Solving QBF with SMV

Francesco M. Donini
Dipartimento di Elettrotecnica ed Elettronica
Politecnico di Bari
Via Re David 200
Bari, Italia

Paolo Liberatore
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113
Roma, Italia

Fabio Massacci
Dipartimento di Ingegneria Civile e Ambientale
Università di Trento
Via Sommarive, 14
Povo (Trento), Italia

Marco Schaerf
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113
Roma, Italia

Abstract

The possibility of solving the Quantified Boolean Formulae (QBF) problems using the SMV system is a consequence of two well-known theoretical results: the membership of QBF to PSPACE, and the PSPACE-hardness of LTL (and therefore, of SMV). Nevertheless, such results do not imply the existence of a reduction that is also of practical utility. In this paper, we show a reduction from QBF to SMV that is linear (instead of cubic), and uses a constant-size specification.

This new reduction has three applications the previous one has not: first, it allows for solving QBF problems using SMV-like systems, which are now more developed than direct QBF solvers; second, we can use it to verify whether the performance behavior of direct QBF solvers is intrinsic of the problem, or rather an effect of the solving algorithm; third, random hard SMV instances can be easily generated by reduction from QBF hard instances (whose generation method is now established).

1 Introduction

In recent years a number of solvers for Quantified Boolean Formulae (QBFs) have been proposed in the literature [CSGG00, Rin99, EETW00, FMS00]. QBFs are a natural extension of satisfiability problems in classical propositional logic (SAT), and are of great theoretical interest since they allow to precisely characterize all computational complexity classes in the polynomial hierarchy and PSPACE [Sto76]. From a knowledge representation point of view, it is well known that most problems in propositional reasoning, such as planning, nonmonotonic reasoning, abduction, and diagnosis can be reduced to the problem of checking the truth of a quantified boolean formula.

An obviously interesting problem is to find out whether the phenomena of the existence of a steep state transition and of the easy-hard-easy pattern, which are found in SAT problems, also arise in QBF, and if they are similarly related. The initial investigations based on the Evaluate algorithm [CSGG00] have shown that both phenomena exist in QBFs, even though not as evident as in SAT. Further investigations by Gent and Walsh [GW99] and others have confirmed this initial investigation.

In this paper we show a very simple and efficient reduction from the problem of checking the truth of a QBF into model-checking of an LTL (linear temporal logic) or a CTL (computation tree logic) formula over an SMV model. In the following section we briefly present LTL, CTL, and SMV. The reader interested in a more formal and detailed presentation should refer to McMillan's thesis [McM93] and Huth and Ryan's book [HR00] that contain a detailed exposition. Implementing a translator based on this reduction allows for using the SMV system as a QBF solver. A first benefit is that, since the SMV system is widely used, well engineered, and efficient, this method may be more efficient than direct QBF solving.

A second benefit is the effect of SMV being based on a completely different technology than most of the current QBF solvers, which are usually variations of the DLL procedure [DLL62], and are therefore way too similar to prove that some computational properties (such as the easy-hard-easy pattern) are intrinsic of the problem.

A third important benefit of the proposed reduction is for the benchmark generation and system evaluation in model checking. Indeed, currently one is forced to choose between two alternatives:

1. generate computationally challenging benchmarks using the problems constructed from the theoretical results; these problems (model-checking explicit Kripke structures) are not interesting for evaluating actual systems, which take a different input (an SMV specification);
2. use the large number of "real world" problems (written using SMV specifications); unfortunately, there is no way to know whether these problems are really representative of a broader class of problems and really computationally challenging.

We would like to have the best of both approaches: *generation of computationally challenging (and controllable) benchmark with model-checking specifications actually used in practice.* The reduction we propose allows for generating benchmarks from random QBF instances, which can be generated in a way that guarantees their computational properties. Namely, the critical parameters for the generation of hard instances of QBF have been deeply studied [CSGG00].

From a theoretical point of view, this reduction implies that CTL and LTL model checking Kripke specifications written in SMV is PSPACE-hard *even when the specification has constant size* (known reductions employ specifications of polynomial size). However, the benefit of this reduction is more practical than theoretical: besides having a new method for solving QBF instances, we can now generate model checking problems whose hardness can be somehow "controlled". This allows for testing the empirical effectiveness of various model checking optimizations. QBFs offer complete problems for all classes in the polynomial hierarchy [Sto76], and therefore very suitable for the experimental analysis, as they include problems that belong to a wide range of complexity classes [CSGG00, EETW00], a property not easily found with direct experimental analysis. See for instance the difficulties noted by Daniele, Giunchiglia, and Vardi [DGV99] for the experimental comparison of LTL automata generation algorithms.

```
MODULE main
VAR
  request : boolean;
  state   : {ready, busy};
ASSIGN
  init(state) := ready;
  next(state) := case
        state = ready & request = 1 : busy;
                                   1 : {ready, busy};
        esac;

SPEC
  AG(request -> AF state = busy)
```

Figure 1: An example SMV program

We remark that a proof of PSPACE completeness of LTL, implying the existence of a polynomial reduction from QBF to SMV, already exists [SC85]. However, this reduction requires a cubic encoding and requires the usage of the *until* modality, while our reduction is linear and employs the temporal modalities *globally* and *next* only. This is why the new reduction is more suited to a practical use.

2 Preliminaries

In this section we briefly introduce the model-checking system SMV; the following presentation is adapted from the NuSMV manual [CCO$^+$]. Further details on SMV are in McMillan's book [McM93], while NuSMV is described in the paper by Cimatti et al. [CCGR00].

The primary purpose of an SMV specification is to describe the transition relation of a Kripke structure. Any expression in the propositional calculus can be used to describe this relation. This provides a great deal of flexibility, and at the same time a certain danger of inconsistency. For example, the presence of a logical contradiction can result in a deadlock – a state or states with no successor. This can make some specifications vacuously true, and makes the description unimplementable. While the model checking process can be used to check for deadlocks, it is best to avoid the problem when possible by using a restricted description style. The SMV system supports this by providing a parallel-assignment syntax. The semantics of assignment in SMV is similar to that of single assignment data flow language. By checking programs for multiple parallel assignments to the same variable, circular assignments, and type errors, the interpreter insures that a program using only the assignment mechanism is implementable. Consequently, this fragment of the language can be viewed as a description language, or a programming language.

Consider the SMV program in Figure 1. The first part defines the Kripke structure. The space of states of the Kripke structure is determined by the declarations of the state variables (in the above example `request` and `state`). The variable `request` is declared to be of (predefined) type boolean. This means that it can assume the (integer) values 0 and 1. The variable `state` is a scalar variable, which can take the symbolic values `ready` or `busy`.

The following assignment sets the initial value of the variable state to ready. The initial value of request is completely unspecified, i.e. it can be either 0 or 1.

The transition relation of the Kripke structure is expressed by defining the value of variables in the next state (i.e. after each transition), given the value of variables in the current states (i.e. before the transition). The `case` segment sets the next value of the variable `state` to the value `busy` (after the column) if its current value is `ready` and `request` is 1 (i.e. true). Otherwise (the 1 before the column) the next value for `state` can be any in the set {`ready,busy`}.

The variable `request` is not assigned. This means that there are no constraints on its values, and thus it can assume any value, which is thus an unconstrained input to the system.

Specifications can be expressed in CTL (Computation Tree Logic). The NuSMV system also allows for specifications in LTL (Linear Temporal Logic). These logics allow a rich class of temporal properties, including safety, liveness, fairness and deadlock freedom, to be specified in concise a syntax.

The keyword `SPEC` is followed by a CTL formula, that is intended to be checked for truth in the Kripke structure defined above. The intuitive reading of the formula is that every time `request` is true, then in all possible future evolution, eventually `state` must become `busy`.

The two (modal) propositional temporal logics LTL and CTL are used to express temporal properties of the modeled system. LTL is a temporal logic whose underlying model of time is linear. More precisely, every model of LTL is a Kripke structure where all worlds are connected in a (possibly infinite) chain. The syntax of LTL includes three unary modal operators X, F and G and a binary one U. Where the intended meanings are:

- $X\alpha$ means: α is true in the next state;
- $F\alpha$ means: α is true in some future state;
- $G\alpha$ means: α is true in all future states;
- $\alpha U\beta$ means: α is true Until β becomes true.

When a linear model of time is not adequate we can resort to CTL whose semantics is based on branching time. The semantics is given via unrestricted Kripke models. In the contest of CTL a sequence of connected states is called a path. In CTL it is possible to quantify either existentially or universally on the paths.

3 Intuition

In this section we provide some intuitions of the reductions presented in Sections 4.

The truth of the QBF formula F is transformed into a model checking problem for SMV in which the model is made up by a number of (slightly modified) counters, a boolean formula in CNF and a number of modules that non-deterministically set a boolean variable.

In a nutshell, the SMV system works as follows:

- all possible assignments to the universal variables are generated using one binary counter whose digits are the assignments to the universal variables, while the assignments to the existential ones are "guessed" non-deterministically. In each state of the system the satisfaction of the CNF formula is checked. Whenever the CNF formula is not satisfied the counting is halted (no carry is propagated to the next variable).
- The LTL (or CTL) specification simply says that for any execution there exists a state that makes the formula false.

4 From QBF to SMV structures

QBFs extend propositional logic by allowing propositional variables to be quantified over, either existentially or universally. The *evaluation problem* for a QBF is to decide whether a given QBF is true or not.

Let F be the Quantified Boolean Formula over n propositional variables:

$$Qx_1Qx_2\dots Qx_n.(\gamma_1 \wedge \cdots \wedge \gamma_m)$$

where Q is a quantifier (either $\forall$ or $\exists$) and each γ_j is a clause over the variables $\{x_1, \dots, x_n\}$. The sequence of quantifiers $Qx_1Qx_2\dots Qx_n$ is called the prefix of F and will be denoted as P, while un-quantified boolean formula $\gamma_1 \wedge \cdots \wedge \gamma_m$ is called the matrix of F and will be denoted as E. A clause γ_j with k literals is represented as $l_{j_1} \vee \dots \vee l_{j_k}$, where each literal l is either a variable or its negation.

```
MODULE forall(carry-in)
VAR
   value : boolean;
ASSIGN
   init (value) := 0;
   next (value) := value ^ carry-in;
DEFINE
   carry-out := value & carry-in;
```

Figure 2: The `forall` module in SMV syntax

```
MODULE exists(carry-in)
VAR
   value : boolean;
ASSIGN
   init (value) := {0,1};
   next (value) :=
     case
        carry-in : {0,1};
        1        : value;
     esac;
DEFINE carry-out := carry-in;
```

Figure 3: The `exists` module in SMV syntax

In the above formulation we do not pose any restriction on the alternation of the quantifiers. For any given variable x_i it is only necessary to know its position (i) in the prefix and whether it is existentially or universally quantified.

We now briefly present our encoding of QBFs in SMV. As usual $\{0, 1\}$ means the non-deterministic choice between 0 and 1. Each universally quantified variable is coded in a module `forall(carry-in)`, described in Fig. 2, which is a binary digit of a binary counter with carry [HR00, p.184]. The bit is stored in a variable *value*. The presence of a module `forall(x`$_{i+1}$`.carry-out)` in the main module corresponds to a universally quantified variable x_i.

The module `exists(carry-in)` in Fig. 3 is used for existentially quantified variables. Its single boolean variable `value` is non-deterministically set to 0 or 1. Whenever a carry (the parameter `carry-in`) changes the value of the inner variable, its value is non-deterministically chosen again. The module never changes the carry, but simply forwards the carry it receives from the inner variable to the outer variable.

The boolean formula is coded in a purely reactive module `formula` (Fig. 4) that evaluates the truth value of matrix.

To build the whole SMV code we simply need to:

1. incorporate the definition of the modules `forall` and `exists`;

```
MODULE formula(v1,.......,vn)

DEFINE
        c1 := "first clause of F";
        c2 := "second clause of F";
        .
        .
        cm := "last clause of F";
        sat := c1 & c2 & ....... & cm;
```

Figure 4: The `formula` Module in SMV syntax

```
MODULE main

VAR
        xn    : QUANT(clauses.sat);
        xn-1  : QUANT(xn.carry-out);
        xn-2  : QUANT(xn-1.carry-out);
        .
        x2    : QUANT(x3.carry-out);
        x1    : QUANT(x2.carry-out);
        clauses : formula(x1.value,...,xn.value);
SPEC AF (!clauses.sat)
```

Figure 5: The `main` Module in SMV syntax

2. instantiate the module `formula` with the clauses in the matrix;

3. in the `main` module (see Fig. 5):

   (a) declare the variables $x_1, \ldots, x_n$ using the appropriate module (either `forall` or `exists`) and a parameter, which is the carry-out of the next module, but for x_n whose parameter is *sat* of the formula module;

   (b) declare the variable *clauses* with the module `formula`, and all the values of $x_1, \ldots, x_n$ as parameters;

   (c) declare the specification.

The specification is independent of the QBF and can be expressed either in LTL or in CTL. In LTL it is $\mathbf{F}(\neg clauses.sat)$, while in CTL it is $\mathbf{AF}(\neg clauses.sat)$. Either formula is interpreted as: for all execution paths of the system, eventually the matrix will not be satisfied. Then clearly the QBF F is false.

An Example of the Translation

In order to make the translation easier to understand, we show it in action: we convert the simple QBF formula $\forall x_1 \exists x_2 . x_1 \equiv x_2$ into SMV code, where $x_1 \equiv x_2$ is an abbreviation for $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$.

First of all, we have the modules `forall` and `exists`, which are not repeated here because they do not de-

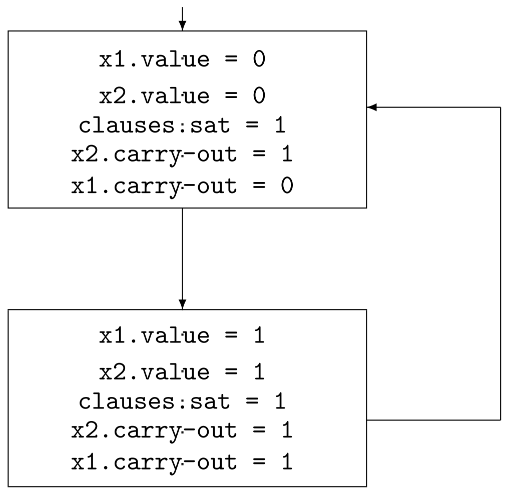

Figure 6: A path that makes the specification false in the translation of $\exists x_1 \forall x_2 . x_1 \equiv x_2$.

pend on the specific formula. The `formula` module is:

```
MODULE formula(x1, x2)
DEFINE
  c1 := x1 | !x2;
  c2 := !x1 | x2;
  sat := c1 & c2;
```

To complete the example, only the `main` module is missing. Here, we specify how the variables are quantified. In particular, each variable is defined to be the result of the appropriate module (`forall` or `exists`), using the carry of the next variable as input (exception made for the last variable, whose module input is the truth value of the formula).

```
MODULE main
VAR
  x2 : exists(clauses.sat);
  x1 : forall(x2.carry-out);
```

Finally, the specification is independent from the formula, and is as follows:

```
SPEC AF (!clauses.sat)
```

In Figure 6 we show an infinite path (in fact a cycle) that makes such a specification false. The initial state encodes the assignment 00 to variables x_1, x_2 of the QBF, while its successor state encodes the assignment 11. Together, 00 and 11 form a proof that $\forall x_1 \exists x_2 . x_1 \equiv x_2$ is a valid QBF.

In order to complete the example, we show how the formula $\exists x_1 \forall x_2 . x_1 \equiv x_2$ is translated (this is almost the same formula as above, but the quantification of variables are swapped). The only point of the SMV code that depends on the quantifiers is the `main` module: now x_1 is existentially quantified while x_2 is universally quantified.

```
MODULE main
VAR
  x2 : forall(clauses.sat);
  x1 : exists(x2.carry-out);
```

The rest of the SMV code is the same as the first example of translation. In Figure 7 the Kripke structure resulting from such translation is depicted. Clearly, for every initial state the specification is now satisfied.

Summarizing, changing the quantification of variables amounts to changing the definition of variables in the `main` module, while changing the matrix leads to a different `formula` module.

5 Correctness of the reduction

To ease notation, we let the symbol 1 stand for $\top$, and 0 for $\bot$. Let $P.E$ be a QBF, with n boolean variables. To determine whether the QBF is valid, one can start by peeling off the outermost quantifier: if it's $\exists x_1$, we choose one of the truth values 1 or 0 and substitute for the newly freed occurrence of x_i; if it's $\forall x_1$, substitute both 1 and 0 for the newly freed occurrences of x_1. In short, while evaluating QBFs we are generating a tree, where existential quantifiers increase the depth, and universal quantifiers force branching. If we always reach a leaf where the assignments of boolean values to variables make the matrix E true, the overall formula F is valid.

If we look at the assignments sequentially, an *assignment* is a sequence of bits; hence, we denote an assignment to variables $x_1, \ldots, x_n$ as a string $b_1 b_2 \cdots b_n \in \{0,1\}^n$, *i.e.,* a sequence of n bits, where b_i is assigned to x_i.

We denote UNIVS(P) the sequence of indexes of the universally quantified variables in the prefix P (in the same order) while EXISTS(P) is the sequence of indexes of the existential quantified variables in P. Given two sequences A and B, the *projection* of B along A, denoted as $\pi_B(A)$, is the subsequence of A obtained by selecting only the elements whose index is in B.

The key of the proof is that we establish a correspondence between the sequence of assignments to the boolean variables generated by the quantifiers of a valid QBF and the states of the SMV model. Given

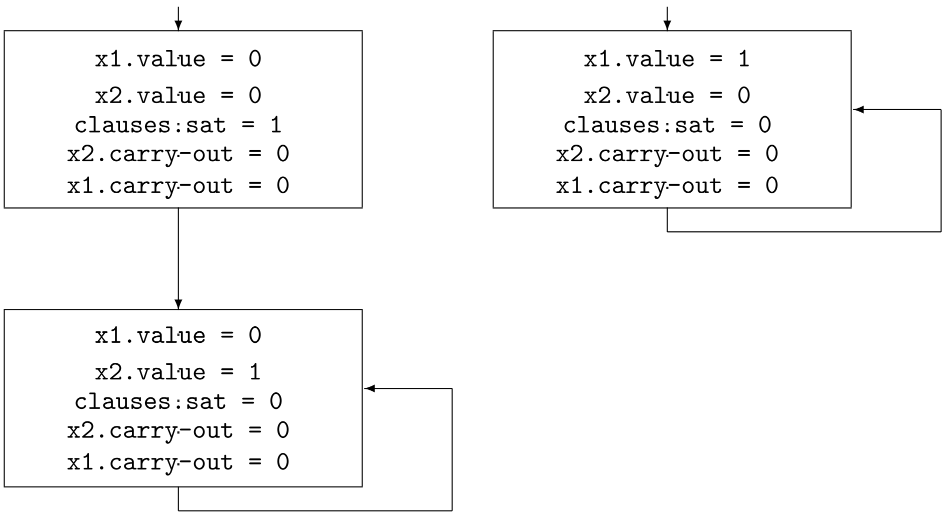

Figure 7: The Kripke structure resulting from the translation of $\forall x_1 \exists x_2 . x_1 \equiv x_2$. The state corresponding to assignment 11 is unreachable from initial states, so it is not represented.

two assignments, we start by defining a relation of successor between them.

Definition 1 (Successor) *Let P be a prefix of a QBF, and let s_1, s_2 be two assignments to variables in P. Then s_2 is a* successor *of s_1 (denoted by $s_1 \rightsquigarrow_P s_2$) if the following conditions hold:*

1. *$\pi_{\text{UNIVS}(P)}(s_2)$ is the successor of $\pi_{\text{UNIVS}(P)}(s_1)$ in the lexicographical order, in $\{0,1\}^{|\text{UNIVS}(P)|}$;*
2. *for any $k \leq n$, if $\pi_{1\cdots k}(\pi_{\text{UNIVS}(P)}(s_1)) = \pi_{1\cdots k}(\pi_{\text{UNIVS}(P)}(s_2))$ then $\pi_{1\cdots k}(\pi_{\text{EXISTS}(P)}(s_1)) = \pi_{1\cdots k}(\pi_{\text{EXISTS}(P)}(s_2))$*

The first condition is equivalent to: the integer numbers defined by taking the value of the universally quantified variables of the sequences must be consecutive. The second condition says that, if the universally quantified variables of two assignments coincide up to an index k, then the existentially quantified variables must coincide up to the same index. In other words, two consecutive assignments $s_1 = a_1 \cdots a_n$ and $s_2 = b_1 \cdots b_n$ can assign a different value to an existentially quantified variable x_i (so, $a_i \neq b_i$) only if there is a universally quantified variable x_j with $j < i$ (*i.e.,* it has a lower index), and $a_j \neq b_j$.

Since the prefix will be clear form the context, in what follows we omit P in $\rightsquigarrow$.

For instance, if $P = \forall x_1 \exists x_2 \forall x_3 \forall x_4 \exists x_5$, then $10010 \rightsquigarrow 10100$. This is because since $\text{UNIVS}(P) = 134$, $\text{EXISTS}(P) = 25$, and $\pi_{\text{UNIVS}(P)}(10010) = 101$, $\pi_{\text{UNIVS}(P)}(10100) = 110$ hence Condition 1 is met. Also, Condition 2 requires that $\pi_{1\cdots 2}(\pi_{\text{UNIVS}(P)}(10010)) = 10 = \pi_{1\cdots 2}(\pi_{\text{UNIVS}(P)}(10100))$ implies that $\pi_{1\cdots 2}(\pi_{\text{EXISTS}(P)}(10010)) = 0 = \pi_{1\cdots 2}(\pi_{\text{EXISTS}(P)}(10100))$, which is indeed true. Instead, $11010 \not\rightsquigarrow 10100$ because Condition 2 is not met.

We now define a sequence of assignments that covers all assignments needed to satisfy a given prefix.

Definition 2 (Consistent sequence) *A sequence of assignments $(s_1, \ldots, s_m)$ is* consistent *with a prefix P if and only if:*

1. *it is made of $m = 2^{|\text{UNIVS}(P)|}$ assignments;*
2. *$\pi_{\text{UNIVS}(P)}(s_1) = 0 \cdots 0$;*
3. *$s_i \rightsquigarrow s_{i+1}$, for every $i = 1, \ldots, n-1$.*

Observe that in a consistent sequence of assignments, the bits assigned to universally quantified variables span from 0 to $2^m - 1$. For the last assignment s_m of the sequence, it must be $\pi_{\text{UNIVS}(P)}(s_m) = 1 \cdots 1$. Not all possible sequences of assignments are consistent with a prefix. For example, let $P = \exists x_1 \forall x_2 \forall x_3$. We have that $\{000, 001, 010, 011\}$ is consistent (we guessed $x_1 = 0$ and tried all possible values of x_2 and x_3) whereas $\{000, 001, 110, 011\}$, and $\{000, 001, 010\}$ are not consistent. Intuitively, once a value is set for an existential variable, all universal variables with higher index must range over all possible truth values in the successive assignments. In practice, if we have $\exists x_i \forall x_{i+1} \forall x_{i+2} \ldots$ and we set, say, $x_i = 1$ we must use a binary counter to test all possible values of $x_{i+1}x_{i+2}$.

The intuition behind consistent sequences of assignments is that they can represent both a proof of validity of a QBF F, and a path falsifying the specification of the translation of F in SMV. The rest of this section is then devoted to these two issues: on one side, prove that a QBF is valid iff there exists a consistent sequence of assignments all satisfying the matrix; on the other side, prove that the translation of F in SMV has a false specification iff there exists a consistent sequence of assignments all verifying `clauses.sat`.

If the values in an assignment are less than the number of variables n, we call it *partial assignment.* We implicitly assume that the truth values of a partial assignment are used for the first variables, while the last variables are left unassigned.

Definition 3 (Partial Evaluation) *Let E be a boolean formula over variables $x_1, \ldots, x_n$, and let $s = a_1 \cdots a_k$, with $k \leq n$ be a partial assignment. The* partial evaluation *of E according to s, denoted by $E|s$, is the boolean formula obtained by replacing variables $x_1, \ldots, x_k$ with $a_1, \ldots, a_k$ respectively.*

For sake of completeness, we can now recall the inductive definition of validity of a QBF w.r.t. a partial assignment.

Definition 4 *[Validity of a QBF] Let $P.E$ be a QBF, where $P = Q_{k+1}x_{k+1} \cdots Q_n x_n$, and let $s = a_1 \cdots a_k$ be a partial assignment. Formula $P.E|s$ is valid iff one of the following conditions hold:*

1. *If $k = n$ (i.e., P is empty and s is total), then F is valid iff $E|s \equiv 1$;*
2. *If $P = \forall x_{k+1} P'$ (i.e., the first element of P is a universally quantified variable), then F is valid iff both $P'.E|s0$ and $P'.E|s1$ are valid;*
3. *If $P = \exists x_i P'$ (i.e., the first element of P is an existentially quantified variable), then the formula is true iff either $P'.E|s0$ or $P'.E|s1$ is valid.*

Obviously, F is valid iff it is valid w.r.t. the empty assignment.

Theorem 1 *A QBF $P.E$ is valid if and only if there exists a sequence of assignments that is consistent with P, and such that every assignment satisfies E.*

Proof. $\Rightarrow$ Let $P.E$ be valid: then Definition 4 is met. Based on the definition, we recursively construct a sequence of assignments. Let $S(\mathcal{P}, \mathcal{E}, s)$ a function from a (generic) prefix $\mathcal{P}$, a (generic) matrix $\mathcal{E}$, and a partial assignment s, to a sequence of assignments. Let $\circ$ denote the concatenation of sequences of assignments. Then $S(\mathcal{P}, \mathcal{E}, s)$is defined as follows:

1. if $\mathcal{P}$ is empty, then $S(\mathcal{P}, \mathcal{E}, s) = s$;
2. if $\mathcal{P} = \exists x.\mathcal{P}'$, and v is the truth value making $\mathcal{P}'.(\mathcal{E}|(s\cdot v))$ valid, then $S(\mathcal{P}, \mathcal{E}, s) = S(\mathcal{P}', \mathcal{E}, s\cdot v)$;
3. if $\mathcal{P} = \forall x.\mathcal{P}'$ then $S(\mathcal{P}, \mathcal{E}, s) = S(\mathcal{P}', \mathcal{E}, s \cdot 0) \circ S(\mathcal{P}', \mathcal{E}, s \cdot 1)$

Note that the value v in Point 2. exists by Definition 4. We now prove that computing $S(P, E, \emptyset)$ one obtains a sequence of assignments that is consistent with P, and such that each of its assignments satisfies E.

The second claim follows from an invariant: from the first call, and at each inner recursive call of $S(\mathcal{P}, \mathcal{E}, s)$, the formula $\mathcal{P}.(\mathcal{E}|s)$ is valid, hence it is valid also for leaf calls which yield the total assignments.

Regarding the first claim, we prove the following properties of S: for each call of $S(\mathcal{P}, \mathcal{E}, s)$, the sequence of assignments it generates contains $2^{|\text{UNIVS}(\mathcal{P})|}$ assignments, starts with an assignment st for which $\pi_{\text{UNIVS}(\mathcal{P})}(t) = 0 \cdots 0$, and ends with an assignment of the form sq for which $\pi_{\text{UNIVS}(\mathcal{P})}(q) = 1 \cdots 1$. All of these properties are proved by induction on the number $|\text{UNIVS}(\mathcal{P})|$. Then, Points 1. and 2. of Definition 2 immediately follow, and Point 3. is proved again by induction on $|\text{UNIVS}(\mathcal{P})|$.

$\Leftarrow$ Let $s_1, \ldots, s_m$ be the consistent sequence of assignments. Let $b_1^i \cdots b_k^i$ denote the first k assignments of s_i. The proof is by induction, on the following statement: if $Q_{k+1}x_{k+1} \cdots Q_n x_n.(E|b_1^i \cdots b_k^i)$ are all valid, for every $i = 1, \ldots, m$, then $Q_k x_k \cdots Q_n x_n.(E|b_1^i \cdots b_{k-1}^i$ are all valid. The base case is for $k = n$, which is true because every assignment satisfies E. We work backwards on k, achieving the proof for $k = 0$. If $Q_k = \exists$, then the inductive claim is trivially true, since b_k^i is either 0 or 1. If $Q_k = \forall$, suppose $b_k^i = 0$. Let $u = |\text{UNIVS}(Q_{k+1}x_{k+1} \cdots Q_n x_n)|$. Then since the sequence is consistent with P, the assignment s_j, with $j = i + 2^u$, is such that $b_1^i \cdots b_{k-1}^i = b_1^j \cdots b_{k-1}^j$, and $b_k^j = 1$. Since also $Q_{k+1}x_{k+1} \cdots Q_n x_n.(E|b_1^j \cdots b_k^j)$ is valid, the claim follows. $\Box$

We now prove the correspondence between paths falsifying the specification in the Kripke structure of the translation of a QBF, and consistent sequences of assignments all satisfying E.

Theorem 2 *Let $F = P.E$ be a QBF, and let $\mathcal{S}_F$ be the Kripke structure defined by the translation of F. Then* *the specification is false iff there exists a sequence of assignments, consistent with P and such that every assignment satisfies E.*

Proof. $\Rightarrow$ If the specification is false, then there exists an infinite path satisfying $G(clauses.sat)$, that is, in every state the assignment to variables satisfies E. We now prove that the first $m = 2^{|\text{UNIVS}(P)|}$ states of the path define a consistent sequence of assignments.

We go through all points of Definition 2. Point 1. is true by hypothesis.

The initial state of the SMV model assigns 0 to every universal variable, hence the assignment of the initial state satisfies Point 2. of Definition 2.

For Point 3, let s_i and s_{i+1} be two states in the path. To avoid multiple notations, let also s_i and s_{i+1} denote the truth assignments obtained by taking the value of variable v in each module. If E is true, then clauses.sat is 1, hence the carry-in of the module corresponding to the most internal variable is 1. Modules for existential variables just pass the carry over to the next module, so modules for universal variables act as a standard binary counter (see [HR00]). This satisfies Condition 1. of Definition 1. Moreover, if $\pi_{1,\dots,k}(\pi_{\text{UNIVS}(P)}(s_1)) = \pi_{1,\dots,k}(\pi_{\text{UNIVS}(P)}(s_2))$, then carry-in of the modules 1–k must be 0. But then, the existential modules just replicate in s_{i+1} the value of v that was set in s_i, hence $\pi_{1,\dots,k}(\pi_{\text{EXISTS}(P)}(s_1)) = \pi_{1,\dots,k}(\pi_{\text{EXISTS}(P)}(s_2))$. This satisfies Condition 2. of Definition 1. In conclusion, $s_i \rightsquigarrow s_{i+1}$.

$\Leftarrow$ Follows easily from the definition of a consistent sequence of assignments. The infinite path goes from s_1 through s_m and then back to s_1.

$\square$

Simply combining Theorem2 and Theorem1, we can now state our main theorem.

Theorem 3 *Given a QBF F, let P_F be the corresponding* SMV *Program. Then, the specification in P_F is false if and only if F is true.*

As for the size of the encoding, it is clear that modules `forall` and `exists` have constant size. The module `formula` has the same size of E, while `main` has size linear in the number of variables of F. The size of the specification is also constant (it does not depend on the QBF F). Moreover, it is easy to show that this encoding can be computed using additional logarithmic workspace.

Obviously this doesn't mean that the final expanded model would have size $O(poly(n, m))$. However, we are only interested in mapping a QBF into an SMV-specification of polynomial size. This result implies that model-checking in SMV is PSPACE-hard even for specifications of constant size.

6 Experimental Results

In this section we present the most important experimental results on randomly generated QBF instances. All instances are obtained using the 2QBF-5CNF FCL2 model [CSGG00], we now recall.

In the early QBF work by Cadoli et al. [CGS97], random kQBF instances were generated according to the *Fixed Clause Length* (FCL) model [SML96]. The FCL model for QBFs has the following parameters:

- the number k of distinct sets of propositional variables in formula $Q_1X_1 \cdots \exists X_k \,.\, E(X_1, \dots, X_k)$,
- the cardinalities $|X_1|, \dots, |X_k|$,
- the number m of clauses in E,
- the number h of literals per clause.

In this model each formula is generated so that every clause contains h literals. If V is the set $X_1 \cup X_2 \cup \cdots \cup X_k$, then a clause is produced by randomly choosing h distinct variables in V and negating each one with probability 0.5. The FCL model for QBF directly extends the FCL model for SAT.

The model also constraints the m clauses to be different to each other, and that none of them includes both a literal and its complement. Moreover, to avoid generating trivially false kQBF instances, a clause cannot contain universally quantified variables only.

Gent and Walsh [GW99] noted that with the FCL model the probability of generating instances containing two clauses such that in each clause all variables except one are universally quantified and the remainig variabla also appears in the other clause with the opposite sign increases very quickly when m increases. Such instances are therefore false. This is why the FCL2 model has been introduced. In this model, if a clause contains a literal l and another clause contains $\neg l$, then one of the clauses is removed and replaced. FCL2 is similar to two models (named A and B) investigated by Gent and Walsh [GW99], the difference being that the selection criterion of FCL2 is more precise: for example, in model A a clause with less than two existential variables (which does not necessarily

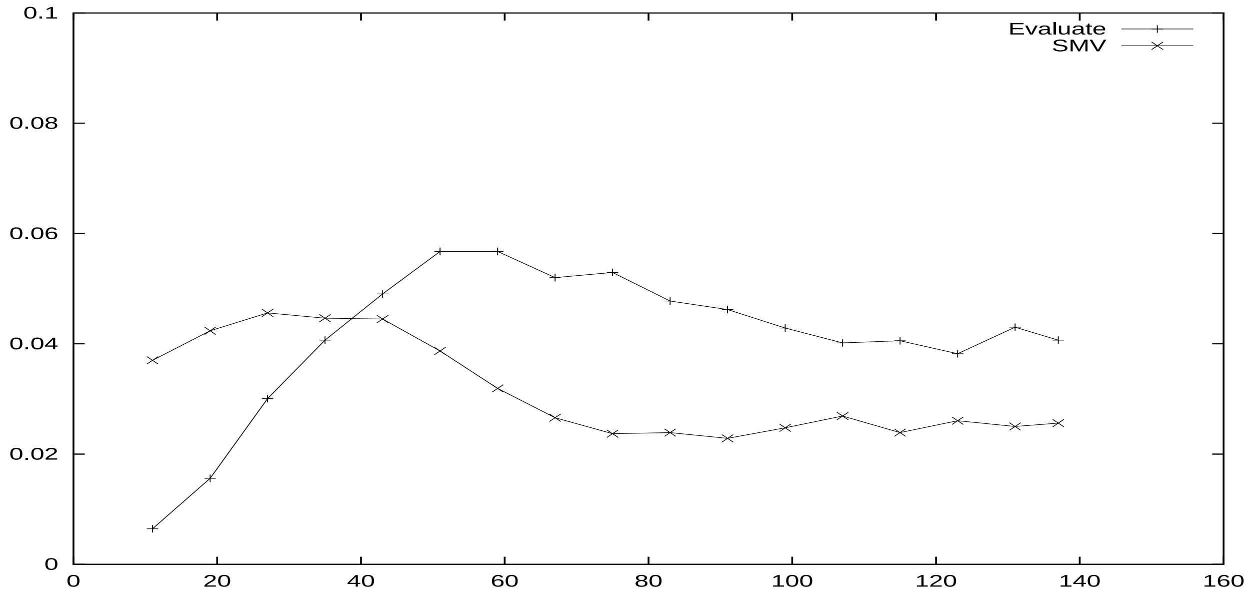

Figure 8: Evaluate vs. SMV (2QBF-5CNF, 14 variables, FCL2 model)

make the formula false) is disallowed. Extensive testing has been done on FCL2 [CSGG00].

Figure 8 reports the result of a comparison between direct QBF solving with Evaluate and solving by reduction to SMV. This test has been done with 14 variables (of which 7 are universally quantified and 7 are existentially quantified). Two facts are quite evident: first, the performance of the two algorithms are comparable, showing that SMV is a viable alternative to direct QBF solving; second, hard QBF instances remain hard after the translation to SMV, which implies that hard random SMV instances can be obtained from hard QBF instances by reduction. Finally, while the shape of the curves are more or less the same, their precise values are quite different. Namely, the peak generated by SMV is moved to the left, and the increase before the peak is less steep. All these observations confirm the usefulness of the proposed translation in practice. Significant differences in the relative efficiency of BDD-based and Davis-Putnam -based solvers for SAT problems have also been reported by Giunchiglia et. al. in [GNT01] and Vardi et al. in [CDSMA$^+$00].

Figure 9 shows the comparison with formulae of ten variables. Some more experiments have been run using NuSMV instead of SMV. The results are reported in Figure 10, 11, 12. Experiments are still running at the time of this writing; other ones suggested by an anonymous reviewer are planned.

7 Conclusions

In this paper we have shown that the evaluation of a QBF can be reduced to model-checking of a simple (constant) formula over an SMV model. The theoretical implication of this result is that CTL and LTL model checking are PSPACE-hard in the size of the structure, when the structure is written using a practical language like the SMV input language. More importantly, the reduction can be exploited to generate challenging benchmarks for model-checking systems. This is useful, as recent results in the QBF literature [CSGG00, EETW00] show that it is possible to generate very hard instances for QBF in a controlled way.

A significant result of the performed test is that the overall QBF algorithm composed of the translation and the solution by SMV has computational properties comparable with those of direct QBF solvers. It is also interesting how some properties, like the position of the peak, that were believed to be intrinsic of the QBF problem, do not appear, suggesting that they are related to the specific solving algorithm.

Acknowledgments

The authors thank Enrico Giunchiglia and Alessandro Cimatti for interesting discussions on the topic of this paper. They also thank the anonimous reviewers for their useful suggestions. Work partially supported by the Italian Ministry for University and Research

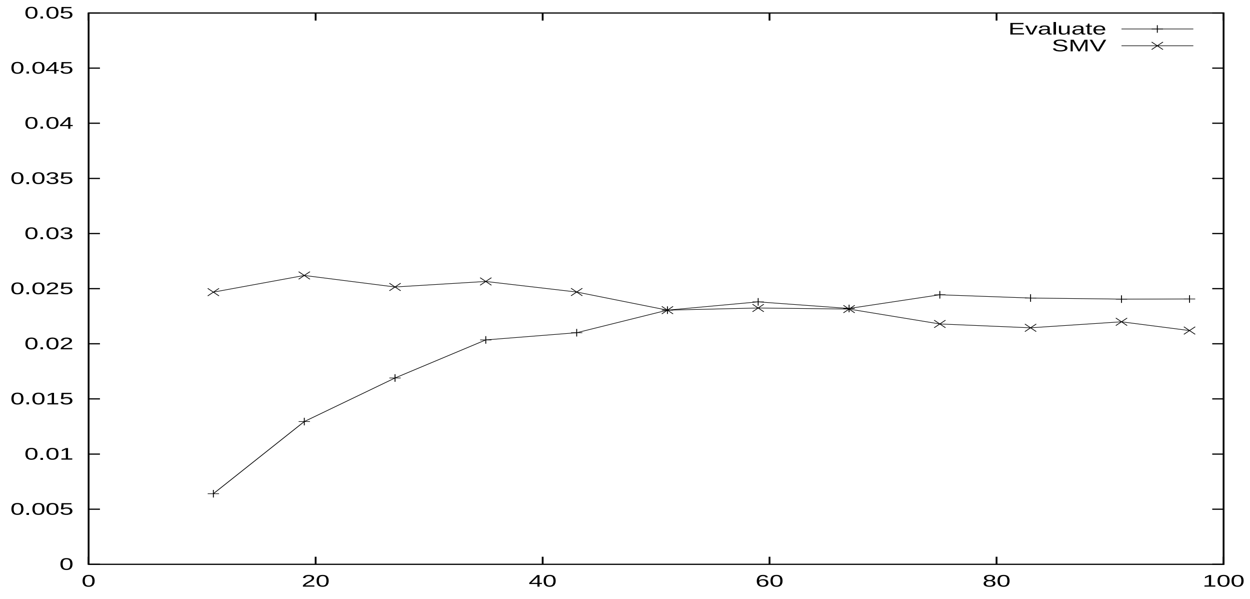

Figure 9: Evaluate vs. SMV (2QBF-5CNF, 10 variables, FCL2 model)

(project MOSES) and Italian National Research Council (project LAICO).

References

[CCGR00] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new symbolic model checker. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(4):410–425, 2000.

[CCO+] Roberto Cavada, Alessandro Cimatti, Emanuele Olivetti, Marco Pistore, and Marco Roveri. Nusmv 2.0 user manual. Available on the Web at http://nusmv.irst.itc.it/NuSMV/userman/.

[CDSMA+00] C Coarfa, D Demopoulos, A. San Miguel Aguirre, D. Subramanian, and M Vardi. Random 3-sat: The plot thickens. In *Proc. of CP 2000*, pages 143–159, 2000.

[CGS97] Marco Cadoli, Andrea Giovanardi, and Marco Schaerf. Experimental analysis of the computational cost of evaluating quantified boolean formulae. In *Proc. of AI*IA'97*, number 1321 in LNAI, pages 207–218. Springer-Verlag, 1997.

[CSGG00] Marco Cadoli, Marco Schaerf, Andrea Giovanardi, and Massimo Giovanardi. An algorithm to evaluate quantified boolean formulae and its experimental evaluation. In Ian Gent, Hans van Maaren, and Toby Walsh, editors, *SAT2000 - Highlights of Satisfiability Research in the Year 2000*, pages 485–521. IOS Press, 2000. Will appear also on *Journal of Automated Reasoning*, special issue on *SAT2000*.

[DGV99] M. Daniele, F. Giunchiglia, and M. Vardi. Improved automata generation for linear temporal logic. In *Proc. of CAV'99*, volume 1633 of *LNCS*, pages 249–260. Springer-Verlag, 1999.

[DLL62] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem proving. *Comm. of the ACM*, 5(7):394–397, 1962.

[EETW00] U. Egly, T. Eiter, H. Tompits, and S. Woltran. Solving advanced reasoning tasks using quantified boolean formulas. In *Proc. of AAAI 2000*, pages 417–422, 2000.

[FMS00] R. Feldmann, B. Monien, and S. Schamberger. A distributed algorithm to evaluate quantified boolean formulae. In *Proc. of AAAI 2000*, pages 285–290, 2000.

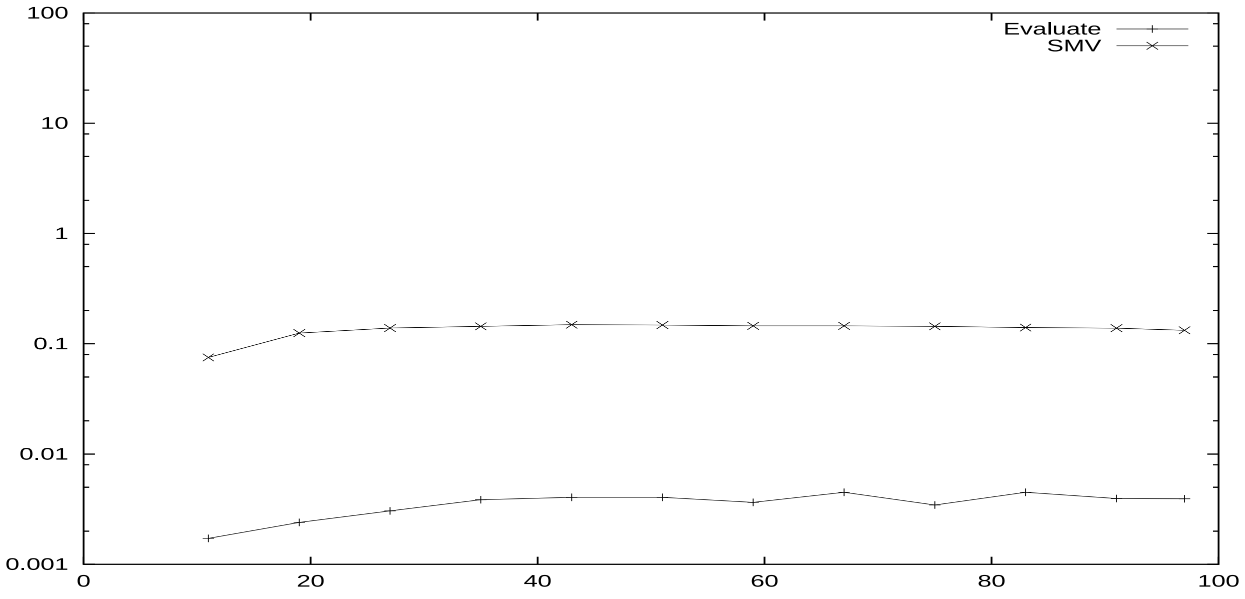

Figure 10: Evaluate vs. NuSMV (2QBF-5CNF, 10 variables, FCL2 model)

[GNT01] E. Giunchiglia, M. Narizzano, and A. Tacchella. An analysis of backjumping and trivial truth in quantified boolean formulas satisfiability. In *Proceedings of the Italian Conference on Artificial Intelligence*, pages 25–28, 2001.

[GW99] I. P. Gent and T. Walsh. Beyond NP: the QSAT phase transition. In *Proc. of AAAI'99*, 1999.

[HR00] M. Huth and M. Ryan. *Logic in Computer Science. Modelling and reasoning about systems.* Cambridge University Press, 2000.

[McM93] Kenneth L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publisher, 1993.

[Rin99] J. Rintanen. Improvements to the evaluation of quantified boolean formulae. In *Proc. of IJCAI'99*, pages 1192–1197, July-August 1999.

[SC85] A. Prasad Sistla and E. M. Clarke. The complexity of propositional linear temporal logic. *J. of the ACM*, 32(3):733–749, 1985.

[SML96] B. Selman, D. Mitchell, and H. Levesque. Generating Hard Satisfiability Problems. *Artificial Intelligence*, 81:17–29, 1996.

[Sto76] L. J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comp. Sci.*, 3:1–22, 1976.

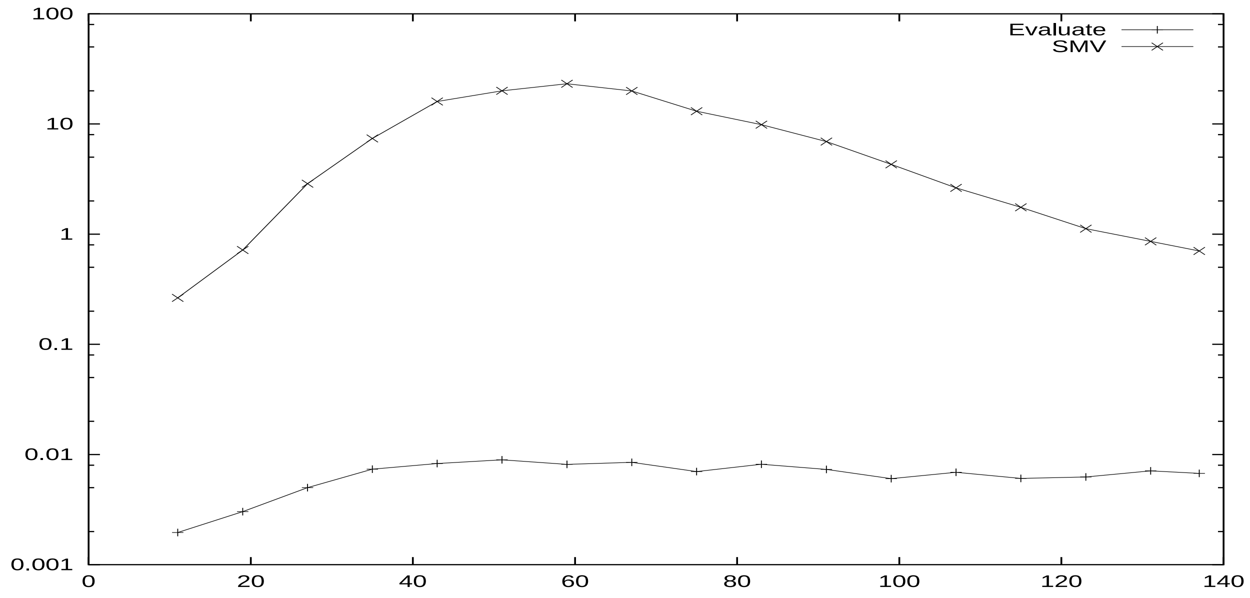

Figure 11: Evaluate vs. NuSMV (2QBF-5CNF, 14 variables, FCL2 model)

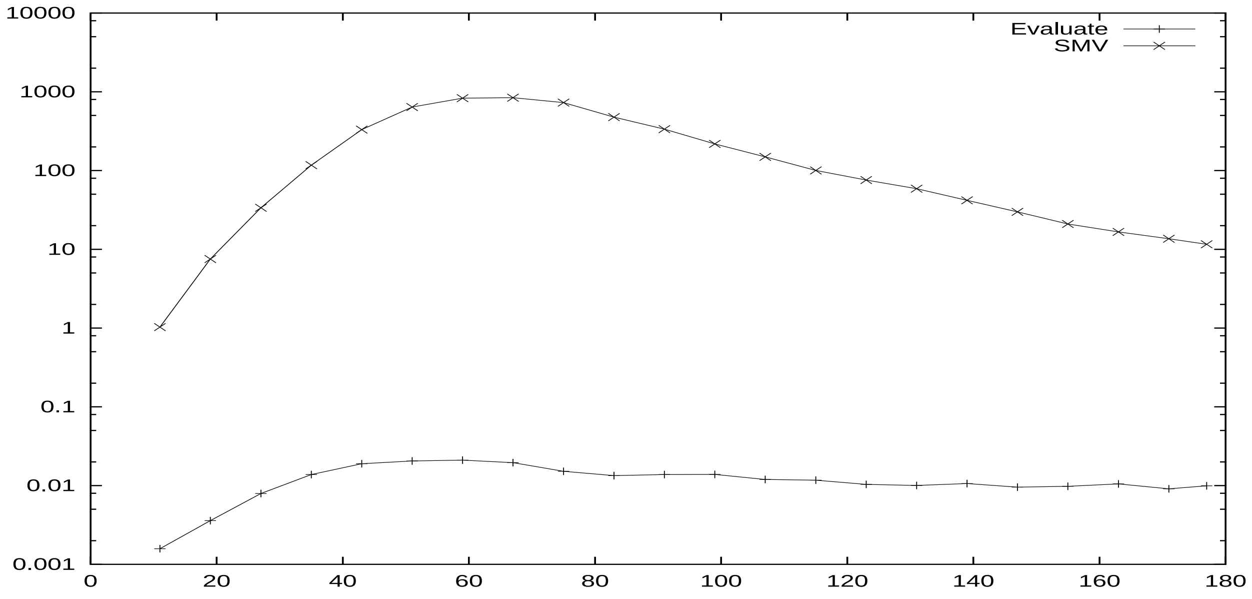

Figure 12: Evaluate vs. NuSMV (2QBF-5CNF, 18 variables, FCL2 model)