Using induction and BDDs to model check invariants

David Déharbe and Anamaria Martins Moreira
UFRN - DIMAp
Universidade Federal do Rio Grande do Norte
Departamento de Informática e Matemática Aplicada
Campus Universitário
Lagoa Nova
59072-970 Natal, RN
Brazil

Abstract
We present an inductive characterization for an invariant to stand in a given finite-state transition system. We show how this characterization can be computed by means of BDD-based operations, without performing a fixpoint iteration over sets of states as the CTL symbolic model checking algorithm does.

1 INTRODUCTION

Model checking is an algorithm for computing the truth of a formula expressed in some logic in a given model. Clarke & Emerson (1981) and Queille & Sifakis (1981) presented independently a fully automatic model checking algorithm for the branching time temporal logic CTL in finite-state transition systems, linear in the size of the formula and in the size of the model. This algorithm has been used to verify systems of up to several million states and transitions (Clarke, Emerson & Sistla 1986), which is enough in practice only for small systems.

A big step forward was made when McMillan (1993) proposed a new model checking algorithm for CTL, based on fixpoint computations of sets of states. In this algorithm, called symbolic model checking, binary decision diagrams (Bryant 1986) are used to represent both the transitions and the states of the model. Since sets of states are represented in intention by their characteristic functions, the size of the verified model is not bound by the memory of the computer carrying the verification and it is possible to verify systems that have several orders of magnitude more states.

However, most of the systems designed today are much larger and, in order to achieve verification, symbolic model checking must be combined, often manually, with other techniques, such as abstraction and composition. It is important to note that most of the involved techniques are only good in a heuristic sense: although theoretically their computational complexity is extremely large, practically, on many examples, they prove to be efficient. Successful verification of industrial hardware designs has been possible with these techniques. However, even with these combinations, the verification of large and complex designs for requires too much resources and is not possible, due to the complexity of the computations involved in the fixpoint computations.

An important class of properties are invariants: a property is an invariant of a model if it holds for every reachable state. In practice, specifications are often composed of a large number of invariants, e.g. (Anderson, Beame, Burns, Chan, Modugno, Notkin & Reese 1996). In this paper we present a theorem that gives a sufficient condition for a given property to be an invariant of a given model. We then show how the symbolic model checking algorithm for CTL can be altered to include to check, using induction, if the sufficient condition is realized whenever the property to be proved is an invariant.

Outline: In Sections 2 to 5, we overview the foundations of symbolic model checking: Kripke structures, the class of models considered; binary decision diagrams, an efficient data structure to represent such structures; elements of fixpoint theory in lattices; syntax and semantics of computation tree logic. In Section 6, we present the main result of the paper: a sufficient condition for a given property to be an invariant of a given model. We also show how to incorporate the computation of this sufficient condition in CTL model checking.

2 KRIPKE STRUCTURES

Let P be a finite set of boolean propositions. A Kripke structure over P is a quadruple $M = (S, T, I, L)$ where:

- S is a finite set of states.
- $T \subseteq S \times S$ is a transition relation, such that $\forall s \in S, \exists s' \in S, (s, s') \in T$.
- $I \subseteq S$ is the set of initial states.
- $L : S \rightarrow 2^P$ is a labeling function. L is injective and associates with each state a set of boolean propositions true in the state.

A path π in the Kripke structure M is an infinite sequence of states $s_1, s_2, \ldots$ such that $\forall i \geq 1, (s_i, s_{i+1}) \in T$. $\pi(i)$ is the i^{th} state of π. The set of states reachable from I, denoted RS, is the set of states s such that there is a path from an initial state to this state:

$$RS = \{s \in S \mid \exists \pi, ((\pi(1) \in I) \wedge \exists i \geq 1, (\pi(i) = s))\} \tag{1}$$

A property f is an *invariant* of M, if f is true of each state s of RS.

2.1 Characteristic functions

Let $M = (S, T, I, L)$ be a Kripke structure over $P = \{v_1, \dots, v_n\}$. Let $\mathbf{v}$ denote $(v_1, \dots v_n)$. The characteristic function of a state $s \in S$, denoted $[s]$, is defined as:

$$[s](\mathbf{v}) \quad = \quad \left(\left(\bigwedge_{v_i \in L(s)} v_i \right) \wedge \left(\bigwedge_{v_i \notin L(s)} \bar{v}_i \right) \right)$$

The definition of the characteristic function is extended to sets of states with the following definitions:

$$\begin{aligned} [\{\}](\mathbf{v}) &= \mathit{false} \\ [\{x\} \cup X](\mathbf{v}) &= [x](\mathbf{v}) \vee [X](\mathbf{v}) \end{aligned}$$

Let $P' = \{v'_1, \dots v'_n\}$ be a set of fresh boolean propositions. The characteristic function of a transition $t = (s_1, s_2) \in T$, denoted $[t]$, is defined as:

$$[t](\mathbf{v}, \mathbf{v}') \quad = \quad [s_1](\mathbf{v}) \wedge [s_2](\mathbf{v}')$$

This definition can be extended to represent sets of transitions as for sets of states.

To simplify notations, in the rest of the paper we will identify $[X]$ with X.

2.2 State space traversal

Let $M = (S, T, I, L)$ be a Kripke structure over P. The image of a set of states $X \subseteq S$ is the set of states that can be reached in one transition from X:

$$\{s \in S \mid \exists s' \in X, (s', s) \in T\}$$

The characteristic function of the image of X, denoted $Img(X)$, is:

$$Img(X)(\mathbf{v}') \quad = \quad \exists \mathbf{v}, X(\mathbf{v}) \wedge T(\mathbf{v}, \mathbf{v}')$$

Conversely, the inverse image of a set of states $X \subseteq S$, is the set of states from which X can be reached in one transition:

$$\{s \in S \mid \exists s' \in X, (s, s') \in T\}$$

The characteristic function of the inverse image of a set of states X, denoted $Pre(X)$, is:

$$Pre(X)(\mathbf{v}') \quad = \quad \exists \mathbf{v}', X(\mathbf{v}') \wedge T(\mathbf{v}, \mathbf{v}')$$

3 BINARY DECISION DIAGRAMS

Binary Decision Diagrams (BDDs for short) form a heuristically efficient data structure to represent formulas of the propositional logic. Let P be a totally ordered finite set of boolean propositions. Let f be a boolean formula over P, $bdd(f)$ is the BDD representing f, and $|bdd(f)|$ is the size if this BDD. Bryant (1986) showed that BDDs are a canonical representation: two equivalent formulas are represented with the same BDD:

$$f \Leftrightarrow g \text{ iff } bdd(f) = bdd(g)$$

Moreover, most boolean operations can be performed efficiently with BDDs. Let $|b|$ denote the size of BDD b:

- $bdd(\neg f)$ is computed in constant time $O(1)$*.
- $bdd(f \vee g)$ is realized in $O(|bdd(f)|.|bdd(g)|)$.
- $bdd(\exists x, f)$ is performed in $O(|bdd(f)|^2)$.

In this paper, we will use usual boolean operators to denote the corresponding operation on BDDs, e.g. $bdd(f) \vee bdd(g) = bdd(f \vee g)$.

We explain the basic principles of the BDD representation on an example. Fig. 1 presents the binary decision tree for the 2-bit comparison: $x_1 \Leftrightarrow y_1 \wedge x_2 \Leftrightarrow y_2$. The binary tree representation of a formula is exponential in the number of free boolean propositions in the formula.

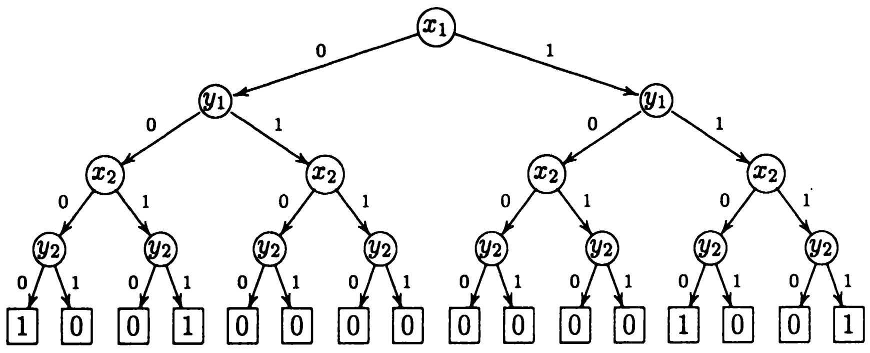

Figure 1 Binary tree for $x_1 \Leftrightarrow y_1 \wedge x_2 \Leftrightarrow y_2$.

The corresponding BDD is obtained by repeatedly applying the following rules:

*Actually, negation on BDDs as presented in (Bryant 1986) is a linear operation. However, Madre (1990) proposed a slight modification of the data structure that resulted in reducing the negation to a constant operation. Schematically, the modification consists in tagging edges and identifying the representations of f and $\neg f$. Current BDD packages integrate this modification.

- remove duplicate terminal vertices,
- remove duplicate vertices bottom-up,
- remove opposite vertices,
- remove redundant tests.

Fig. 2 presents the BDD of the 2 bit comparator with ordering $x_1 < y_1 < x_2 < y_2$. For an n-bit comparator, the BDD representation is linear with ordering $x_1 < y_1 < \ldots < x_n < y_n$ and thus is exponentially better than the binary tree representation. However, with ordering $x_1 < \ldots < x_n < y_1 < \ldots < y_n$, the BDD representation is exponential.

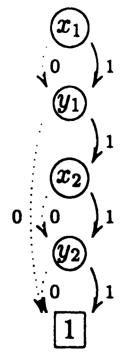

Figure 2 BDD for $x_1 \Leftrightarrow y_1 \wedge x_2 \Leftrightarrow y_2$.

Bryant (1986) showed that some functions have an exponential BDD representation for any variable ordering, and that finding the optimum variable ordering is NP-hard. However, in practice, heuristic methods generally achieve a good variable ordering, when such ordering exists.

In a Kripke structure, states, transitions and sets thereof can be characterized with propositional logic formulas. These formulas can be represented and manipulated via their BDD representation. Therefore, BDDs are an efficient data structure to perform computations on Kripke structures.

4 LATTICES AND FIXPOINTS

A lattice is a set, a partial order on the elements of this set, a least element $\bot$, and a greatest element $\top$.

Let P be a finite set of atomic propositions. Let $M = (S, T, I, L)$ be a finite Kripke structure over P. We consider the lattice $(2^S, \subseteq)$ of subsets of S with set inclusion as the ordering. The empty set $\{\}$ and S are respectively the least and greatest elements of this lattice. Since a subset of S can be identified with its characteristic function, this lattice can also be interpreted as the lattice of characteristic functions, with boolean implication as ordering, *false* is the least element, and the characteristic function of S is the greatest element.

A function $\tau : 2^S \rightarrow 2^S$ is called a ***predicate transformer***. τ is ***monotonic*** iff $P \subseteq Q$ implies $\tau(P) \subseteq \tau(Q)$. Tarski (1955) showed that if τ is monotonic, τ has a least fixpoint, denoted $\mathbf{lfp}Z[\tau(Z)]$, and a greatest fixpoint, denoted $\mathbf{gfp}Z[\tau(Z)]$:

$$\begin{array}{rcccl} \mathbf{lfp}Z[\tau(Z)] & = & \cap\{Z \mid \tau(Z) = Z\} & = & \cup_i \tau^i(\mathit{false}) \\ \mathbf{gfp}Z[\tau(Z)] & = & \cup\{Z \mid \tau(Z) = Z\} & = & \cap_i \tau^i(\mathit{true}) \end{array}$$

5 COMPUTATION TREE LOGIC

5.1 Syntax

The set $T_{ctl}(P)$ of Computation Tree Logic (CTL for short) formulas over a set of propositions P is the smallest set such that $P \subseteq T_{ctl}(P)$ and, if f and g are in $T_{ctl}(P)$, then $\neg f$, $f \wedge g$, $\mathbf{EX}f$, $\mathbf{EG}f$, and $\mathbf{E}[f\mathbf{U}g]$ are in $T_{ctl}(P)$.

5.2 Semantics

The semantics of CTL are defined with respect to a Kripke structure $M = (S, T, I, L)$ over a set of atomic propositions P. If f is in $T_{ctl}(P)$, $M, s \models f$ means that f holds at state s of M.

Let f and g be in $T_{ctl}(P)$, then

1. $M, s \models p$ iff $p \in L(s)$.
2. $M, s \models \neg f$ iff $M, s \not\models f$.
3. $M, s \models f \wedge g$ iff $M, s \models f$ and $M, s \models g$.
4. $M, s \models \mathbf{EX}f$ iff there exists a state s' of M such that $(s, s') \in T$ and $s' \models f$. s has a successor where f is valid.
5. $M, s \models \mathbf{EG}f$ iff there exists a path π of M such that $\pi(1) = s$ and $\forall i \geq 1, M, \pi(i) \models f$. s is at the start of a path where f holds globally.
6. $M, s \models \mathbf{E}[f\mathbf{U}g]$ iff there exists a path π of M such that $\pi(1) = s$ and and $\exists i \geq 1, M, \pi(1) \models g \wedge \forall j, i > j \geq 1, M, \pi(j) \models f$. s is at the start of a path where g holds eventually and f holds until g becomes valid.

A formula f is valid in structure M if it is valid for all initial states:

$$M \models f \text{ iff } \forall s \in I, M, s \models f.$$

CTL symbolic model checking uses the BDD representations of the characteristic functions of sets of states and transitions. The algorithm is based on the fixpoint characterization of the different temporal operators of CTL defined in (Clarke & Emerson 1981). For instance,

$$\mathbf{E}[f\mathbf{U}g] \quad = \quad \mathbf{lfp}Z[g \vee (f \wedge \mathbf{EX}Z)]$$

Invariants in CTL: Note that, in CTL, an invariant has the form $\mathbf{AG}f$, which is $\neg\mathbf{E}[\mathit{true}\mathbf{U}\neg f]$. It therefore requires the computation of a least fixpoint associated to the operator **EU**. This fixpoint can be implemented with a loop construct that repeatedly performs inverse image computations until the fixpoint is reached*. In the worst case, the number of iterations is $|S|$, the number of states of the model.

6 SUFFICIENT CONDITION OF INVARIANCE

Let $M = (S, T, I, L)$ be a Kripke structure over P. In this section, we introduce a predicate transformer τ over the lattice $(2^S, \subseteq)$ and use it to characterize the reachable states as a least fixpoint. We then show, using induction principles, that if, in M, the initial states I satisfy f and if f is preserved by the transition relation T, then f is an invariant of M. Representing characteristic functions with BDDs, it is possible to decide whether this condition holds with the conventional symbolic model checking algorithm.

6.1 Theoretical results

Let τ be the predicate transformer defined as $\tau(Z) = I \cup Z \cup Img(Z)$. We are first going to show that τ is monotonic (Lemma 1). Then we prove that the set of reachable states is the least fixpoint of τ (Lemma 2 and Theorem 1). We then give the sufficient condition for a property to be an invariant of M (Lemma 3 and Theorem 2).

Lemma 1 *The predicate transformer $\tau(Z) = I \cup Z \cup Img(Z)$ is monotonic.*

Proof: Let $P_1 \subseteq P_2$. To prove that $\tau(P_1) \subseteq \tau(P_2)$, consider a state $s \in \tau(P_1)$. Thus, $s \in I$ or $s \in P_1$ or $s \in Img(P_1)$:

- if $s \in I$, then $s \in \tau(P_2)$.
- if $s \in P_1$, since $P_1 \subseteq P_2$, then $s \in P_2$ as well, and $s \in \tau(P_2)$.
- if $s \in Img(P_1)$, since $P_1 \subseteq P_2$, then $Img(P_1) \in Img(P_2)$ and $s \in Img(P_2)$; thus $s \in \tau(P_2)$.

*Iwashita, Nakata & Hirose (1996) proposed a variant algorithm for a fragment of CTL, based on image computations. In this work, the verification of invariant also requires a fixpoint computation.

In each case, $s \in \tau(P_2)$; thus $\tau(P_1) \subseteq \tau(P_2)$. □

Lemma 2 *The set of reachable states RS is a fixpoint of the predicate transformer $\tau(Z) = I \cup Z \cup Img(Z)$.*

Proof: From the definition of τ, the inclusion $RS \subseteq \tau(RS)$ is trivial. From the definition of RS, we also have the inclusions $I \subseteq RS$ and $Img(RS) \subseteq RS$. Therefore $\tau(RS) \subseteq RS$. Thus, RS is a fixpoint. □

Theorem 1 (Characterization of the reachable states) *The set of reachable states RS is the least fixpoint of the predicate transformer $\tau(Z) = I \cup Z \cup Img(Z)$.*

Proof: Since M is finite and τ is monotonic (from Lemma 1), the least fixpoint is $\cup_i \tau^i(\mathit{false})$. It is therefore sufficient to show that $RS = \mathbf{lfp}Z[\tau(Z)] = \cup_i \tau^i(\mathit{false})$.

- By application of Tarski's theorem, and since RS is a fixpoint of τ (from Lemma 2), then $\cup_i \tau^i(\mathit{false}) = \mathbf{lfp}(Z)[\tau(Z)] \subseteq RS$.
- $RS \subseteq \cup_i \tau^i(\mathit{false})$ is proved by induction on the position on the paths for each the reachable states. From Equation 1, if $s \in RS$, then there is a path $\pi = s_1, s_2, \ldots$ with $s_1 \in I$ and $\exists i \geq 1, s_i = s$. We show that, if s appears at the i position on a path, then $s \in \tau^i(\mathit{false})$.

  - The basis case is trivial: If $i = 1$, then $s \in I$, and therefore $s \in \tau^1(\mathit{false}) = I$.
  - The induction hypothesis states that the above property stands for every s and every $i \leq n$. Let s be a state of a path $s_1, \ldots s_n, s_{n+1}$, such that $s_{n+1} = s$. By induction hypothesis, $s_n \in \tau^n(\mathit{false})$. Since $(s_n, s) \in T$, then $s \in Img(\tau^n(\mathit{false}))$ and therefore $s \in \tau(\tau^n(\mathit{false})) = \tau^{n+1}(\mathit{false})$. □

Lemma 3 *Let $M = (S, T, I, L)$ be a Kripke structure. If f is valid for all initials states of M (i.e. $I \subseteq f$), and T preserves f (i.e. $Img(f) \subseteq f$), then f is a fixpoint of the predicate transformer $\tau(Z) = I \cup Z \cup Img(Z)$.*

Proof: Trivially, $f \subseteq I \cup f \cup Img(f) = \tau(f)$. Also, by hypothesis $I \subseteq f$ and $Img(f) \subseteq f$, thus $\tau(f) = I \cup f \cup Img(f) \subseteq f$. □

Theorem 2 (Sufficient condition for invariance) *Let $M = (S, T, I, L)$ be a Kripke structure. If f is valid for all initials states of M (i.e. $I \subseteq f$), and T preserves f (i.e. $Img(f) \subseteq f$), then all reachable states verify f: $RS \subseteq f$.*

```
routine ModelCheckA (var $f$ : $T_{CTL}(P)$, var $M$: model) : boolean
    var $I$: BDD
    $I \leftarrow M.I$ -- Initial states of $M$
    if ($f = \mathbf{AG}g$) then
        var G : BDD;
        $G \leftarrow ModelCheck(g, M)$
        if ($I \Rightarrow G$) then
            if ($Img(G, M) \Rightarrow G$) then
                $\rightarrow$ true
            else
                $\rightarrow$ ($I \Rightarrow ModelCheck(f, M)$)
            end if
        else
            $\rightarrow$ false
        end if
    else
        $\rightarrow$ ($I \Rightarrow ModelCheck(f, M)$)
    end if
end routine ModelCheckA
```

Figure 3 Modified model checking algorithm

Proof: Consider the predicate transformer τ. Since $I \subseteq f$ and $Img(f) \subseteq f$, applying Lemma 3, f is a fixpoint of τ. Also from Theorem 1, RS is the least fixpoint of τ. Thus, $RS \subseteq f$. □

Note that the condition $I \subseteq f$ and $Img(f) \subseteq f$ is only a sufficient and not a necessary condition for $RS \subseteq f$. As a matter of fact, let $s \in (S \backslash RS) \cap f$ be a non-reachable state where f stands. If $Img(\{s\}) \neg\subseteq f$, the sufficient condition is not satisfied, but we cannot conclude about $RS \subseteq f$.

6.2 Application

Let *ModelCheck* be the conventional CTL symbolic model checking algorithm defined in (McMillan 1993). This algorithm has been implemented in several tools and successfully used to verify hardware, software and protocols specifications.

ModelCheck takes as argument a CTL formula f to be verified and returns true or false. Figure 3 presents *ModelCheckA*, an alternative algorithm. In *ModelCheckA*, if f is of the form $\mathbf{AG}g$, the conditions presented in Theorem 2 are tested. If they are verified, then g is an invariant of M and $f = \mathbf{AG}g$ holds.

Note that, on the one hand, the complexity of the test for the validity of the sufficient condition is the complexity of one call to *Img*. On the other hand, the complexity of conventional symbolic model checking, in the worst case, corresponds to $|S|$ calls to *Img*. Therefore, if the test for the sufficient condition fails, the additional complexity is that of one call to *Img*. However, if the test succeeds, the gain can be equivalent to up to $|S| - 1$ calls to *Img*. Recall that, for a sequential circuit with n flip-flops, $|S| = 2^n$. This series of observations leads us to be optimistic about the usefulness of the proposed modification.

7 CONCLUSION

The idea at the basis of this paper is that if, in a given model M, for a given property f on the states of M, we can show that f is valid for all initial states, and that f is preserved by the transitions of M, then f is an invariant of M. This test for validity seems to compare advantageously to conventional symbolic model checking techniques.

After a rapid overview of different formal tools involved in symbolic model checking, we define a sufficient condition for a property to be an invariant of a finite Kripke structure. A proof provides a theoretical justification. We show how to modify the conventional CTL symbolic model checking algorithm to include a test for this condition. This test can be implemented using BDDs and is based in an inductive characterization of invariants.

We are currently working to incorporate these ideas into the verification tool SMV (McMillan 1993) as a heuristic to check a family of properties. Once this implementation is realized, we plan to study the practical usefulness of the heuristic, i.e. that some invariant properties can actually be checked without computing the fixpoint of the conventional algorithm.

REFERENCES

Anderson, R. J., Beame, P., Burns, S., Chan, W., Modugno, F., Notkin, D. & Reese, R. (1996), Model checking large software specifications, *in* '4th Symposium on the Foundations of Software Engineering', ACM/SIGSOFT, pp. 156–166.

Bryant, R. (1986), 'Graph-based algorithm for boolean function manipulation', *IEEE Transactions Computers* C(35), 1035–1044.

Clarke, E. & Emerson, E. A. (1981), Design and synthesis of synchronization skeletons for branching time temporal logic, *in* 'Logics of Programs: Workshop', Vol. 131 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 52–71.

Clarke, E., Emerson, E. & Sistla, A. (1986), 'Automatic verification of finite-state concurrent systems using temporal logic specifications', *ACM Transactions On Programming Languages and Systems* 8(2), 244–263.

Iwashita, H., Nakata, T. & Hirose, F. (1996), CTL model checking based on forward state traversal, *in* 'ICCAD'96', p. 82.

Madre, J.-C. (1990), PRIAM Un outil de vérification formelle des circuits intégrés digitaux, PhD thesis, Ecole nationale supérieure des télécommunications, Paris, France. 90 E 007.

McMillan, K. (1993), *Symbolic Model Checking*, Kluwer Academic Publishers.

Queille, J.-P. & Sifakis, J. (1981), Specification and verification of concurrent systems in CESAR, *in* 'Procs. 5th international symposium on programming', Vol. 137 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 244–263.

Tarski, A. (1955), 'A lattice-theoretical fixpoint theorem and its applications', *Pacific J. Math* pp. 285–309.

8 BIOGRAPHY

David Déharbe got his PhD in Computer Science in 1996 from Université Joseph Fourier, Grenoble, France. He stayed for 2 years at Carnegie Mellon University, School of Computer Science as a Visiting Researcher. Since March 1997, he is substitute professor at Universidade Federal do Rio Grande do Norte, in the Department of Computer Science and Applied Mathematics (DIMAp). His main research interests include formal verification of hardware designs, and semantics of hardware description languages.

Anamaria Martins Moreira got her PhD in Computer Science in 1995 from Institut National Polytechnique de Grenoble, Grenoble, France. She stayed for 18 months at Carnegie Mellon University, School of Computer Science as a Visiting Researcher. Since March 1997, she is working at Universidade Federal do Rio Grande do Norte, in the Department of Computer Science and Applied Mathematics (DIMAp). Her main research interests include formal specification theory and applications.