languages and Systems*, 8(2):185–214, April 1986.
5. C. T. Chou and D. Peled. Formal verification of a partial-order reduction technique for model checking. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, number 1055 in Lecture Notes in Computer Science, pages 241–257, 1996.
6. F. Corella, Z. Zhou, X. Song, M. Langevin, and E. Cerny. Multiway decision graphs for automated hardware verification. *Formal Methods in System Design*, 10(1):7–46, 1997.
7. J. Crow, S. Owre, J. Rushby, N. Shankar, and M. Srivas. A tutorial introduction to PVS. http://www.dcs.gla.ac.uk/prosper/papers.html, 1999.
8. P. Curzon and A. Blandford. Using a verification system to reason about post-completion errors. In *Participants Proceedings of DSV-IS 2000: 7th International Workshop on Design, Specification and Verification of Interactive Systems, at the 22nd International Conference on Software Engineering.*
9. P. Curzon, S. Tahar, and O. Aït-Mohamed. Verification of the MDG components library in HOL. In Jim Grundy and Malcolm Newey, editors, *Theorem Proving*

in Higher-Order Logics: Emerging Trends, pages 31–46. Department of Computer Science, The Australian National University, 1998.
10. L. A. Dennis, G. Collins, M. Norrish, R. Boulton, K. Slind, G. Robinson, M. Gordon, and T. Melham. The PROSPER toolkit. In *The Sixth International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 1785 in Lecture Notes in Computer Science. Springer Verlag, 2000.
11. M. J. Gordon, R. Milner, and C. P. Wadsworth. Edinburgh LCF: A mechanised logic of computation. Number 78 in Lecture Notes in Computer Science, 1979.
12. M. J. C. Gordon. Reachability programming in HOL98 using BDDs. In Mark Aagaard and John Harrison, editors, *Theorem Proving in Higher Order Logics*, number 1869 in Lecture Notes in Computing Science, pages 179–196. Springer-Verlag, Aug. 2000.
13. M. J. C. Gordon and T. F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher-order Logic*. Cambridge University Press, 1993.
14. S. Hazelhurst and C. J. H. Seger. *Symbolic trajectory evaluation*. Springer Verlag. New York, 1997.
15. J. Hurd. Integrating GANDALF and HOL. Technical Report 461, University of Cambridge, Computer Laboratory, April 1999.
16. J. Joyce and C. Seger. Linking BDD-based symbolic evaluation to interactive theorem-proving. In *the 30th Design Automation Conference*, 1993.
17. S. Kort, S. Tahar, and P. Curzon. Hierarchical verification using an MDG-HOL hybrid tool. In T. Margaria and T. Melham, editors, *11th IFIP WG 10.5 Advanced Research Working Conference (CHARME'2001)*, number 2144 in Lecture Notes in Computer Science, pages 244–258, Livingston, Scotland, UK, September 2001. Springer-Verlag.
18. T. F. Melham. *Higher Order Logic and Hardware Verification*. Cambridge Tracts in Theoretical Computer Science 31. Cambridge University Press, 1993.
19. L. C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 1991.
20. S. Rajan, N. Shankar, and M. K. Srivas. An integration of model-checking with automated proof checking. In Pierre Wolper, editor, *Computer-Aided Verification*, number 939 in Lecture Notes in Computer Science, pages 84–97. Springer-Verlag, 1995.
21. H. Xiong. *Providing a Formal Linkage between MDG and HOL Based on a Verified MDG System*. School of Computing Science, Middlesex University, January 2002. Ph.D. thesis.
22. H. Xiong, P. Curzon, and S. Tahar. Importing MDG verification results into HOL. In *Theorem Proving in Higher Order Logics*, number 1690 in Lecture Notes in Computer Science, pages 293–310. Springer-Verlag, September 1999.
23. H. Xiong, P. Curzon, S. Tahar, and A. Blandford. Embedding and verification of an MDG-HDL translator in HOL. In *TPHOLs 2000 Supplemental Proceedings*, Technical Reprot CSE-00-009, pages 237–248, August 2000.
24. H. Xiong, P. Curzon, S. Tahar, and A. Blandford. Proving existential theorems when importing results from MDG to HOL. In Richard J. Boulton and Paul B. Jackson, editors, *TPHOLs 2001 Supplemental Proceedings*, Informatic Research Report EDI-INF-RR-0046, pages 384–399, September 2001.
25. Z. Zhou and N. Boulerice. *MDG Tools (V1.0) User Manual*. University of Montreal, Dept. D'IRO, 1996.