Hiding Propositional Constants in BDDs

Jan Friso Groote
Utrecht University, Faculty of Philosophy
Heidelberglaan 8, 3584 CS Utrecht, the Netherlands
Tel: +31-30-532761 Fax: +31-30-532816
E-mail: jfg@phil.ruu.nl

1 Introduction

In [6] it is described how correctness of safety control systems of railways (interlockings) can be expressed using (large) propositional formulas. Interlockings are correct if these formulas are tautologies. Given the impressive capabilities to show propositional formulas tautologies ascribed to Binary Decision Diagrams (BDDs) [1], it seems natural to apply them to the formulas that we obtained. Somewhat to our dissatisfaction it appeared that the formulas had to be reduced with 98% for the plain BDD technique to yield results. The propositional formulas consist of 4000 lines, and BDDs could only be constructed of formulas covering up to 70 lines. Even if dynamic variable ordering is applied, formulas up till 70 lines can be handled [4].

In this note we describe a very straightforward and easy to implement extension to the BDD technique, using hiding functions. It turns out that these functions make it possible to check whether the sizable formulas describing the correctness of interlockings are tautologies, which shows their strength. This strength is confirmed in [2, 3] where it has been described how using partitioned transition relations the complexity of BDD based verifications can be significantly reduced. In essence these partitioned transition relations can be viewed as an application of the technique presented in this paper. At the end of the article we describe an experiment with randomised 3-SAT formulas. It is shown that if the number of occurrences of variables in 3-SAT formulas is small hiding functions apply very well. All experiments mentioned above indicate that hiding functions are useful if variables occur rather localised in formulas and/or have a relatively small number of occurrences.

2 Hiding functions $f_H^\wedge$ and $f_H^\vee$

We are considering propositional formulas $\phi, \psi, \ldots$ built from a set of propositional constants $p, q, r, \ldots$ and connectives $\wedge$, $\vee$ and $\neg$. In Bryant [1] it has been described how using Reduced Ordered Binary Decision Diagrams (BDDs) it can be shown that a formula is satisfiable, a contradiction or a tautology. As we do not require the actual construction of BDDs, except in an example, we do not explain this technique but refer to [1].

We define two new functions $f_H^\wedge$ and $f_H^\vee$, where H is a set of propositional constants, as follows.

Definition 2.1.

$$f_H^\wedge(\phi) \stackrel{\text{def}}{=} \bigwedge_{\sigma:H\to\{\mathsf{t},\mathsf{f}\}} \sigma(\phi),$$
$$f_H^\vee(\phi) \stackrel{\text{def}}{=} \bigvee_{\sigma:H\to\{\mathsf{t},\mathsf{f}\}} \sigma(\phi).$$

So, $f_H^\wedge(\phi)$ is the conjunction of all formulas where propositional constants in H are instantiated with both true (t) and false (f). E.g. $f^\wedge_{\{p,q\}}(p \vee q) = \mathsf{f}$ and $f^\vee_{\{p,q\}}(p \vee q) = \mathsf{t}$.

The following important lemma is an immediate consequence of the definitions of tautology and contradiction.

Lemma 2.2.

$$f^\wedge_{\alpha(\phi)}(\phi) = \mathsf{t} \quad \text{iff} \quad \phi \text{ is a tautology},$$
$$f^\vee_{\alpha(\phi)}(\phi) = \mathsf{f} \quad \text{iff} \quad \phi \text{ is a contradiction}.$$

Here $\alpha(\phi)$ is the alphabet of ϕ, i.e. the set of all propositional constants that occur in ϕ.

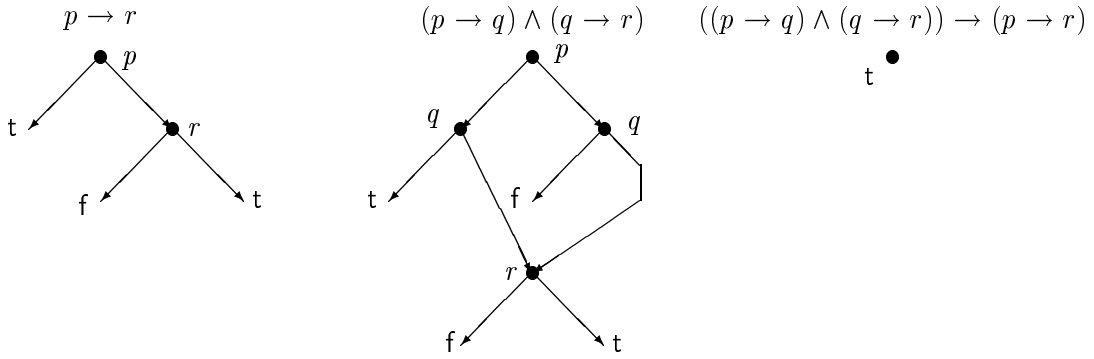
The lemma can in a straightforward way be used to detect that a formula ϕ is a tautology; just calculate $f^\wedge_{\alpha(\phi)}(\phi)$. But this is very inefficient, as it resembles the truth table method. Efficiency is obtained if the following facts are applied to calculate $f^\wedge_{\alpha(\phi)}(\phi)$.

Lemma 2.3.

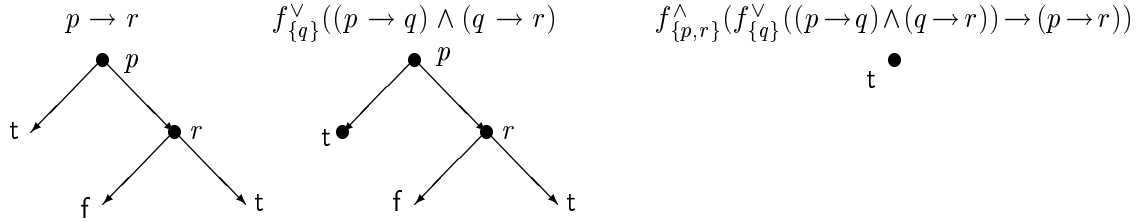
- $f_H^\wedge(p) = \mathsf{f}$ if $p \in H$,
- $f_H^\wedge(p) = p$ if $p \notin H$,
- $f_H^\vee(p) = \mathsf{t}$ if $p \in H$,
- $f_H^\vee(p) = p$ if $p \notin H$,
- $f_H^\wedge(\neg\phi) = \neg f_H^\vee(\phi)$,
- $f_H^\vee(\neg\phi) = \neg f_H^\wedge(\phi)$,
- $f_H^\wedge(\phi \wedge \psi) = f_H^\wedge(\phi) \wedge f_H^\wedge(\psi)$,
- $f_H^\vee(\phi \wedge \psi) = f^\vee_{H\cap\alpha(\phi)\cap\alpha(\psi)}(f^\vee_{(H\cap\alpha(\phi))\setminus\alpha(\psi)}(\phi) \wedge f^\vee_{(H\cap\alpha(\psi))\setminus\alpha(\phi)}(\psi))$,
- $f_H^\wedge(\phi \vee \psi) = f^\wedge_{H\cap\alpha(\phi)\cap\alpha(\psi)}(f^\wedge_{(H\cap\alpha(\phi))\setminus\alpha(\psi)}(\phi) \vee f^\wedge_{(H\cap\alpha(\psi))\setminus\alpha(\phi)}(\psi))$,
- $f_H^\vee(\phi \vee \psi) = f_H^\vee(\phi) \vee f_H^\vee(\psi)$.

By applying these rules as rewrite rules on formulas, the functions $f_H^\wedge$ and $f_H^\vee$ are pushed as far as possible inside the formula. Note that the rewrite rules are actually non-terminating. They should only be applied when part of the set H can be pushed inside the formula. The following example shows that the use of the hiding functions can have a beneficial effect on the size of BDDs.

Example 2.4. We show using BDD techniques that $((p \to q) \wedge (q \to r)) \to (p \to r)$ is a tautology (obviously, $\phi \to \psi \equiv \neg\phi \vee \psi$). Without the function $f_H^\wedge$ some intermediate BDDs are:



Using the function $f_H^\wedge$ we must show that $f^\wedge_{\{p,q,r\}}(((p \to q) \wedge (q \to r)) \to (p \to r))$ is true. Using the simplification rules of Lemma 2.3 this simplifies to $f^\wedge_{\{p,r\}}(f^\vee_{\{q\}}((p \to q) \wedge (q \to r)) \to (p \to r))$. The comparable intermediate BDDs that we now get are:



It is clear that the size of the second BDD has become smaller.

Application of the 'rewrite rules' of Lemma 2.3 and the calculation of $f_H^\wedge$ and $f_H^\vee$ on BDDs in case no rewrite rule is applicable is rather straightforward. Suppose we want to show a formula ϕ a tautology. Annotate each subformula ϕ with its alphabet. This can be done in time $O(|\alpha(\phi)| \cdot |\phi|)$. Having done this, the alphabet of any subformula is available in constant time. Now, recursively apply a function $\mathit{ConstructBDD}(\phi, \alpha(\phi), \wedge)$, which simultaneously pushes the hiding function inside a formula and calculates the BDD representation of ϕ. The auxiliary function $\mathit{Inst}(\mathit{type}, H, b)$ instantiates the proposition letters in H in the BDD b according to Definition 2.1.

function $\mathit{ConstructBDD}(\psi$:*formula*; H:*alphabet*; $\mathit{type} \in \{\wedge, \vee\})$:BDD;

Case $\psi \equiv p$: return $\begin{cases} \textcircled{f} & \text{if } \mathit{type} \equiv \wedge \text{ and } p \in H, \\ \textcircled{t} & \text{if } \mathit{type} \equiv \vee \text{ and } p \in H, \\ \textcircled{p} & \text{otherwise.} \end{cases}$

Case $\psi \equiv \neg\chi$: return $\ominus(\mathit{ConstructBDD}(\chi; H; \begin{cases} \wedge & \text{if } \mathit{type} \equiv \vee \\ \vee & \text{if } \mathit{type} \equiv \wedge \end{cases}))$.

Case $\psi \equiv \chi_1 \wedge \chi_2$ and $\mathit{type} \equiv \wedge$:
return $\mathit{ConstructBDD}(\chi_1, H, \wedge) \mathbin{\textcircled{\wedge}} \mathit{ConstructBDD}(\chi_2, H, \wedge)$.

Case $\psi \equiv \chi_1 \wedge \chi_2$ and $\mathit{type} \equiv \vee$:
return $\mathit{Inst}(\vee, H \cap \alpha(\chi_1) \cap \alpha(\chi_2),$
$\mathit{ConstructBDD}(\chi_1; (H \cap \alpha(\chi_1)) \setminus \alpha(\chi_2); \vee) \mathbin{\textcircled{\wedge}}$
$\mathit{ConstructBDD}(\chi_2; (H \cap \alpha(\chi_2)) \setminus \alpha(\chi_1); \vee))$.

Case $\psi \equiv \chi_1 \vee \chi_2$ and $type \equiv \vee$:
return $ConstructBDD(\chi_1, H, \vee) \circledcirc ConstructBDD(\chi_2, H, \vee)$.
Case $\psi \equiv \chi_1 \vee \chi_2$ and $type \equiv \wedge$:
return $Inst(\wedge, H \cap \alpha(\chi_1) \cap \alpha(\chi_2),$
$ConstructBDD(\chi_1; (H \cap \alpha(\chi_1)) \setminus \alpha(\chi_2); \wedge) \circledcirc$
$ConstructBDD(\chi_2; (H \cap \alpha(\chi_2)) \setminus \alpha(\chi_1); \wedge))$.

function $Inst(type \in \{\wedge, \vee\}; H{:}alphabet; b{:}\mathrm{BDD}){:}\mathrm{BDD}$;
Case $H = \emptyset$: return b.
Case $p \in H$:
$c = Inst(type, H \setminus \{p\}, b)$;
return $\begin{cases} c|_{p=\mathsf{t}} \circledcirc c|_{p=\mathsf{f}} & \text{if } type \equiv \vee, \\ c|_{p=\mathsf{t}} \circledwedge c|_{p=\mathsf{f}} & \text{if } type \equiv \wedge. \end{cases}$

Here, $\circledwedge$, $\circledcirc$ and $\circleddash$ are the application of conjunction, disjunction and negation on BDDs. $\textcircled{t}$, $\textcircled{f}$ and $\textcircled{p}$ are the BDD representations for t, f and p, respectively. The notation $c|_{p=\mathsf{t}}$ and $c|_{p=\mathsf{f}}$ is the restriction of c to the case where p is t or f. Descriptions of implementations of these operators can be found in [1].

The considerable efficiency gain in proving the formulas describing safety of railway safety control systems must be contributed to the relatively localised occurrences of propositional constants[1]. Moreover most propositional constants do not occur very often in the formulas. For both reasons the functions $f_H^\vee$ and $f_H^\wedge$ can be pushed very deeply inside formulas. This in turn means that in the construction of the overall BDD the intermediate BDDs contain a relatively small number of different propositional constants. For the safety control system this were 165, whereas the BDD of the safety system constructed without $f_H^\wedge$ and $f_H^\vee$ could contain up to 3175 different variables.

In order to understand the effects of the hiding functions better a small experiment with randomised 3-SAT formulas has been done. Random formulas in conjunctive normal form with n clauses have been generated. Each clause contains 3 literals that are randomly picked (with replacement) from a set of v proposition letters. Each proposition letter is negated with probability $\frac{1}{2}$. We have tried to show the generated formulas are either satisfiable or contradictory both with and without the hiding functions. In Table 1 it is listed how many BDD nodes are required for different choices of n and v. For each n and v the number shown is that for the first experiment. Repeating the experiments yields approximately the same figures. A '-' means that due to the size of the BDD no result has been obtained. The table clearly shows that hiding functions are more desirable if the number of proposition letters grows.

As also remarked in [2, 3] it depends very much on the structure of a formula how far the hiding functions can be pushed inside. Finding the optimal structure of a formula may greatly improve the performance of hiding functions. This can for instance be achieved by by applying commutativity and associativity of the $\wedge$ and $\vee$. However, it is unknown to us whether an (efficient) algorithm for this exists. Work that has been done on this problem

[1] The general form of these formulas is $(\bigwedge_{i \in I} \phi_i) \rightarrow \psi$ where ϕ_i are (small) facts describing the safety unit and ψ expresses a certain safety property. It is clear that the function $f_H^\wedge$ can only pushed far inside if the formulas ψ and ϕ_i do not have too many variables in common

suggests that this is a general but difficult problem (see [5] for more details).

Acknowledgements. I thank Koen van Eyk and Bas van Vlijmen for their comments and help in carrying out some experiments.

References

[1] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[2] J.R. Burch, E.M. Clarke, and D.E. Long. Representing Circuits More Efficiently in Symbolic Model Checking. In proceedings of the 28^th ACM/IEEE Design Automation Conference, pages 403-407, 1991.

[3] J.R. Burch, E.M. Clarke, and D.E. Long. Symbolic Model Checking with Partitioned Transition Relations. In A. Halaas and P.B. Denyer, Editors, Proceedings of 1991 Intl. Conf. on VLSI, 1991.

[4] K. van Eyk. Personal communication.

[5] J.F. Groote. An optimisation problem for an AC operator. To appear in EATCS Bulletin Number 54.

[6] J.F. Groote, S.F.M. van Vlijmen and J.W.C. Koorn. The safety guaranteeing system at station Hoorn-Kersenboogerd. To appear.

25 clauses	5	10	25	50	100	200	400	variables
without hiding	45	96	5234	28734	-	-	-	
with hiding	68	205	1091	116	13	5	5	
50 clauses	5	10	25	50	100	200	400	variables
without hiding	76	171	6093	498592	-	-	-	
with hiding	74	282	10396	13433	2336	28	11	
100 clauses	5	10	25	50	100	200	400	variables
without hiding	134	219	5513	-	-	-	-	
with hiding	116	343	24052	-	-	37874	394	

Table 1: Required number of BDD nodes for 3-SAT problems