June 2007

Symbolically Quantified Propositional Logic
and its application to the Evaluation of QBFs

Marco Benedetti MARCO.BENEDETTI@UNIV-ORLEANS.FR
Laboratoire d'Informatique Fondamentale d'Orléans
University of Orléans
BP6759 - 45067, Orléans (France)

Abstract

We introduce Symbolically Quantified Propositional Logic (SQPL), a formalism in which constraints are represented through a combination of clauses and binary decision diagrams (BDDs). The language is endowed with a formal semantics, some inference rules, and a decision procedure. The introduction of SQPL is motivated by the existence of a simple translation, called *symbolic skolemization*, which maps quantified booleans formulas (QBFs) onto equivalent SQPL formulas. Experiments show that solving QBFs by translating them into SQPL improves the state of the art in QBF reasoning, an important achievement in that many interesting real-world reasoning tasks can be expressed as QBFs.

1. Introduction

We are interested in deciding the validity of *quantified boolean formulas* (QBFs) such as:

$$f = \forall a \exists b \forall c \exists d.\ (\neg a \vee c \vee d) \wedge (\neg b \vee \neg d) \wedge (a \vee b \vee \neg d) \wedge (\neg a \vee b) \tag{1}$$

In this sample formula, the variables (or propositions) a, b, c, d can be assigned to one of the truth values *true* or *false*. The *prefix* "$\forall a \exists b \forall c \exists d$", read left-to-right, instructs us to consider the whole formula as true if and only if for both truth values of a it is possible to assign a truth value to b such that for both truth values of c it is possible to assign a truth value to d such that the *matrix* "$(\neg a \vee c \vee d) \wedge (\neg b \vee \neg d) \wedge (a \vee b \vee \neg d) \wedge (\neg a \vee b)$" is *satisfied* in the propositional sense, i.e. at least one literal in each of the four clauses is true.

As unpretentious and artificial as it might seem at first, the evaluation of QBFs matters in fact to a number of real-world reasoning tasks. Many algorithms have thus been proposed to decide their validity, most of which are basically variants of decision procedures for the close and well-known propositional satisfiability problem (SAT).

In this paper, we introduce an alternative approach which makes direct use of *skolemization*. Such method, inspired to provers for first-order languages, has not been practiced in QBF reasoning so far. Substantial issues arise indeed in handling function symbols effectively from within a logic framework which is function-free by design.

Our technique captures the meaning of skolemized QBF instances within a purpose-built formalism for knowledge representation and reasoning, called *Symbolically Quantified Propositional Logic*. In such logic, constraints are expressed by an original combination of clause-based and BDD-based representations, and reasoning is realized by the joint action of BDD manipulations and inferences over conjunctions of clauses.

The new method is applied to evaluate a large set of challenging QBFs coming from applications. Substantial improvements over the state of the art are reported.

Background and motivation. Let us introduce QBF by comparison and contrast with the well-known satisfiability problem. SAT instances specify a conjunction of constraints—such as the matrix of (1)—but no prefix is given. All the variables are assumed to be existentially quantified, so an instance is satisfiable if and only if some assignment which satisfies all the clauses exists. The SAT problem is central to computer science and complexity theory, as it is the prototypical NP-complete problem (Cook, 1971). In recent years, it has transcended its theoretical role by proving itself capable of key contributions to automating real-world reasoning tasks. Successful examples of usage as a generic problem-solving framework include computer-aided design of integrated circuits (Kim, Whittemore, Silva, & Sakallah, 2000; Larrabee, 1992), Planning (Kautz & Selman, 1992), Model Checking for dynamic systems (Biere, Cimatti, Clarke, Fujita, & Zhu, 1999), Scheduling (Crawford & Baker, 1994), Operations Research, and Cryptography (Massacci & Marraro, 2000), to name a few. Aside from specialized approaches, general architectures have been developed to restate instances of any NP problem as SAT instances (Cadoli & Schaerf, 2005).

For many problems, an encoding into SAT followed by the application of state-of-the-art SAT solvers proves frequently more effective than direct solution methods. This witnesses to the progress obtained in the theory and implementation of propositional reasoners. The best present-day solvers—such as (Moskewicz, Madigan, Zhao, Zhang, & Malik, 2001; Een & Sorensson, 2003)—are based on the "DLL" procedure, which is a backtrack search in the space of partial assignments originally presented in (Davis, Logemann, & Loveland, 1962). Modern solvers incorporate many improvements over such basic scheme (see Section 8).

One step above SAT along the expressivity scale of classical logics we encounter QBF. Quantifiers are made explicit and the possibility is granted to alternate arbitrarily many universal/existential quantification *scopes*. Deciding the validity of unrestricted QBFs is a PSPACE-complete problem, while we span the entire *polynomial hierarchy* by bounding the number of alternations in the prefix (Stockmeyer & Meyer, 1973).

QBF has been attracting increasing interest in recent years, as natural formulations of many real-world problems inherently involve an alternation of quantifiers. Equivalent SAT encodings for such cases can always be compiled, for both languages are ultimately propositional. Yet, an exponential blowup necessarily follows (unless NP=PSPACE). A foundational example of "compressed" encoding is given in the very proof of PSPACE completeness for QBF, in e.g. (Papadimitriou, 1994), where an "iterative squaring" formulation captures in linear space the accepting computations of a turing machine that works with polynomially bounded space. Surprisingly, such formulations are not only of theoretical interest. For example, formal verification problems arise whose SAT encoding exceeds the storage capability of present machines, while the equivalent alternating QBF formulation is extremely compact (Jussila & Biere, 2006; Benedetti & Mangassarian, 2007).

This work is essentially motivated by the existence of interesting applications for QBF. So, let us make a short detour to account for the variety and relevance of such applications. One classical example is (Rintanen, 1999a), where it is shown how to use QBF to encode conditional planning problems; in (Egly, Eiter, Tompits, & Woltran, 2000) the encoding

of several knowledge-representation tasks (such as autoepistemic, default logic, disjunctive logic programming, and circumscription problems) is studied; the use of QBF for checking the equivalence of partial implementations is considered in (Scholl & Becker, 2001); in (Ayari & Basin, 2000) a similar technique is applied to check whether the structural and behavioral descriptions of protocols agree with each other; in (Mneimneh & Sakallah, 2003) a formulation of the vertex eccentricity problem is presented, while (Katz, Hanna, & Dershowitz, 2005; Dershowitz, Hanna, & Katz, 2005; Jussila & Biere, 2006) show how QBF could improve over SAT in a broad set of model checking and formal verification tasks; in (Bryant, Lahiri, & Seshia, 2003) QBF solvers are applied to verify pipeline processors; (Gopalakrishnan, Yang, & Sivaraj, 2004) verifies shared memory multiprocessor executions against memory consistency models; (Cook, Kroening, & Sharygina, 2004, 2005) mentions QBF as an ideal target language for encoding certain properties in software verification; (?) encodes the evaluation of nested counterfactuals over propositional knowledge bases; (Ling, Singh, & Brown, 2005) describes how to formalize the FPGA synthesis problem; (Gambin, Lasota, & Rutkowski, 2006) suggests to use QBFs to characterize stationary states in a Petri-net model of a biological gene regulatory network; (Oetsch, Seidl, Tompits, & Woltran, 2006) uses QBF to check the equivalence of answer-set programs; in (Staber & Bloem, 2007) fault correction in sequential circuits is considered, leveraging QBF certificates (Benedetti, 2005b) for the first time.

How do we decide QBF instances? Most recent work has gone into adapting techniques that were originally developed for SAT solvers. A seminal contribution in this direction is (Cadoli, Giovanardi, & Schaerf, 1998), where the DLL method is generalized to universal quantification. Subsequent works have adapted or generalized all the techniques used in modern SAT solvers (see Section 6). Also, quantified resolution (Kleine-Buning, Karpinski, & Flogel, 1995) in the spirit of (Davis & Putnam, 1960; Dechter & Rish, 1994) has been used to decide QBFs with unexpected success (Biere, 2004). SAT algorithms based on binary decision diagrams (BDDs) apply to QBF as well (see Section 6).

Despite all these contributions, the situation seems open to major improvements. After experimenting with QBF encodings, most authors acknowledge that present QBF solvers are weak, and they foster the study of alternative decision procedures.

Our contribution. We present a novel decision procedure for QBF, which is inspired to first-order provers rather than to SAT solvers. The key idea is to leverage skolemization to replace the quest for a strategy which satisfies the matrix with the problem of associating consistent interpretations to the skolem terms. For example, we replace the validity problem for (1) with the satisfiability problem for the purely universal formula

$$\forall a \forall c.\ (\neg a \vee c \vee d(a,c)) \wedge (\neg b(a) \vee \neg d(a,c)) \wedge (a \vee b(a) \vee \neg d(a,c)) \wedge (\neg a \vee b(a))$$

By demanding function symbols, skolemization brings us beyond the reach of propositional reasoning. This is perhaps the single most important reason why such technique—a standard and ubiquitous tool in provers for first-order and other expressive logics—has never had application in QBF. The obstacle could in principle be overcome by just resorting to first-order provers. Yet, QBFs from applications may contain several quantifier alternations over millions of variables and clauses. The algorithms and data-structures of FOL provers are not intended for providing efficient solutions to huge but essentially boolean problems.

Our solution to these issues revolve around the introduction of SQPL, a purpose-built formalism designed to reason on skolemized QBFs without leaving the realm of efficient boolean-logic techniques. SQPL formulas are composed of two interdependent sets—a set of clauses and a set of BDDs—connected through a mapping from the former into the latter. The set of variables mentioned in the clauses (the "existential" variables) is disjoint from the set of decision variables in the BDDs (the "universal" variables). As a result, SQPL formulas are best visualized as bi-dimensional structures, in which several overlapping diagrams grow vertically in the universal dimension, and are rooted at clauses which lines up across the horizontal existential axis. Examples are in Figure 1 and 2.

SQPL is presented as an autonomous formalism, endowed with custom notions of model, satisfiability, inference. It is then equipped with a full-fledged decision procedure. Finally, a link between QBFs and SQPL is established in the form of a translation called "symbolic skolemization". Such translation takes as input a QBF f and yields an equivalent SQPL formula $SymbSk(f)$, which is satisfiable if and only if f evaluates to true. For example, the symbolic skolemization of (1) is the object depicted in Figure 1.

This machinery is applied to the task of deciding QBFs arising from applications: We symbolically skolemize instances from a large test set, and then we check the satisfiability of the resulting SQPL formulas. Experimental evidences suggest that this method substantially improves over classical SAT-inspired decision procedures.

The structure of this paper. A first example of symbolic skolemization is informally developed in Section 3, after some preliminaries have been discussed in Section 2. The formal characterization of SQPL is developed in Section 4, while Section 5 defines inferences and other manipulations over SQPL formulas. Section 6 is devoted to present a specific SQPL decision procedure. Experimental results over a wide benchmark set from applications are presented in Section 7. In Section 8 the related literature is reviewed and commented. Section 9 closes the paper presenting directions for future work.

2. Preliminaries and notation

Indexes and scenarios. We use roman letters for propositional (or boolean) variables over the boolean space $\mathbb{B} = \{0, 1\}$. The elements of $\mathbb{B}^n, n \in \mathbb{N}^+$ are noted by greek letters and are called *indexes*. Subsets of $\mathbb{B}^n$ are called *scenarios* and are noted using calligraphic fonts. For example: $\Psi = \langle \psi_1, \ldots, \psi_n \rangle \in \mathcal{I} \subseteq \mathbb{B}^n$. The complement of $\mathcal{I} \subseteq \mathbb{B}^n$ is written $\overline{\mathcal{I}}$. Bit lists and the star symbol are used to compactly note scenarios. For example, $\{100, *1*\}$ is the scenario in $\mathbb{B}^3$ which contains $\langle 1, 0, 0 \rangle$ and $\langle \phi, 1, \psi \rangle$ for any $\phi, \psi \in \mathbb{B}$.

Given $\Psi = \langle \psi_1, \ldots, \psi_n \rangle \in \mathbb{B}^n$ and $m \leq n$, we pose $\Psi|_m \doteq \langle \psi_1, \ldots, \psi_m \rangle \in \mathbb{B}^m$. This notation is extended to a scenario $\mathcal{I} \subseteq \mathbb{B}^n$ as $\mathcal{I}|_m \doteq \{\Psi|_m : \Psi \in \mathcal{I}\}$. Two indexes $\Phi \in \mathbb{B}^n$ and $\Psi \in \mathbb{B}^m$, where $m \leq n$, are *compatible* if $\Phi|_m = \Psi$. Given $\mathcal{I} \subseteq \mathbb{B}^n$ and $\mathcal{J} \subseteq \mathbb{B}^m$ we use the natural definition of $\mathcal{I} \cap \mathcal{J}$ if $n = m$. If $n > m$ we pose $\mathcal{I} \cap \mathcal{J} = \{\Psi \in \mathbb{B}^n : \Psi \in \mathcal{I}, \Psi|_m \in \mathcal{J}\}$.

Quantified Boolean Formulas. We consider *closed* QBFs in *prenex conjunctive normal form*[1], noted $f = Q_1 v_1 Q_2 v_2 \ldots Q_n v_n.\ \widetilde{f}(v_1, \ldots, v_n)$, where $Q_i \in \{\forall, \exists\}$. The sequence

1. This choice causes no loss of generality and is adopted by most encodings of real-world problems. However, it might be responsible for an increase in proof complexity.

"$Q_1 v_1 Q_2 v_2 \dots Q_n v_n$" is called prefix, while the matrix $\widetilde{f}(v_1, \dots, v_n)$ is a propositional formula in conjunctive normal form (CNF) mentioning the variables $v_1, \dots, v_n$. Given a QBF f, we always use $\widetilde{f}$ to note its matrix. The variable v_i in the prefix is said to be existentially (universally) quantified if $Q_i = \exists$ ($Q_i = \forall$). The set of all the existentially (universally) quantified variables in a QBF f is noted $var_\exists(f)$ ($var_\forall(f)$). An existential (universal) *scope* is any maximal subsequence of existentially (universally) quantified variables in the prefix.

The variable v_i in the prefix *dominates* the variable v_j if $j > i$ and v_i, v_j belong to different scopes. The set of universal variables dominating the existential variable e in a formula f is noted $var_\forall(f, e)$. The *universal depth* of an existential variable e is $\delta(e) \doteq |var_\forall(f, e)|$. For a clause Γ with existential variables $var_\exists(\Gamma)$ we pose $\delta(\Gamma) \doteq max_{v \in var_\exists(\Gamma)} \delta(v)$.

We use the following notation to build literals out of variables: Given a variable v and a parameter $\varphi \in \mathbb{B}$, the expression $\varphi \oplus v$ means v when $\varphi = 0$, and $\neg v$ when $\varphi = 1$. Given a literal $l = \varphi \oplus v$, the variable associated with l is $var(l) = v$, and its *phase* is φ. Double negation is disallowed: $\neg\neg v$ is always rewritten as v.

Whenever a set-oriented notation for clauses or matrixes simplifies the exposition we adopt it. For example, a clause $C = l_1 \vee l_2 \vee l_3$ may be treated as the set of literals $\{l_1, l_2, l_3\}$, and similarly we may write $C \in \widetilde{f}$ to mean that the clause C appears in the matrix $\widetilde{f}$.

An *assignment* A to the variables V is any (possibly partial) function $A : V \mapsto \mathbb{B}$, and it can be described by the set of literals $S_A = \{v : A(v) = 1\} \cup \{\neg v : A(v) = 0\}$. By $S * \widetilde{f}$ we denote the result of assigning the set of literals S in the CNF $\widetilde{f}$. The result is a CNF obtained by first removing from $\widetilde{f}$ all the clauses containing some literal in S, and then deleting all the occurrences of literals whose complement is in S.

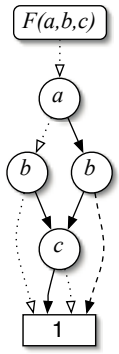
A clause containing no literal is called *empty clause* and is noted $\Box$, while a CNF with no clause is noted $\emptyset$. Formulas with an empty matrix or some empty clause are *trivial*.

A QBF with empty matrix is true, while QBFs with empty clauses are false. A non-trivial formula $f = Q_1 v_1 Q_2 v_2 \dots Q_n v_n.\ \widetilde{f}$ with $Q_1 = \forall$ is true if both $f_1 = Q_2 v_2 \dots Q_n v_n.\ (\{v_1\} * \widetilde{f})$ and $f_2 = Q_2 v_2 \dots Q_n v_n.\ (\{\neg v_1\} * \widetilde{f})$ are true. If $Q_1 = \exists$ then f is true if f_1 is true or f_2 is true. A more articulated semantics which explicitly introduces the notion of *quantified model* (or strategy) is described in e.g. (Büning & Zhao, 2004; Benedetti, 2005b).

Forall-reduction is an equivalence-preserving transformation for QBFs in CNF. A clause Γ is forall-reduced by deleting from it all the universal literals that dominate no variable in $var_\exists(\Gamma)$. We only consider formulas in which all the clauses are forall-reduced.

Binary Decision Diagrams. We make us of *Shared Reduced Ordered Binary Decision Diagrams* (Bryant, 1986) with *complemented arcs* (ROBDDs, or just BDDs henceforth).

A BDD $\mathcal{E}$ is a directed acyclic graph with one root and one sink node (labeled by "1"). It represents some total function $F_\mathcal{E}(u_1, u_2, \dots, u_n)$ from $\mathbb{B}^n$ to $\mathbb{B}$. Internal nodes of the diagram are called *decision nodes* and are labeled by one of the *decision variables* $U = \{u_1, u_2, \dots, u_n\}$. A decision node has invariably two children, one attached to the outcoming *then-arc*, the other to the *else-arc*. The *else-arc* may be *complemented*. The root is labeled by $F_\mathcal{E}$ and has one (possibly complemented) outgoing arc. A unique path from the root to the sink is identified by assigning a boolean value to each variable in U. The *then-arc* is chosen for variables assigned to 1, the *else-arc* is followed otherwise. The function $F_\mathcal{E}$ evaluates to 1 on $\langle \psi_1, \psi_2, \dots, \psi_n \rangle \in \mathbb{B}^n$ iff an even number of complemented arcs occur along the path defined by the decisions $u_1 = \psi_1, \dots, u_n = \psi_n$.



As an example, let us consider the BDD aside, where solid arrows denote then-arcs, while dashed (dotted) arcs are used for regular (complemented) else-arcs. It represents a boolean function $F(a, b, c)$ of the three boolean variables a, b and c. We have, for example, $F(0, 1, 1) = 1$ and $F(1, 1, 1) = 0$. The BDDs we utilize are *ordered* in that decision variables are encountered in the same order along all the paths. This property guarantees *canonicity*, i.e. each function is associated with a single canonic BDD. We say that a BDD $\mathcal{E}$ with n decision variables *recognizes* a scenario in $\mathbb{B}^n$, namely the *on-set* of $F_{\mathcal{E}}$. Our sample BDD recognizes the scenario $\{\langle 0, 1, 1\rangle, \langle 1, 1, 0\rangle\} \subseteq \mathbb{B}^3$.

BDDs are also *reduced*, in the sense that no isomorphic subgraphs are allowed. Furthermore, the version with complemented arcs associates the same subgraph to S and $\overline{S}$ (for every set S), the latter being referenced by a complemented arc. Complementation and reduction cut space requirements while preserving canonicity.

We represent collections of BDDs as a single, multi-rooted directed graph called *forest*. Canonicity spans over the whole set of nodes, thus allowing the sharing of structural information among different diagrams. The *support set* of a BDD $\mathcal{I}$ in a forest, noted $supp(\mathcal{I})$, is the subset of decision variables mentioned in the subgraph related to $\mathcal{I}$.

The BDD way of coding information is *symbolic* in that it avoids the explicit enumeration of sets' elements in favor of a more abstract, diagram-based way of computing characteristic functions. Such representations may be exponentially more succinct than explicit ones (Wegener, 2000), and all the operations on the sets/functions they represent can be performed by manipulating decision diagrams without "decompressing" their meaning (Bryant, 1986).

Quantification. To deal with quantification and prefixes we employ *tree-shaped dominance functions*. Given two sets A an B, a tree-shaped dominance of B over A is any function $d : A \mapsto 2^B$ such that $(A/\sim, \prec)$ is a tree, where the equivalence relation $\sim$ is defined as $a \sim b$ iff $d(a) = d(b)$, and $\prec$ is a partial order relation over the quotient set $A/\sim$ defined as $[a] \prec [b]$ iff $d([a]) \subset d([b])$, with $d([a]) = d(a)$. We say that a subset $A' \subseteq A$ is *linear* w.r.t. a dominance function d of B over A if for any two $a, b \in A'$ it is $d(a) \subseteq d(b)$ or $d(b) \subseteq d(a)$, i.e. $\prec$ is a total order over $A'/\sim$, i.e. $(A'/\sim, \prec)$ has only one branch.

3. Symbolic Skolemization and SQPL: The Intuition

We describe informally a 4-step translation which turns the QBF (1) into the "equivalent" SQPL formula in Figure 1. This technique will be formalized in Section 4.

1. Translation of f into a satisfiability equivalent FOL instance $f^I = Sk(f)$ with no existential quantifier;
2. Translation of f^I into an equivalent SAT instance $f^{II} = Prop(f^I)$;
3. Rewrite of f^{II} in a factorized form $f^{III} = Fact(f^{II})$;
4. Introduction of a BDD-based representation to obtain the final $f^{IV} = Symb(f^{III})$.

Step 1. In order to apply the familiar FOL skolemization, we slightly rephrase the problem by casting the QBF into a FOL syntax. Skolemization leverages the existence of two distinct semantics levels, namely the level of *predicates* and the level of *terms*. Skolem terms replace

other terms (existential variables) as arguments of predicates. QBF lacks this two-level semantics. To join the gap we introduce boolean FOL languages, in which two restrictions are enforced: (i) the domain of interpretation is the boolean space $\mathbb{B}$, and (ii) a unary predicate $\mathbf{p}$ is defined, always interpreted as $\mathbf{p}(1) = \mathbf{t}$, $\mathbf{p}(0) = \mathbf{f}$. This allows us to rewrite every QBF as a boolean FOL formula. For example, if f is the QBF (1) we obtain

$$\begin{array}{ll} \forall a \exists b \forall c \exists d. & (\neg \mathbf{p}(a) \vee \mathbf{p}(c) \vee \mathbf{p}(d)) \wedge (\neg \mathbf{p}(b) \vee \neg \mathbf{p}(d)) \wedge \\ & \wedge\ (\mathbf{p}(a) \vee \mathbf{p}(b) \vee \neg \mathbf{p}(d)) \wedge (\neg \mathbf{p}(a) \vee \mathbf{p}(b)) \end{array} \tag{2}$$

which is equivalent[2] to (1). Then, existential quantifiers are eliminated through *outer skolemization*, i.e. we substitute to each existential variable e a skolem term s^e having as arguments the universal variables $var_\forall(f, e)$. We obtain a purely universal formula:

$$\begin{array}{l} f^I = Sk(f) = \forall a \forall c.\ \ (\neg \mathbf{p}(a) \vee \mathbf{p}(c) \vee \mathbf{p}(s^d(a,c)))\ \ \wedge\ \ (\neg \mathbf{p}(s^b(a)) \vee \neg \mathbf{p}(s^d(a,c))) \wedge \\ \qquad\qquad \wedge\ (\mathbf{p}(a) \vee \mathbf{p}(s^b(a)) \vee \neg \mathbf{p}(s^d(a,c)))\ \ \wedge\ \ (\neg \mathbf{p}(a) \vee \mathbf{p}(s^b(a))) \end{array} \tag{3}$$

From a FOL perspective existential quantifiers disappear, logical equivalence is lost, satisfiability is preserved. We can bring into the open what skolemization does from a second-order perspective: Each existential quantification over e is replaced by an outermost second-order quantification over s^e, i.e. over the existence of a proper interpretation for such Skolem term. With second-order quantification in square parentheses, we write:

$$\forall a \exists b \forall c \exists d.\ \widetilde{f}(a,b,c,d) \quad \Longleftrightarrow \quad [\exists s^b(\cdot) \exists s^d(\cdot,\cdot)]\ \forall a \forall c.\ \widetilde{f}(a,b,c,d)[b/s^b(a), d/s^d(a,c)] \tag{4}$$

The condition posed by the second-order quantifi cations in (4) is what we aim to capture.

Step 2. We restate the satisfiability of (3) as the satisfiability of a suited PL formula. To this end, we *flatten* the interpretation of predicates and terms onto one single propositional space, and we express constraints over such space to capture the second order quantifications in (4), i.e. the existence of proper interpretations for the skolem terms.

Let I be an interpretation for the skolem symbols in $\widetilde{f}$. The interpretation function e^I associated by I to $s^e(u_1, u_2, \ldots, u_m)$, $m = \delta(e)$, is a function from $\mathbb{B}^m$ to $\mathbb{B}$. As such, it can be completely specified by 2^m boolean parameters denoting the truth value of the function on each point of its domain. We can thus represent interpretation functions as CNF formulas, which are associated to terms and clauses by a parametric *propositional skolemization* function $Prop_I(\cdot)$. Let us consider, for example, the terms $s^b(a)$ and $s^d(a,c)$ in (3). Each interpretation of s^b is a function $b^I : \mathbb{B} \mapsto \mathbb{B}$, completely specified by two parameters $b^I(0) = b_0 \in \mathbb{B}$ and $b^I(1) = b_1 \in \mathbb{B}$. We pose:

$$Prop_I(s^b(a)) = (a \vee b_0) \wedge (\neg a \vee b_1)$$

Similarily, we introduce four boolean parameters d_{00}, d_{01}, d_{10}, and d_{11} (where $d_{ij} = d^I(i,j)$ for every $\langle i, j \rangle \in \mathbb{B}^2$) to spell out a CNF version of the interpretation $d^I : \mathbb{B}^2 \mapsto \mathbb{B}$ of s^d:

$$Prop_I(s^d(a,c)) = (a \vee c \vee d_{00}) \wedge (a \vee \neg c \vee d_{01}) \wedge (\neg a \vee c \vee d_{10}) \wedge (\neg a \vee \neg c \vee d_{11})$$

2. This notion of equivalence is understood equivalently as satisfiability or logical equivalence. Indeed, boolean FOL formulas obtained from QBFs mention no free variable and no function symbol, and their domain of interpretation and predicate interpretation is fixed, hence they are valid iff the are satisfiable.

The propositional skolemization of the auxiliary predicate $\mathbf{p}$ is responsible for merging the interpretation of predicates and terms. Formally, this is obtained by posing $Prop_I(\mathbf{p}(t)) = Prop_I(t)$, for each term t. For negated atoms we pose:

$$Prop_I(\neg\mathbf{p}(s^d(a,c))) = (a \vee c \vee \overline{d_{00}}) \wedge (a \vee \neg c \vee \overline{d_{01}}) \wedge (\neg a \vee c \vee \overline{d_{10}}) \wedge (\neg a \vee \neg c \vee \overline{d_{11}})$$

Next comes the propositional skolemization of clauses, which we define as the disjunction of the skolemizations of their literals. For example, from the third clause of (3) we obtain

$$\begin{array}{ccccc} \mathbf{p}(a) & \vee & \mathbf{p}(s^b(a)) & \vee & \neg\mathbf{p}(s^d(a,c)) \\ \overbrace{a} & \vee & \overbrace{[(a \vee b_0) \wedge (\neg a \vee b_1)]} & \vee & \overbrace{[(a \vee c \vee \overline{d_{00}}) \wedge (a \vee \neg c \vee \overline{d_{01}}) \wedge (\neg a \vee c \vee \overline{d_{10}}) \wedge (\neg a \vee \neg c \vee \overline{d_{11}})]} \end{array}$$

The result is a disjunction of conjunctions of disjunctions. To turn it into a CNF, we distribute the disjunctions appearing as principal connectives. An upper bound on the number of clauses that might spring from a similar process is given by the product of the cardinalities of each literal interpretation. However, most clauses are necessarily satisfied by complementary literals, and can be safely removed. For example, after distributing the disjunction between the second and third literal interpretations in our example we obtain

$$a \vee [(a \vee b_0 \vee c \vee \overline{d_{00}}) \wedge (a \vee b_0 \vee \neg c \vee \overline{d_{01}}) \wedge (\neg a \vee b_1 \vee c \vee \overline{d_{10}}) \wedge (\neg a \vee b_1 \vee \neg c \vee \overline{d_{11}})] \quad (5)$$

Then, we distribute the remaining disjunction, and only two clauses survive:

$$Prop_I(\mathbf{p}(a) \vee \mathbf{p}(s^b(a)) \vee \neg\mathbf{p}(s^d(a,c))) = (a \vee b_0 \vee c \vee \overline{d_{00}}) \wedge (a \vee b_0 \vee \neg c \vee \overline{d_{01}}) \quad (6)$$

The last part is the definition of a propositional skolemization for a conjunction of clauses, which we define as the conjunction of the interpretation of each clause. We obtain

$$\begin{array}{rcl} Prop_I(\widetilde{f^I}) & = & (\neg a \vee c \vee d_{01}) \wedge (\neg a \vee \overline{b_0} \vee \neg c \vee d_{00}) \wedge (\neg a \vee \overline{b_0} \vee c \vee d_{01}) \wedge (a \vee \overline{b_1} \vee \neg c \vee d_{10}) \wedge \\ & & \wedge (a \vee \overline{b_1} \vee c \vee d_{11}) \wedge (a \vee b_1 \vee \neg c \vee \overline{d_{10}}) \wedge (a \vee b_1 \vee c \vee \overline{d_{11}}) \wedge (\neg a \vee b_0) \end{array} \quad (7)$$

In order for (3) to be satisfiable, some interpretation I (i.e. specific boolean values for the six parameters b_0, b_1, d_{00}, d_{01}, d_{10}, and d_{11}) that renders (7) a tautology must exist. A conjunction of clauses is tautologic only if it is empty, so what we look for is an assignment to the six parameters that satisfies (7) regardless of the values of the variables a and c.

To capture formally this search problem, we consider the parameter I as an additional argument to $Prop$. By reusing the parameter names as names for fresh variables, we write

$$\begin{array}{rcl} Prop(I, \widetilde{f^I}) & = & (\neg a \vee c \vee d_{01}) \wedge (\neg a \vee \neg b_0 \vee \neg c \vee d_{00}) \wedge (\neg a \vee \neg b_0 \vee c \vee d_{01}) \wedge (a \vee \neg b_1 \vee \neg c \vee d_{10}) \wedge \\ & & \wedge (a \vee \neg b_1 \vee c \vee d_{11}) \wedge (a \vee b_1 \vee \neg c \vee \neg d_{10}) \wedge (a \vee b_1 \vee c \vee \neg d_{11}) \wedge (\neg a \vee b_0) \end{array} \quad (8)$$

The existence of an interpretation I requested by the second-order quantifications in (4) can thus be interpreted at the propositional level of (8) by quantifying on the existence of suited values for b_0, b_1, d_{00}, d_{01}, d_{10}, and d_{11}. To do this we extend $Prop$ to prefixes:

$$Prop([\exists b(\cdot)\exists d(\cdot,\cdots)]\forall a \forall c) \;=\; \exists b_0 \exists b_1 \exists d_{00} \exists d_{01} \exists d_{10} \exists d_{11} \forall a \forall c$$

Then, by applying *Prop* to both the prefix and the matrix of (4) we obtain:

$$\exists b_0 \exists b_1 \exists d_{00} \exists d_{01} \exists d_{10} \exists d_{11} \forall a \forall c \quad \begin{array}{l} (\neg a \vee c \vee d_{01}) \wedge (\neg a \vee \neg b_0 \vee \neg c \vee d_{00}) \wedge (\neg a \vee \neg b_0 \vee c \vee d_{01}) \wedge \\ \wedge\ (a \vee \neg b_1 \vee \neg c \vee d_{10}) \wedge (a \vee \neg b_1 \vee c \vee d_{11}) \wedge \\ \wedge\ (a \vee b_1 \vee \neg c \vee \neg d_{10}) \wedge (a \vee b_1 \vee c \vee \neg d_{11}) \wedge (\neg a \vee b_0) \end{array} \tag{9}$$

By forall-reducing (9) we complete the propositional skolemization of (1):

$$f^{II} = Prop(f^I) = \exists b_0 \exists b_1 \exists d_{00} \exists d_{01} \exists d_{10} \exists d_{11}.\ \begin{array}{l} (d_{01}) \wedge (\neg b_0 \vee \neg d_{00}) \wedge (\neg b_0 \vee \neg d_{01}) \wedge \\ \wedge\ (\neg b_1 \vee \neg d_{10}) \wedge (\neg b_1 \vee \neg d_{11}) \wedge (b_1 \vee \neg d_{10}) \wedge (b_1 \vee \neg d_{11}) \wedge (b_0) \end{array} \tag{10}$$

This purely existential QBF may be seen as a SAT instance, which is satisfiable if (10) is true, and unsatisfiable if (10) is false. By just dropping the quantifiers we write

$$(d_{01}) \wedge (\neg b_0 \vee \neg d_{00}) \wedge (\neg b_0 \vee \neg d_{01}) \wedge (\neg b_1 \vee \neg d_{10}) \wedge (\neg b_1 \vee \neg d_{11}) \wedge (b_1 \vee \neg d_{10}) \wedge (b_1 \vee \neg d_{11}) \wedge (b_0) \tag{11}$$

To summarize, Step 2 enforces the following property: *If (and only if) we find a model for the SAT instance (11) we are entitled to conclude that (3) is satisfiable, i.e. that (1) is true.*

In our example, it is easy to check that (11) is unsatisfiable, hence the QBF (1) is false. By comparing (10-11) with the original formula (3) we observe that:

- (10-11) encode the existence of interpretations for the terms introduced during skolemization that are consistent with the clausal constraints in (3). As we talk explicitly about the value of such interpretations on each point of their domains, the size of (10-11) gets in general exponentially larger than the size of (3). In particular, each FOL clause Γ generates $2^{\delta(\Gamma)-|var_\forall(\Gamma)|}$ propositional clauses;
- The satisfiability of a universal FOL formula is turned into the satisfiability of a SAT instance in CNF, written over fresh variables, i.e. none of the variables in (3) remains in (10-11), and at the same time a collection of new variables is introduced;
- If (11) is unsatisfiable, so is (3). If (11) is satisfiable, not only we are ensured that a proper interpretation for the functions s^b and s^d do exist to satisfy (3), but we have explicitily *computed* such an interpretation. Every model of (11) gives us the desired truth value of acceptable skolem functions over each point of their domains.

Step 3. Formulas of overwhelming[3] size may spring from Steps 1-2. This step and the following head for an exponential compression of such formulas, achieved through the usage of BDDs. The final representation will be not larger than the originating QBF, though it will basically differ from that QBF. Decision diagrams make their appearance during Step 4. As a preliminary transformation, we introduce here a *factorized* notation.

The names of the variables in (10-11) belong to a structured namespace, in so as to each proposition we associate two pieces of information: the name of a skolem term, and an index representing a point in its definition domain. This structured namespace interacts with the distribution of clauses in propositional skolemizations in a predictable way:

3. We examine the issue of which size is to be considered *practically* overwhelming in Section 6.

- All the clauses $Prop(\Gamma)$ originating from a QBF clause Γ mentions all and only the skolem function related to $var_\exists(\Gamma)$;
- Clauses in $Prop(\Gamma)$ differ from one another only for the indexes of the parameters, and such indexes are completely determined by $var_\forall(\Gamma)$.

In our example, the association between original QBF clauses and sets of clauses in the propositional skolemization is as follows:

$$\overbrace{(\neg a \vee c \vee d)}^{(d_{01})} \wedge \overbrace{(\neg b \vee \neg d)}^{(\neg b_0 \vee \neg d_{00}) \wedge (\neg b_0 \vee \neg d_{01}) \wedge (\neg b_1 \vee \neg d_{10}) \wedge (\neg b_1 \vee \neg d_{11})} \wedge \overbrace{(a \vee b \vee \neg d)}^{(b_1 \vee \neg d_{10}) \wedge (b_1 \vee \neg d_{11})} \wedge \overbrace{(\neg a \vee b)}^{(b_0)}$$

As we expect, sets of clauses under the same brace share skolem function names, but differ as to indexes. At the same time, indexes in the same clause are mutually consistent. These two features allow us to factorize the shared information, and to write only four *factorized clauses*. We adopt the following notation:

$$\underbrace{(d_{01})}_{[d]_{\{01\}}} \wedge \underbrace{(\neg b_0 \vee \neg d_{00}) \wedge (\neg b_0 \vee \neg d_{01}) \wedge (\neg b_1 \vee \neg d_{10}) \wedge (\neg b_1 \vee \neg d_{11})}_{[\neg b, \neg d]_{\{00,01,10,11\}}} \wedge \underbrace{(b_1 \vee \neg d_{10}) \wedge (b_1 \vee \neg d_{11})}_{[b, \neg d]_{\{10,11\}}} \wedge \underbrace{(b_0)}_{[b]_{\{0\}}}$$

The four clauses in the resulting *factorized matrix* are of the form $\Gamma_\mathcal{I}$, where $\Gamma = [e_1, \ldots, e_n]$ is a list of (signed) names of skolem functions (or, in terms of the originating QBF, a list of existential literals), and $\mathcal{I} \subseteq \mathbb{B}^{\delta(\Gamma)}$ is the scenario associated to $\Gamma_\mathcal{I}$.

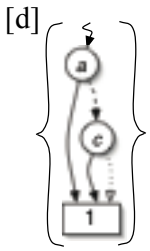
Factorization has to admit an inverse *expansion* operation through which we recover the original non-factorized form. For expansion to be unambiguously defined, we need to know how to "cut" and redistribute factorized indexes among literals, and to this end it suffices to be aware of the arity of each skolem function. Such information is exposed in the prefix of (10), so we retain it. And, for uniformity, the prefix is itself factorized.

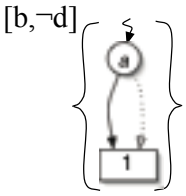
For example, the sequence of quantifications $\exists d_{00} \exists d_{01} \exists d_{10} \exists d_{11}$ is reproducible once we know the name (d) and arity (2) of the skolem term it refers to. We note together these two pieces of information by writing $\exists [d]^2$. Analogously, the factorized version of the whole prefix of (10) is written $\exists [b]^1 \exists [d]^2$. The factorized formula obtained from (10) is:

$$f^{III} = Fact(f^{II}) = \exists [b]^1 \exists [d]^2 .\ [d]_{\{01\}} \wedge [\neg b, \neg d]_{\{00,01,10,11\}} \wedge [b, \neg d]_{\{10,11\}} \wedge [b]_{\{0\}} \qquad (12)$$

and it univocally expands to (10) itself. Factorization transfers the (exponential) blowup from the size of the clause set into the size of the index sets mentioned as pedixes, without in fact getting rid of it. Nevertheless, we profit from this convenient notation in the paper.

Step 4. The scenario $\mathcal{I} \subseteq \mathbb{B}^{\delta(\Gamma)}$ associated to a factorized clause $\Gamma_\mathcal{I}$ can be compactly represented by delegating to a BDD the task of deciding which indexes are inside the scenario and which ones are excluded. We need as many decision variables in the support set as there are bits in the indexes, i.e. $\delta(\Gamma)$. We name such decision variables after the universal variables of the originating QBF, using the natural correspondence established by skolemization. In particular, the variable which decides the j-th bit in any index is given the name of the j-th universal variable in the total order induced by the left-to-right succession of quantifiers in the prefix. This naming scheme is independent of the specific skolem term or factorized clause we consider, for a linear prefix makes any skolem term of arity n have the first n universal variables as arguments.





For example, in the clause $[d]_{\{01\}}$ from (12), the first bit of the index 01 refers to the first argument of the skolem symbol associated to d, which is necessarily the first universal variable in the prefix, i.e. a. Similarily, the second bit refers to c. The same correspondence holds for any other clause, e.g. for $[b, \neg d]_{\{10,11\}}$. It follows that the bits of any scenario $\mathcal{I} \subseteq \mathbb{B}^{\delta(e)}$ in a fatorized literal $[e]_{\mathcal{I}}$ coming from a QBF f are associated to the decision variables $var_\forall(f, e)$. Similarly, the scenario $\mathcal{I}$ in a clause $[\Gamma]_{\mathcal{I}}$ is represented by a BDD whose support set is $var_\forall(f, \Gamma)$. By replacing the explicit scenarios mentioned in $[d]_{\{01\}}$ and $[b, \neg d]_{\{10,11\}}$ with the BDDs over $var_\forall(f, d)$ which recognize $\{01\}$ and $\{10, 11\}$ respectively, we obtain the two *symbolic* clauses depicted aside.
To turn an entire factorized formula into a symbolic form, it is convenient to associate a single forest of BDDs to the formula as a whole, rather than one separate BDD per clause. In so doing, we profit from the semantic canonicity of BDDs. In our graphical notation, symbolic clauses tell their associated scenarios by providing a "pointer" to the proper node in the forest. For example, the symbolic formula associated to (12) is depicted in Figure 1.

Notice that the decision order used in the forest is not constrained by the prefix. All the decision orders are acceptable, though the exact shape of the forest and amount of sharing depend on such order. In Figure 1 we depict the canonic versions of the forests associated to the possible orderings over $var_\forall(f)$. Whichever the order, factorized clauses over the same scenario (coming from QBF clauses mentioning an identical set of universal literals) point to the same node in the forest. Beyond this guaranteed amount of sharing, a stronger compression may take place, depending on the clause structure. In our example, the order $c \to a$ (which, incidentally, contradicts the prefix) allows for more sharing than $a \to c$.

By denoting with $Symb(\cdot)$ the transformation that introduces BDDs, we pose:

$$f^{IV} = Symb(f^{III}) = SymbSk(f) \qquad SymbSk(f) \doteq Symb(Fact(Prop(Sk(f)))) \tag{13}$$

It is possible to think of structures like the ones in Figure 1 as just custom compression mechanisms for the propositional expansion of skolemized quantified formulas. In this sense, it is interesting to observe that a direct definition of $SymbSk$ exists which sidesteps any exponential size blowup, and is computed in polynomial-time (Section 4.4).

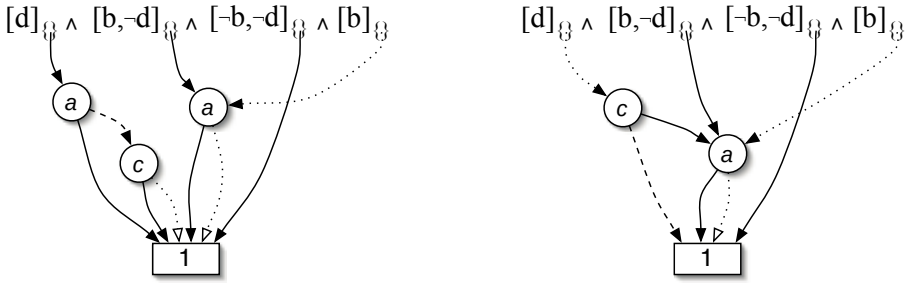


Figure 1: Symbolic skolemization for (1), with decision order $a \to c$ (left) and $c \to a$ (right).

Alternatively, symbolic formulas may be considered as members of an autonomous language, with its notions of model and satisfiability, its inference rules and decision procedures. This is the perspective we assume in the next section.

4. Symbolically Quantified Propositional Logic

In this section we characterize the SQPL formalism. Our primary concern is to formulate a language which can be endowed with effectively automatizable proof procedures, and which is at least as expressive as QBF[4]. The ultimate objective is indeed to trasform any given QBF into an "equivalent" SQPL formula, so to decide the former by reasoning on the latter.

The starting point for our construction is the notion of *quantification structure*, which is meant to capture in a language-independent way all the relevant information about how universal and existential variables relate to each other.

Definition 4.1 (Quantification Structure) *Given a finite set of propositional variables V, a* quantification structure *(QS) over V is a triple $\langle V_\exists, V_\forall, dom\rangle$ where $\{V_\exists, V_\forall\}$ is a partition of V, and $dom : V_\exists \mapsto 2^{V_\forall}$ is a tree-shaped dominance of $V_\forall$ over $V_\exists$.*

The intuition is that each universal variable $u \in dom(e)$ dominates $e \in V_\exists$, as the existential quantifier that binds e is in the scope of all the universal quantifiers that bind variables in $dom(e)$. Each element $[e]$ of the quotient set $V_\exists/\sim$ (where $a \sim b$ iff $dom(a) = dom(b)$) is called *existential scope*. The set of scopes can be equipped with the partial order relation $[a] \prec [b]$ iff $dom(a) \subset dom(b)$, so to obtain the *quantification tree* $(V_\exists/\sim, \prec)$.

Even though such tree-shaped structures are the ones we ultimately target, we provisionally restrict our formalization to the slightly less general case of *linear quantification structures*, for two reasons: First, in a large part of the literature only linear prefixes are considered, for it is always possible to cast a non-linear quantification into an equivalent[5] linear one. Second, the notation is strongly simplified under the linearity assumption, though a straightforward generalization is possible (we discuss this in Section 8).

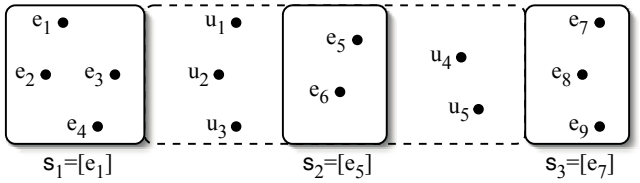
Linear QSs are those in which $V_\exists$ is linear w.r.t. dom, i.e. $\prec$ is a total order over $V_\exists/\sim$, so that $(V_\exists/\sim, \prec)$ has a unique branch. This special case is best captured by a simpler, ad-hoc definition which abstracts over the identities of universal variables.

Definition 4.2 (Mute Quantification Structure) *A* Mute Quantification Structure *(MQS) is a couple $\langle V_\exists, \delta\rangle$, where $V_\exists$ is a set of variables, and $\delta : V_\exists \mapsto \mathbb{N}^{\geq 0}$ is called* depth function *over $V_\exists$. The mute quantification structure associated to the quantification structure $\langle V_\exists, V_\forall, dom\rangle$ is $\langle V_\exists, \delta\rangle$, where $\delta : V_\exists \mapsto \{0, 1, \ldots, |V_\forall|\}$ is defined as $\delta(e) \doteq |dom(e)|$.*

4. By "as expressive" we mean capable to represent the meaning of any QBF with no exponential increase in the size of the representation.
5. This notion of equivalence is to be understood as logical equivalence between the original, arbitrarily-shaped quantified formula, and its *prenex* version obtained through a sequence of equivalence preserving rewrites that push quantifiers outwards, to eventually obtain a linear sequence of quantifications (the prefix) followed by a quantifier-free formula. The way non-prenex formulas compare to their equivalent prenex versions has been recently investigated in (Baaz & Leitsch, 1994; Egly, 1998) from a theorical perspective, and in (Egly, Seidl, Tompits, Woltran, & Zolda, 2003; Benedetti, 2005c; Giunchiglia, Narizzano, & Tacchella, 2006; Egly, Seidl, & Woltran, 2006) from a practical, solver-related viewpoint.

A mute QS only says how many universal variables dominates each existential variable, but does not specify their names. For linear quantification structures, this "relaxed" representation causes no loss of information so long as satisfiability is concerned. In particular, the set of existential scopes $V_\exists/\sim$ and their mutual relationships are accurately preserved in a mute QS by posing $a \sim b$ iff $\delta(a) = \delta(b)$ and $[a] \prec [b]$ if $\delta(a) < \delta(b)$. Furthermore, $\delta(e)$ suffices to recover $dom(e)$, provided the total order induced by the prefix over $V_\forall$ is known.

For example, a QS with $V_\exists = \{e_1, \ldots, e_9\}$, $V_\forall = \{u_1, \ldots, u_5\}$, $dom(x) = \emptyset$ for $x \in \{e_1, e_2, e_3, e_4\}$, $dom(x) = \{u_1, u_2, u_3\}$ for $x \in \{e_5, e_6\}$, and $dom(x) = V_\forall$ otherwise, can be visualized as a linear structure with three ordered existential scopes $s_1 \prec s_2 \prec s_3$:



which respectively contain variables dominated by $\emptyset = dom(e_1) \subset dom(e_5) \subset dom(e_7) = V_\forall$. The associated MQS is $\delta(e_1) = \cdots = \delta(e_4) = 0, \delta(e_5) = \delta(e_6) = 3, \delta(e_7) = \cdots = \delta(e_9) = 5$.

4.1 SQPL formulas

We portrayed symbolic formulas as objects in a bi-dimensional space, where the x and y axes are respectively concerned with existential and universal variables. In this section we formalize such intuition, building on top of quantification structures.

We extend the dom function in a QS $\langle V_\exists, V_\forall, dom\rangle$ to literals over $V_\exists$ as $dom(l) \doteq dom(var(l))$, and to sets of literals over $V_\exists$ as $dom(\Gamma) = \cup_{l\in\Gamma} dom(l)$. If the set of literals Γ is such that $var(\Gamma)$ is linear w.r.t. dom, then there is always at least one "deepest" literal $l \in \Gamma$ whose universal dependencies include all the others' ones, so that $dom(\Gamma) = dom(l)$.

Definition 4.3 (Symbolic literal, clause) *Given a quantification structure $\mathcal{Q} = \langle V_\exists, V_\forall, dom\rangle$ and a forest of BDDs $\mathcal{D}$ such that $supp(\mathcal{D}) \subseteq V_\forall$, we define:*

- *a* symbolic literal over $\mathcal{Q}$ and $\mathcal{D}$ *as a couple $\langle l, \mathcal{I}\rangle$, where l is a literal with $var(l) \in V_\exists$, and $\mathcal{I}$ is a (signed) node in $nodes(\mathcal{D})$ such that $supp(\mathcal{I}) \subseteq dom(l)$;*
- *a* symbolic clause over $\mathcal{Q}$ and $\mathcal{D}$ *as a couple $\langle \Gamma, \mathcal{I}\rangle$, where Γ is a consistent set of literals over $V_\exists$ such that $var(\Gamma)$ is linear w.r.t. dom, and $\mathcal{I}$ is a (signed) node in $nodes(\mathcal{D})$ such that $supp(\mathcal{I}) \subseteq dom(\Gamma)$.*

Definition 4.4 (Symbolic Formula) *A* symbolic formula *is a triple $\mathcal{F} = \langle \mathcal{Q}, \mathcal{D}, \Pi\rangle$, where*

- *$\mathcal{Q} = \langle V_\exists, V_\forall, dom\rangle$ is a quantification structure (called* prefix *of $\mathcal{F}$);*
- *$\mathcal{D}$ is a forest of BDDs with $supp(\mathcal{D}) \subseteq V_\forall$ (called* forest *of $\mathcal{F}$);*
- *Π is a set of symbolic clauses over $\mathcal{Q}$ and $\mathcal{D}$ (called* matrix *of $\mathcal{F}$).*

A symbolic formula is normal, *if $\mathcal{D}$ is root-minimal w.r.t. the set of nodes mentioned in Π.*

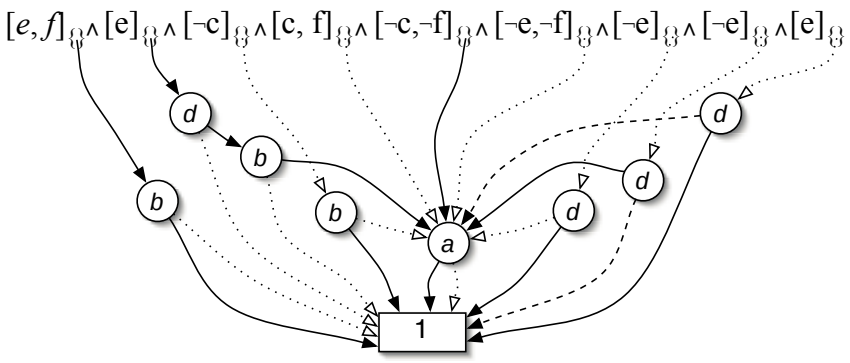


Figure 2: A symbolic formula with (linear) quantification structure defined by $V_\exists = \{c, e, f\}$, $V_\forall = \{a, b, d\}$, $dom(c) = \{a, b\}$, $dom(e) = dom(f) = \{a, b, d\}$.

Note how the components of each clause are forced to be mutually consistent by the quantification of the formula they belong to. Namely, Γ has to be linear w.r.t. *dom*, and it can only be coupled with BDDs that do *not* include decisions on variables outside $dom(\Gamma)$.

Thank to the canonic shape of the forest, different clauses are not just unrelated units of information. Any two distinct clauses may share (part of) the representation of their meaning, as the decision diagrams they point to exhibit common subgraphs by design.

For example, the objects in Figure 1 are symbolic formulas over the (linear) quantification structure defined by $V_\exists = \{b, d\}$, $V_\forall = \{a, c\}$, $dom(b) = \{a\}$, $dom(d) = \{a, c\}$. Another example is depicted in Figure 2. Information sharing is present in both cases.

Despite the crucial role played by BDDs our formal treatment sometimes abstracts over the presence of a forest. We do so by presenting our results not directly for symbolic formulas, but for an alternative incarnation thereof, the factorized formulas.

4.2 Factorized formulas

We extend the depth function δ in the MQS $\langle V, \delta\rangle$ to the literals over V as $\delta(\neg e) \doteq \delta(e), e \in V$, and to sets of literals as $\delta(\Gamma) \doteq \max_{l\in\Gamma} \delta(l)$. We omit henceforth the pedix in $V_\exists$ and write $\langle V, \delta\rangle$: No ambiguity arises as we disregarded names for universal variables.

Definition 4.5 (Factorized literal, clause) *A* factorized literal *over the MQS $\langle V, \delta\rangle$ is a couple $\langle l, \mathcal{I}\rangle$, where l is a literal over V, and $\mathcal{I} \subseteq \mathbb{B}^{\delta(l)}$. A* factorized clause *over the MQS $\langle V, \delta\rangle$ is a couple $\langle \Gamma, \mathcal{I}\rangle$, where Γ is a consistent set of literals over V, and $\mathcal{I} \subseteq \mathbb{B}^{\delta(\Gamma)}$.*

To note a factorized literal $\langle l, \mathcal{I}\rangle$ for which $\delta(l) > 0$ we write $[l]_\mathcal{I}$. We write just $[l]$ if $\delta(l) = 0$. Similarly, a factorized clause $\langle \Gamma, \mathcal{I}\rangle$, with $\Gamma = \{l_1, \ldots, l_h\}$ and $\mathcal{I} = \{i_0, \ldots, i_k\} \subseteq \mathbb{B}^{\delta(\Gamma)}$, is written as $\Gamma_\mathcal{I}$ or $[l_1, \ldots, l_h]_\mathcal{I}$ or $[l_1, \ldots, l_h]_{\{i_0,\ldots,i_k\}}$. A clause $\Gamma_\mathcal{I}$ with $\mathcal{I} = \emptyset$ and $\Gamma \neq \emptyset$ is called *evanescent*. A clause with $\mathcal{I} \neq \emptyset$ and $\Gamma = \emptyset$ is called *empty symbolic clause*, and is denoted by writing $[\,]$. The case $\Gamma = \emptyset$ and $\mathcal{I} = \emptyset$ is not in the language.

Definition 4.6 (Factorized formula) *A* factorized formula *is a couple $\mathcal{F} = \langle \mathcal{Q}, \Pi\rangle$, where $\mathcal{Q}$ is a MQS called the* factorized prefix *of $\mathcal{F}$, and Π is a set of factorized clauses over $\mathcal{Q}$, called the* factorized matrix *of $\mathcal{F}$.*

The notation we adopt for factorized formulas (Section 3, Step 3) is meant to resemble the one used to note QBFs in prenex conjunctive form. In particular, we denote the factorized matrix $\widetilde{\mathcal{F}} = \Pi$ by writing the "and"-separated list of the clauses in $\Pi = \{C_1, \ldots, C_n\}$, and its factorized prefix $\langle V, \delta \rangle$ by listing all the scopes in $V/\!\sim$. A scope $\{e_1, \ldots, e_k\} = [e] \in V/\!\sim$, $\delta(e) = w$ is noted as $\exists [e_1, \ldots, e_k]^w$. We obtain:

$$\text{Prefix}(\mathcal{F}) \;=\; \mathcal{P} \;=\; \exists [e_1^1, \ldots, e_{k_1}^1]^{w_1} \exists [e_1^2, \ldots, e_{k_2}^2]^{w_2} \cdots \exists [e_1^m, \ldots, e_{k_m}^m]^{w_m}$$

where $m = |V/\!\sim|$, $\delta(e_j^i) = w_j$ for $i = 1, \ldots, m$, $j = 1, \ldots, k_i$, and withouth loss of generality we assume $w_i < w_{i+1}$ for $i = 1, \ldots, m-1$. The entire formula is thus noted

$$\mathcal{F} \;=\; \mathcal{P}.\widetilde{\mathcal{F}} \;=\; \exists [e_1^1, \ldots, e_{k_1}^1]^{w_1} \exists [e_1^2, \ldots, e_{k_2}^2]^{w_2} \cdots \exists [e_1^m, \ldots, e_{k_m}^m]^{w_m}.\; C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

A sample factorized formula is (12), on page 10. Another example is

$$\begin{aligned} \exists [c]^2 \exists [e, f]^3.\quad & [e, f]_{\{000,001,100,101\}} \wedge [e]_{\{101,111\}} \wedge [\neg c]_{\{01\}} \wedge \\ & \wedge [c, f]_{\{100,101,110,111\}} \wedge [\neg c, \neg f]_{\{000,001,010,011\}} \wedge \\ & \wedge [\neg e, \neg f]_{\{100,101,110,111\}} \wedge [\neg e]_{\{100,110\}} \wedge [\neg e]_{\{001,011\}} \wedge [e]_{\{000\}} \end{aligned} \tag{14}$$

which, by compactly denoting scenarios, becomes

$$\begin{aligned} \exists [c]^2 \exists [e, f]^3.\quad & [e, f]_{\{*0*\}} \wedge [e]_{\{1*1\}} \wedge [\neg c]_{\{01\}} \wedge [c, f]_{\{1**\}} \wedge [\neg c, \neg f]_{\{0**\}} \wedge \\ & \wedge [\neg e, \neg f]_{\{1**\}} \wedge [\neg e]_{\{1*0\}} \wedge [\neg e]_{\{0*1\}} \wedge [e]_{\{000\}} \end{aligned} \tag{15}$$

The shift from symbolic to factorized formulas consists in replacing the (linear) QS by the associated MQS, and the BDDs in each clause by the scenarios they recognize. For example, (14) is the factorized version of the symbolic formula in Figure 2.

Factorized formulas may require exponentially more space to be noted than their symbolic counterpart, and they do not capture non-linear prefixes[6]. However, they are simple to note and manipulate, and they preserve most of the features of SQPL formulas. So long as the properties we study do not rely on specific details of the BDD way of encoding information, the two representations are equivalent and interchangeable.

4.3 Semantics for SQPL formulas

We are interested in defining when a (factorized) SQPL formula is *satisfiable*.

Definition 4.7 (Factorized assignment) *A* factorized assignment $eval_{\mathcal{Q}}$ *over the MQS* $\mathcal{Q} = \langle V, \delta \rangle$ *is any (partial) function that associates to* $v \in V$ *a couple* $\langle \mathcal{E}^+, \mathcal{E}^- \rangle$ *with* $\mathcal{E}^+, \mathcal{E}^- \subseteq \mathbb{B}^{\delta(v)}$. *An assignment is* consistent *if* $\mathcal{E}^+ \cap \mathcal{E}^- = \emptyset$ *for each* $v \in V$ *on which* $eval_{\mathcal{Q}}$ *is defined.*

We call the two elements in $eval(v)$ the positive and negative scenario associated to v, and note them $eval^+(v)$ and $eval^-(v)$ respectively, so $eval(v) = \langle \mathcal{E}^+, \mathcal{E}^- \rangle = \langle eval^+(v), eval^-(v) \rangle$. Assignment is extended to literals as $eval(l) = eval(v)$ if $l = v$, and $eval(l) = \langle eval^-(v), eval^+(v) \rangle$ if $l = \neg v$, and to sets of literals as $eval^+(\Gamma) \doteq \langle eval^+(\Gamma), eval^-(\Gamma) \rangle$, with $eval^+(\Gamma) \doteq \{i \in \mathbb{B}^{\delta(\Gamma)} : \exists l \in \Gamma, i|_{\delta(l)} \in eval^+(l)\}$, $eval^-(\Gamma) \doteq \{i \in \mathbb{B}^{\delta(\Gamma)} : \forall l \in \Gamma, i|_{\delta(l)} \in eval^-(l)\}$.

6. The extension of factorized formulas to non-linear prefixes requires a tree-shaped denotation for general prefixes and a slightly more complex mechanism to reconstruct the meaning of explicit scenarios in clauses. We abstain from this extension as the gain in generality does not justify the loss in readability.

Definition 4.8 (Factorized valuation for clauses, formulas) *Given a clause $C = \Gamma_{\mathcal{I}}$ over the quantification structure $\mathcal{Q}$ and a consistent factorized assignment $eval_{\mathcal{Q}}$, we pose:*

- $eval_{\mathcal{Q}}(C) = \mathbf{t}$ *if* $\mathcal{I} = \emptyset$ *or* $\mathcal{I} \subseteq eval^{+}_{\mathcal{Q}}(\Gamma)$*;*
- $eval_{\mathcal{Q}}(C) = \mathbf{f}$ *if* $\Gamma = \emptyset$ *or* $\mathcal{I} \cap eval^{-}_{\mathcal{Q}}(\Gamma) \neq \emptyset$*.*

For a factorized formula $\mathcal{F} = \langle \mathcal{Q}, \Pi \rangle$ we pose

- $eval(\mathcal{F}) = \mathbf{t}$ *if* $\Pi = \emptyset$ *or* $eval_{\mathcal{Q}}(C) = \mathbf{t}$ *for every* $C \in \Pi$*;*
- $eval(\mathcal{F}) = \mathbf{f}$ *if* $eval_{\mathcal{Q}}(C) = \mathbf{f}$ *for some* $C \in \Pi$*.*

The valuation of formulas and clauses is in general a partial function. A sufficient condition for it to be total is that the factorized assignment is total, i.e. $eval_{\mathcal{Q}}$ is total over V, and for every $v \in V$ it is $eval^{+}(v) \cup eval^{-}(v) = \mathbb{B}^{\delta(v)}$.

As usual, symbolic empty clauses represent contradictions, in that they evaluate to $\mathbf{f}$ under any valuation, and make any formula in which they appear evaluate to $\mathbf{f}$.

Definition 4.9 *We say that a factorized formula $\mathcal{F} = \langle \mathcal{Q}, \Pi \rangle$ is:*

- satisfiable*, if $eval_{\mathcal{Q}}(\mathcal{F}) = \mathbf{t}$ for some consistent assignment over $\mathcal{Q}$;*
- unsatisfiable*, if $eval_{\mathcal{Q}}(\mathcal{F}) = \mathbf{f}$ for every consistent assignment over $\mathcal{Q}$;*
- tautologic*, if $eval_{\mathcal{Q}}(\mathcal{F}) = \mathbf{t}$ for every total consistent assignment over $\mathcal{Q}$.*

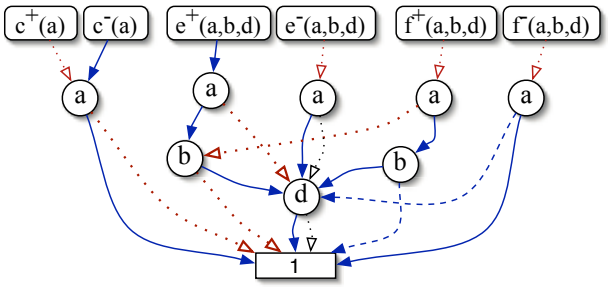
An assignment can be represented by the set of literals $\{[v]_{eval^{+}(v)} \mid v \in V\} \cup \{[\neg v]_{eval^{-}(v)} \mid v \in V\}$. Vice-versa, a set of literals S represents $eval^{+}(v) = \cup_{[v]_{\mathcal{I}} \in S} \mathcal{I}$, $eval^{-}(v) = \cup_{[\neg v]_{\mathcal{I}} \in S} \mathcal{I}$. An assignment such that $eval(\mathcal{F}) = \mathbf{t}$ is called *satisfying assignment* for $\mathcal{F}$. A set of literals is a *model* for $\mathcal{F}$ iff the assignment it represents satisfies $\mathcal{F}$. For example, we can check that

$$\{[c]_{\{1*\}}, [\neg c]_{\{0*\}}, [e]_{\{000,1*1\}}, [\neg e]_{\{1*0,0*1\}}, [f]_{\{100,001\}}, [\neg f]_{\{1*1\}}\} \tag{16}$$

is a model for (14), which is thus satisfiable. Conversely, (12) has no model, hence it is unsatisfiable. The meaning of a formula $\mathcal{F}$ is the set of all its models, denoted by $\mathcal{M}(\mathcal{F})$.

Definition 4.10 *Two formulas $\mathcal{F}$ and $\mathcal{F}'$ are* logically equivalent *(or, simply,* equivalent*) if $\mathcal{M}(\mathcal{F}) = \mathcal{M}(\mathcal{F}')$. They are* satisfiability equivalent *if $|\mathcal{M}(\mathcal{F})| = 0$ iff $|\mathcal{M}(\mathcal{F}')| = 0$.*

All the notions presented in this section are readily extended to symbolic formulas by demanding to BDDs the representation of scenarios. For example, a symbolic assignment over $\mathcal{Q} = \langle V_{\exists}, V_{\forall}, dom \rangle$ is represented by a forest of BDDs containing, for each $v \in V_{\exists}$, two roots $\langle \mathcal{E}^{+}(v), \mathcal{E}^{-}(v) \rangle$ with support set inside $dom(v)$. They represent the positive and negative scenarios for v, respectively. The picture aside presents a symbolic satisfying assignment for the SQBF in Figure 2. This representation is the symbolic version of (16) in the same sense as Figure 2 is a symbolic version of (14). Further details are in (Benedetti, 2005b).



4.4 Symbolic skolemization

Let us consider a closed QBF in prenex conjunctive normal form:

$$f = Q_1 v_1 \, Q_2 v_2 \cdots Q_n v_n . \widetilde{f} \tag{17}$$

where $Q_i \in \{\exists, \forall\}$ for $i = 1, \ldots, n$ and v_i are distinct variable names.

Definition 4.11 (Associated Quantification Structure) *The quantification structure associated to a prenex QBF in the form (17) is $\langle V_\exists, V_\forall, dom \rangle$, where $V_\exists = \{v_i : Q_i = \exists\}$, $V_\forall = \{v_i : Q_i = \forall\}$, and for each $v_i \in V_\exists$, it is $dom(v_i) \doteq \{v_j | j < i, Q_j = \forall\}$.*

The MQS associated to a QBF like (17) is the MQS associated to its QS.

Definition 4.12 (Symbolic skolemization) *The symbolic skolemization $SymbSk(f)$ of a QBF f is the SQPL formula $\langle \mathcal{Q}, \mathcal{D}, \Pi \rangle$, where $\mathcal{Q} = \langle V_\exists, V_\forall, dom \rangle$ is the quantification structure associated to f, $\mathcal{D}$ is a BDD forest on $V_\forall$ root-minimal w.r.t. the BDDs in Π, and Π contains, for each clause $C \in \widetilde{f}$, a symbolic clause $\Gamma_\mathcal{I}$ obtained as follows:*

- *if $\{l_1, \ldots, l_h\}$ are the existential literals in C, then $\Gamma = [l_1, \ldots, l_h]$;*
- *if $\{\phi_1 \oplus u_1, \ldots, \phi_k \oplus u_k\}$ are the universal literals in C, then $\mathcal{I}$ is a BDD, in the forest $\mathcal{D}$, which recognizes the scenario $u_1 = \phi_1, \ldots, u_k = \phi_k$. If there is no universal literal in C, then $\mathcal{I}$ is the sink node "1".*

Theorem 4.1 *For any QBF f, $SymbSk(f)$ is satisfiable iff f is true.*

For any QBF f, the factorized formula associated to the SQPL formula $SymbSk(f)$ is called *factorized skolemization of f*, and is denoted as $FactSk(f)$. For example, the symbolic skolemization of the QBF (1) is given in Figure 1 (Section 3), and its associated factorized formula is (12), which is unsatisfiable, hence (1) is false. Another example is given in Figure 2, where the symbolic skolemization of the QBF

$$\begin{aligned} \forall a \forall b \exists c \forall d \exists e \exists f .\ & (\neg b \vee e \vee f) \wedge (a \vee c \vee f) \wedge (a \vee d \vee e) \wedge (\neg a \vee \neg b \vee \neg d \vee e) \wedge (\neg a \vee b \vee \neg c) \wedge \\ & \wedge (\neg a \vee \neg c \vee \neg f) \wedge (a \vee \neg d \vee \neg e) \wedge (\neg a \vee d \vee \neg e) \wedge (a \vee \neg e \vee \neg f) \end{aligned} \tag{18}$$

is depicted. That formula, associated to (14), is satisfiable, hence (18) is true.

4.5 Propositional Expansion

Each SQPL formula may be associated to an equivalent SAT instance as follows.

Definition 4.13 (Propositional Expansion) *The* propositional expansion *of a (mute) quantification structure $\mathcal{Q} = \langle V, \delta \rangle$ is the set of propositional symbols $Pexp(\mathcal{Q}) \doteq \{e_\Psi \mid e \in V, \Psi \in \mathbb{B}^{\delta(e)}\}$. The* propositional expansion *of a clause $\Gamma_\mathcal{I}$ over the quantification structure $\mathcal{Q} = \langle V, \delta \rangle$ is a CNF propositional formula with variables in $Pexp(\mathcal{Q})$ defined as*

$$Pexp(\Gamma_\mathcal{I}) \doteq \bigwedge_{\Psi \in \mathcal{I}} \left(\bigvee_{\phi \oplus e \in \Gamma} \phi \oplus e_{\Psi|_{\delta(e)}} \right) \tag{19}$$

The propositional expansion *of $\mathcal{F} = \langle \mathcal{Q}, \Pi \rangle$ is $Pexp(\mathcal{F}) \doteq \wedge_{\Gamma_\mathcal{I} \in \Pi} Pexp(\Gamma_\mathcal{I})$.*

According to this definition, there are always as many propositional clauses in $Pexp(\Gamma_{\mathcal{I}})$ as there are indexes in $\mathcal{I}$, while the shape of the expanded clauses and the number of distinct literals they mention are strongly dependent on the quantification structure.

For example, the expansion of (12) is (10). The expansion of a symbolic formula is just the expansion of the factorized formula associated to it. For example, the expansion of (the matrix of) the formula in Figure 2, i.e. the expansion of (14), is

$$\begin{gathered}(e_{000} \vee f_{000}) \wedge (e_{001} \vee f_{001}) \wedge (e_{100} \vee f_{100}) \wedge (e_{111} \vee f_{111}) \ \wedge \ (e_{101}) \wedge (e_{111}) \wedge \\ \wedge \ (\neg c_{01}) \ \wedge \ (c_{10} \vee f_{100}) \wedge (c_{10} \vee f_{101}) \wedge (c_{11} \vee f_{110}) \wedge (c_{11} \vee f_{111}) \wedge \\ \wedge \ (\neg c_{00} \vee \neg f_{000}) \wedge (\neg c_{00} \vee \neg f_{001}) \wedge (\neg c_{01} \vee \neg f_{010}) \wedge (\neg c_{01} \vee \neg f_{011}) \wedge \\ \wedge \ (\neg e_{100} \vee \neg f_{100}) \wedge (\neg e_{101} \vee \neg f_{101}) \wedge (\neg e_{110} \vee \neg f_{110}) \wedge (\neg e_{111} \vee \neg f_{111}) \wedge \\ \wedge \ (\neg e_{100}) \wedge (\neg e_{110}) \wedge (\neg e_{001}) \wedge (\neg e_{011}) \ \wedge \ (e_{000})\end{gathered} \tag{20}$$

When we mix symbolic constructs (literals, clauses, formulas) and the propositional formulas associated to them via $Pexp$, we name *ground* the latters. For example, the propositional expansion of the symbolic literal $[l]_{\{01,10,11\}}$ contains the ground literals l_{01}, l_{10}, and l_{11}, and clauses in (20) are ground, as opposed to the symbolic objects they come from.

The key property of a propositional expansion is stated in the following result.

Lemma 4.1 *For every factorized formula $\mathcal{F}$, $Pexp(\mathcal{F})$ is satisfiable (in the standard SAT sense) iff $\mathcal{F}$ is satisfiable (according to Definition 4.9).*

Not only a formula and its expansion are satisfiability-equivalent, but a close relationship between their models exists. Given a set of literals $\mathcal{A}$ on the quantification structure $\mathcal{Q}$, let us denote by $Pexp(\mathcal{A})$ the conjunction of the expansion of every literal in $\mathcal{A}$ w.r.t. $\mathcal{Q}$.

Lemma 4.2 *For every formula $\mathcal{F}$, $\mathcal{A}$ is a model of $\mathcal{F}$ iff $Pexp(\mathcal{A})$ is a model of $Pexp(\mathcal{F})$.*

This lemma allows—at least in principle—to decide factorized formulas by just enrolling a SAT solver (more on this in Section 6). Furthermore, it simplifies the proof of many results on SQPL, which can be estabilished in terms of classical results for propositional logic.

4.6 The size of symbolic formulas

The presence of a BDD-based representation for scenarios cannot be ignored when it comes to reason about the *size* of the formulas we handle.

Definition 4.14 (SQPL size) *For a formula $\mathcal{F} = \langle \mathcal{Q}, \mathcal{D}, \Pi \rangle$, we define three metrics:*

- *the* symbolic size $|\mathcal{F}|_{symb} \doteq |\Pi|$*, where* $|\Pi| = \sum_{\Gamma_{\mathcal{I}} \in \Pi} |\Gamma|$*;*
- *the* structural size $|\mathcal{F}| \doteq |\mathcal{D}| + |\Pi|$ *($|\mathcal{D}|$ is the number of decision nodes in the forest);*
- *the* ground size $|\mathcal{F}|_{grn} \doteq |Pexp(\mathcal{F})|$ *(the size of the expansion is measured in literals).*

The structural size measures the amount of space needed to store all the components of a symbolic formula, while its ground size quantifies the amount of space necessary to spell out its propositional expansion. The symbolic size of a formula depends on the decision order while its ground size is invariant. For example, the SQBF in Figure 1 has symbolic size 9 for the decision order $a \prec c$, and 8 for $c \prec a$. In both cases, its ground size is 14 (8 in the

clausal sense), as it follows from counting the number of literals and clauses in (11). For the SQBF $\mathcal{F}$ in Figure 2, it is $|\mathcal{F}|_{symb} = 21$, $|\mathcal{F}|^{cls}_{symb} = 17$, $|\mathcal{F}|_{grn} = 40$, and $|\mathcal{F}|^{cls}_{grn} = 24$.

It makes sense to wonder about how complex is to compute $|\mathcal{F}|_{grn}$. Fortunately, while the construction of $Pexp(\mathcal{F})$ may require exponential time and space, an exact assessment of its size does not require the actual construction, and is cheap to compute.

Lemma 4.3 *For any symbolic formula $\mathcal{F}$, $|\mathcal{F}|_{grn}$ can be computed in time $O(|\mathcal{F}|)$ and $|\mathcal{F}|_{grn} \geq |\mathcal{F}|_{symb}$. If $\mathcal{F} = SymbSk(f)$, then $|\mathcal{F}| \leq |f|$ and $|\mathcal{F}|_{grn} = \sum_{\Gamma \in \widetilde{f}} 2^{\delta(\Gamma) - |var_\forall(\Gamma)|}$.*

5. Inferences and other operations over SQPL formulas

We introduce some manipulations of the bi-dimensional structure of symbolic formulas. The emphasis is on designing symbolic versions of standard propositional inference rules whose effect on $\mathcal{F}$ can be understood in terms of the changes they perform on its propositional expansion. Large (possibly exponential) sets of inferences over $Pexp(\mathcal{F})$ are performed at once, as single operations on $\mathcal{F}$. The key idea is to take "universal reasoning" apart from "existential reasoning", and let BDD primitives operate the former, while standard clause-oriented manipulations cope with the latter. We describe the factorized form of each rule. The symbolic version is obtained by demanding to BDDs the operations on scenarios.

5.1 Resolution and Subsumption

We generalize some classical propositional inference rules, denoted by the usual notation

$$\frac{\alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_n}{\beta_1 \quad \beta_2 \quad \cdots \quad \beta_m} \qquad \text{and} \qquad \frac{\alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_n}{\beta_1 \quad \beta_2 \cdots \quad \beta_m}$$

where the left side designates an expansion rule which adds the inferred clauses $\beta_1, \beta_2, \ldots, \beta_m$ to the already known premises $\alpha_1, \alpha_2, \ldots, \alpha_n$, while on the right side a contraction rule is shown which replaces the clauses $\alpha_1, \alpha_2, \ldots, \alpha_n$ with $\beta_1, \beta_2, \ldots, \beta_m$, thus performing a deletion (if the consequences are a proper subset of the premises) or a simplification (otherwise).

Definition 5.1 (Symbolic Binary Resolution) *Given two clauses $\Gamma'_{\mathcal{I}'}, \Gamma''_{\mathcal{I}''} \in \Pi$ in $\mathcal{F} = \langle \mathcal{Q}, \Pi \rangle$, with $\mathcal{Q} = \langle V, \delta \rangle$, and any variable $e \in V$ such that $e \in \Gamma'$, $\neg e \in \Gamma''$ and $\delta(e) = \min(\delta(\Gamma'), \delta(\Gamma''))$, we call* symbolic binary resolution *(SBR) the inference rule*

$$\frac{\Gamma'_{\mathcal{I}'} \quad \Gamma''_{\mathcal{I}''}}{\Gamma_{\mathcal{I}}} \; (SBR) \quad \textit{where } \Gamma = (\Gamma' \setminus \{e\} \cup \Gamma'' \setminus \{\neg e\}) \textit{ and } \mathcal{I} = (\mathcal{I}' \cap \mathcal{I}'')|_{\delta(\Gamma)}$$

The generated clause $\Gamma_{\mathcal{I}}$ is the symbolic resolvent *of the two* resolving clauses $\Gamma'_{\mathcal{I}'}$ *and* $\Gamma''_{\mathcal{I}''}$.

Let us consider, for example, the formula

$$\exists [a]^1 \exists [b]^2 \exists [c]^3 . \, [a, \neg b, c]_{\{1**, 01*, 001\}} \wedge [\neg a, b]_{\{**\}} \wedge [\neg b, \neg c]_{\{*0*\}} \wedge [b]_{\{10\}} \wedge [\neg a, \neg b]_{\{**\}} \quad (21)$$

The first clause does not resolve against the second one on a, despite this variable is mentioned in opposite polarities, because $1 = \delta(a) \neq \min(\delta([a, \neg b, c]_{\{1**, 01*, 001\}}), \delta([b]_{\{10\}})) = 2$. Rather, the first and third clause resolve on c, yielding a resolvent of universal depth 2:

$$\frac{[a, \neg b, c]_{\{1**, 01*, 001\}} \quad [\neg b, \neg c]_{\{*0*\}}}{[a, \neg b]_{\{*0\}}}$$

which can in turn be resolved against $[b]_{\{10\}}$, yielding $[a]_{\{1\}}$. We are entitled to conclude that a symbolic formula is unsatisfiable if (and only if) we are able to derive the empty symbolic clause from it, through (a sequence of applications of) symbolic resolution.

Theorem 5.1 *SBR is a sound and complete inference rule for SQPL.*

For example, let us consider the following resolution derivation for the empty clause, constructed starting from the formula (12) on page 10:

$$\dfrac{\dfrac{\dfrac{[a,\neg b,c]_{\{1**,01*,001\}} \quad [\neg b,\neg c]_{\{*0*\}}}{[a,\neg b]_{\{*0\}}} \quad [b]_{\{10\}}}{[a]_{\{1\}}} \quad \dfrac{[b]_{\{10\}} \quad [\neg a,\neg b]_{\{**\}}}{[\neg a]_{\{1\}}}}{[\,]}$$

This proves that (12) is unsatisfiable. As usual with resolution-based proof procedures, some complete resolution strategy is to be selected to automatize proof search. We consider a symbolic version of *variable elimination* which turns resolution into a satisfiability-preserving simplification rule by properly arranging multiple SBR applications.

Definition 5.2 (Symbolic Variable Elimination) *Given a formula $\mathcal{F} = \langle \mathcal{Q}, \Pi \rangle$, with $\mathcal{Q} = \langle V, \delta \rangle$, the* symbolic elimination *of a variable $e \in \mathcal{V}$ with $\delta(e) = \delta(\Pi)$ is the* contraction

$$\dfrac{\alpha_1 \cdots \alpha_n \quad \beta_1 \cdots \beta_m}{\gamma_{1,1} \cdots \gamma_{1,m} \;\; \cdots \;\; \gamma_{n,m}} \; (SVE)$$

where $\alpha_1 \cdots \alpha_n \in \Pi$ are all the clauses mentioning e, $\beta_1 \cdots \beta_m \in \Pi$ are all the clauses mentioning $\neg e$, and the $n \cdot m$ clauses $\gamma_{n,m}$ are such that the inferences

$$\dfrac{\alpha_i \quad \beta_j}{\gamma_{i,j}} \; (SBR)$$

are valid for each $i = 1, \ldots n$ and $j = 1, \ldots m$.

By dropping parent clauses, SVE fails to preserve equivalence. However, it preserves satisfiability while monotonically shrinking the set of variables. Its repeated application necessarily leads to either the empty clause or the empty formula, hence to decide the original formula.

As usual in clause-based reasoning, the injection of superfluous entailed clauses injected by SVE (or by other inference rules) can be partly contrasted by *subsumption* rules.

Definition 5.3 (Symbolic Subsumption) *Given two clauses $\Gamma'_{\mathcal{I}'}, \Gamma''_{\mathcal{I}''} \in \Pi$ in $\mathcal{F} = \langle \mathcal{Q}, \Pi \rangle$, with $\mathcal{Q} = \langle V, \delta \rangle$, such that $\Gamma \subseteq \Gamma'$, we define three* symbolic subsumption *rules*

$$\dfrac{\Gamma_{\mathcal{I}} \quad \Gamma'_{\mathcal{I}'}}{\Gamma_{\mathcal{I}}} \; (t\text{-}SSUB) \qquad \dfrac{\Gamma_{\mathcal{I}} \quad \Gamma'_{\mathcal{I}'}}{\Gamma_{\mathcal{I}'\cup\mathcal{I}}} \; (m\text{-}SSUB) \qquad \dfrac{\Gamma_{\mathcal{I}} \quad \Gamma'_{\mathcal{I}'}}{\Gamma_{\mathcal{I}} \quad \Gamma'_{\mathcal{I}'\cap\overline{\mathcal{I}}}} \; (p\text{-}SSUB)$$

The total *symbolic subsumption rule (t-$SSUB$) and the* merging *subsumption rule (m-$SSUB$) are two deletion rules that are only applied when $\mathcal{I}'|_{\delta(\Gamma)} \subseteq \mathcal{I}$ and $\Gamma = \Gamma'$ respectively.* Partial *symbolic subsumption (p-$SSUB$) is a contraction rule, and is always applicable.*

Total subsumption is akin to standard propositional subsumption, in that it expunges clauses which express redundant constraints. Merging subsumption joins multiple symbolic clauses with the same literals in one single clause over the union of their scenarios. For example, after applying all merging subsumptions to formula (15) on page 15, we obtain

$$\exists[c]^2\exists[e]^3\exists[f]^3.\quad \begin{array}{l}[e,f]_{\{*0*\}} \wedge [e]_{\{1*1,000\}} \wedge [\neg c]_{\{01\}} \wedge [c,f]_{\{1**\}} \wedge \\ \wedge [\neg c,\neg f]_{\{0**\}} \wedge [\neg e,\neg f]_{\{1**\}} \wedge [\neg e]_{\{1*0,0*1\}}\end{array} \tag{22}$$

and after applying subsumption in all its forms we obtain

$$\exists[c]^2\exists[e]^3\exists[f]^3.\quad \begin{array}{l}[e,f]_{\{001,100\}} \wedge [e]_{\{1*1,000\}} \wedge [\neg c]_{\{01\}} \wedge [c,f]_{\{1**\}} \wedge \\ \wedge [\neg c,\neg f]_{\{00*\}} \wedge [\neg e,\neg f]_{\{1*1\}} \wedge [\neg e]_{\{1*0,0*1\}}\end{array} \tag{23}$$

Theorem 5.2 *The inference rules x-SSUB, $x \in \{t, m, p\}$, are sound for SQPL.*

At the symbolic level, all the rules described in this section require BDD operations which are polynomial (in most cases) in the structural size of the involved clauses.

5.2 Normalization

According to Definition 4.8, the presence of evanescent clauses never impacts on the satisfiability of the overall formula. So, these clauses are always safely removed.

Lemma 5.1 *The "evanescence elimination" inference, consisting in removing every evanescent clause from a given symbolic formula, is a sound contraction rule.*

The recognition of evanescent clauses is performed in constant time as BDDs are canonic. In the rest we assume that formulas are normalized w.r.t. evanescence, i.e. if any inference yields clauses with an empty scenario, those clauses are dropped immediately.

p-SSUB is an example of a rule that can generate evanescent clauses. Total subsumption is indeed a special case of partial subsumption, in which the second resolvent is evanescent. We keep the two cases distinct, as only the former rule is a deletion. We call *subsumption-normalized* those formulas to which total and merging subsumption have been applied until fixpoint. This normalization is meant to reduce the size of the propositional expansion by expunging sets of propositionally subsumed clauses by means of a few symbolic operations.

A further level of normalization is attained by abstracting over the presence of dimensions (for the scenarios in the clauses) which are oblivious to the overall meaning of the formula.

Definition 5.4 (abstraction operator) *Given any set $\mathcal{I} \subseteq \mathbb{B}^n$, any integer i such that $1 \leq i \leq n$, and a boolean parameter $\phi \in \mathbb{B}$, we pose $\mathcal{I}{\downarrow}^i \doteq \mathcal{I}{\downarrow}^i_0 \cup \mathcal{I}{\downarrow}^i_1$, where*

$$\mathcal{I}{\downarrow}^i_\phi \doteq \{\langle\psi_1,\ldots,\psi_{i-1},\psi_{i+1},\ldots,\psi_n\rangle \mid \langle\psi_1,\ldots,\psi_{i-1},\phi,\psi_{i+1},\ldots,\psi_n\rangle \in \mathcal{I}\} \subseteq \mathbb{B}^{n-1}$$

This abstraction operator is extended to clauses and formulas as follows.

Definition 5.5 *For a clause $C = \Gamma_\mathcal{I}$ over a quantification structure $\mathcal{Q} = \langle V, \delta\rangle$ we pose $C{\downarrow}^i_\phi \doteq \Gamma_\mathcal{I}$ if $i > \delta(\Gamma)$, and $C{\downarrow}^i_\phi \doteq \Gamma_{\mathcal{I}{\downarrow}^i_\Phi}$ otherwise. For a formula $\mathcal{F} = \langle\mathcal{Q}, \Pi\rangle$ we pose $\mathcal{F}{\downarrow}^i_\phi \doteq \langle\mathcal{Q}', \Pi'\rangle$, where $\Pi' = \{C{\downarrow}^i_\Phi \mid C \in \Pi\}$, $\mathcal{Q}' = \langle V, \delta'\rangle$, $\delta'(v) = \delta(v) - 1$ if $\delta(v) \geq i$ and $\delta'(v) = \delta(v)$ otherwise. The operator ${\downarrow}^i$ is similarly extended.*

The result we use to achieve further normalization is as follows.

Theorem 5.3 (Pruning of oblivious scenarios) *Given $\mathcal{F} = \langle \mathcal{Q}, \Pi \rangle$, let $i \leq \delta(\mathcal{F})$ be such that $\forall \Gamma_{\mathcal{I}} \in \Pi, i \leq \delta(\Gamma)$ implies $\mathcal{I}{\downarrow_0^i} = \mathcal{I}{\downarrow_1^i}$. Then, $\mathcal{F}{\downarrow^i}$ is satisfiable iff $\mathcal{F}$ is satisfiable.*

We say that a formula is normalized w.r.t. oblivious scenarios if the condition in Theorem 5.3 applies for no $i \leq \delta(\mathcal{F})$. Such normalized situation can be attained by iteratively replacing $\mathcal{F}$ with $\mathcal{F}{\downarrow^i}$, until fixpoint. For example, the formula

$$\mathcal{F} = \exists [a]^1 \exists [b]^3 . \, [a, b]_{\{0*1, 1*0\}} \wedge [a, \neg b]_{\{1*1\}} \wedge [\neg a]_{\{0\}} \tag{24}$$

is not normal, as the condition applies with $i = 2$. The normalized form is

$$\mathcal{F}{\downarrow^2} = \exists [a]^1 \exists [b]^2 . \, [a, b]_{\{01, 10\}} \wedge [a, \neg b]_{\{11\}} \wedge [\neg a]_{\{0\}} \tag{25}$$

Normalization is easier to define at the symbolic (BDD-based) representation level.

Definition 5.6 (Symbolic pruning of oblivious scenario) *The normalized version of $\langle \mathcal{Q}, \mathcal{D}, \Pi \rangle$ is $\langle \mathcal{Q}', \mathcal{D}, \Pi \rangle$, where $\mathcal{Q} = \langle V_\exists, V_\forall, dom \rangle$, $\mathcal{Q}' = \langle V_\exists, V'_\forall, dom \rangle$, $V'_\forall = V_\forall \cap supp(\mathcal{D})$.*

Essentially, the forest of diagrams and clause sets are untouched, and it suffices to remove from the quantification structure those universal variables upon which no BDD performs decisions. This operation coincides with normalization as described above because $\mathcal{I}{\downarrow_0^i} = \mathcal{I}{\downarrow_1^i}$ if and only if no decision node on the variable of index i belongs to the BDD $\mathcal{I}$.

Most BDD operations only rely on the actual structure of the forest, and not on the nominal set of decision variables cited in the prefix. Hence, the fact that formulas may loose their normal form as a consequence of inferences and manipulations is sometimes inconsequential. However, in specific occasions, e.g. during expansion, a normalized form is convenient. For example, the expansion of the matrix of (24) is

$$(a_0 \vee b_{001}) \wedge (a_0 \vee b_{011}) \wedge (a_1 \vee b_{100}) \wedge (a_1 \vee b_{110}) \wedge (a_1 \vee \neg b_{101}) \wedge (a_1 \vee \neg b_{111}) \wedge (\neg a_0)$$

while the eqi-satisfiable expansion of the matrix of its normalized form (25) is

$$(a_0 \vee b_{01}) \wedge (a_1 \vee b_{10}) \wedge (a_1 \vee \neg b_{11}) \wedge (\neg a_0)$$

The latter can be in general exponentially smaller than the former.

To conclude, let us say that a formula is simply *normalized* if it is normalized w.r.t. both oblivious scenarios and evanescent/subsumed clauses. The elimination of redundant clauses preserves models, while the pruning of scenarios only preserves satisfiability.

5.3 Assignment and Split

One of the key manipulations propositional formulas undergo is called *literal assignment*. We assign a literal l in a clause-set f by first removing all the clauses containing l from f, and then removing all the occurrences of the literal $\neg l$ from the remaining clauses. This operation can be described as a combination of subsumption and resolution steps against the unit clause l, and as such it is readily ported to our symbolic framework.

Definition 5.7 (Symbolic Assignment) *Given a formula $\mathcal{F} = \langle \mathcal{Q}, \Pi \rangle$, with $\mathcal{Q} = \langle V, \delta \rangle$, and a literal $[l]_{\mathcal{I}}$ over $\mathcal{Q}$, let it be $\Pi = \{\alpha_1, \ldots, \alpha_n,\ \beta_1, \ldots, \beta_m,\ \gamma_1, \ldots, \gamma_k\}$, where $\alpha_1, \ldots \alpha_n$ and $\beta_1, \ldots, \beta_m$ are (all the) clauses that mention l and $\neg l$ respectively, while $\gamma_1, \ldots, \gamma_k$ do not mention $var(l)$. The result of the* symbolic assignment *of $[l]_{\mathcal{I}}$ in $\mathcal{F}$, noted $[l]_{\mathcal{I}} * \mathcal{F}$, is the formula $\langle \mathcal{Q}, \Pi' \rangle$ where $\Pi' = \{\alpha'_1, \alpha''_1, \ldots, \alpha'_n, \alpha''_n,\ \beta'_1, \ldots, \beta'_m,\ \gamma_1, \ldots, \gamma_k\}$ is such that*

$$\frac{\alpha_i \quad [l]_{\mathcal{I}}}{\alpha'_i \quad \alpha'_i}\ (p\text{-}SSUB) \qquad \textit{and} \qquad \frac{\beta_j \quad [l]_{\mathcal{I}}}{\beta'_j}\ (SBR)$$

are valid inferences for $i = 1, \ldots, n$ and $j = 1, \ldots, m$.

Notice that the SBR rule is always applicable under the conditions given in this definition as the constraint $\delta(e) = \min(\delta(\Gamma'), \delta(\Gamma''))$ is satisfied if at least one of the resolvent clauses is unit. For example, if $\mathcal{F}$ is (21), then the matrix of $\mathcal{F}' = [\neg b]_{\{0*,11\}} * \mathcal{F}$ is

$$\widetilde{\mathcal{F}'} = [a, \neg b, c]_{\{10*\}} \wedge [\neg a, b]_{\{10\}} \wedge [\neg a]_{\{0*,11\}} \wedge [\neg b, \neg c]_{\{10*\}} \wedge [b]_{\{10\}} \wedge [\neg a, \neg b]_{\{10\}} \qquad (26)$$

Symbolic assignment can be understood in terms of the underlying propositional expansion of $\mathcal{F}$. In particular, the expansion of $[l]_{\mathcal{I}} * \mathcal{F}$ is obtained from the expansion of $\mathcal{F}$ by assigning (in the standard propositional sense) all the literals in the expansion of $[l]_{\mathcal{I}}$. Formally:

Lemma 5.2 *For all the symbolic formulas $\mathcal{F} = \langle \mathcal{Q}, \Pi \rangle$ and all the literals $[l]_{\mathcal{I}}$ over $\mathcal{Q}$ it is $Pexp([l]_{\mathcal{I}} * \mathcal{F}) = Pexp([l]_{\mathcal{I}}) * Pexp(\mathcal{F})$.*

From a complexity perspective, let us notice that $Pexp([l]_{\mathcal{I}})$ may contain exponentially many elements (w.r.t. $|\mathcal{F}|$), while the computation of the symbolic assignment only involves list and BDD operations that can be completed in linear time.

Given a formula $\mathcal{F} = \langle \mathcal{Q}, \Pi \rangle$ with $\mathcal{Q} = \langle V, \delta \rangle$, let us define the *minimal depth* of a clause $\Gamma_{\mathcal{I}} \in \Pi$ as $\delta_{\min}(\Gamma_{\mathcal{I}}) \doteq \min_{l \in \Gamma} \delta(l)$, and the *minimal depth* of $\mathcal{F}$ as $\delta_{\min}(\mathcal{F}) \doteq \min_{\Gamma \in \Pi} \delta_{\min}(\Gamma)$.

Definition 5.8 (α and β symbolic formulas) *A symbolic formula is an α-formula if $\delta_{\min}(\mathcal{F}) > 0$, and a β-formula if $\delta_{\min}(\mathcal{F}) = 0$.*

Theorem 5.4 (Split Rules) *A non-trivial formula $\mathcal{F}$ is satisfiable if and only if*

- *$\mathcal{F}$ is an α-formula and both $\mathcal{F}{\downarrow_0^i}$ and $\mathcal{F}{\downarrow_1^i}$ are satisfiable for some $1 \leq i \leq \delta_{\min}(\mathcal{F})$;*
- *$\mathcal{F}$ is a β-formula and either $[l] * \mathcal{F}$ or $[\neg l] * \mathcal{F}$ (or both) are satisfiable for any literal $[l]$ such that $\delta(l) = \delta_{\min}(\mathcal{F}) = 0$.*

A natural recursive decision procedure can be designed on top of this result, akin to the and/or search performed by the extension of the classical DLL algorithm to quantified formulas (see Section 8). Such procedure is described in Section 6.

5.4 Symbolic Unit Clause Propagation and Pure Literal Elimination

While deciding satisfiability, the unconditioned assignment of a literal is only justified if it preserves the meaning of the formula, or, at least, its satisfiability. Some easy sufficient conditions for this to hold are inherited from propositional logic, and are presented here.

Let us call *symbolic unit clause* any clause $\Gamma_{\mathcal{I}}$ in which $|\Gamma| = 1$.

Definition 5.9 (Symbolic Unit Clause Propagation) *Given a formula $\mathcal{F} = \langle \mathcal{Q}, \Pi \rangle$, and a unit clause $[l]_{\mathcal{I}} \in \Pi$, the formula $[l]_{\mathcal{I}} * \mathcal{F}$ is said to be obtained from $\mathcal{F}$ by symbolic unit clause resolution* (SUCR) *over $[l]_{\mathcal{I}}$. We call* Symbolic Unit Clause Propagation (SUCP) *the application of SUCR repeated until no unit clause appears any longer in the formula.*

This rule is easily interpreted from a ground perspective. Under any quantification structure, a symbolic unit clause $[l]_{\mathcal{I}}$ expands into $Pexp([l]_{\mathcal{I}})$, which is a set of $|\mathcal{I}|$ unit clauses, all of which need to be satisfied by assigning to true their unique literal to avoid an immediate contradiction. This set of assignments is performed, all at once, as $[l]_{\mathcal{I}} * \mathcal{F}$, and during this operation the meaning of the formula is fully preserved.

Further symbolic units, besides the ones already present in the initial formula, may be generated by SUCR itself, and they are in turn assigned, or *propagated*. Unit clause propagation is a confluent process in PL. This property is inherited by SUCP, hence its result is univocally determined. Indeed, SUCP induces a particular order of propagation from the viewpoint of the expansion $Pexp(\mathcal{F})$, but no unit will be eventually missed, as no unit exists in $Pexp(\mathcal{F})$ which does not come from the expansion of a symbolic unit in $\mathcal{F}$.

Theorem 5.5 *SUCP preserves satisfiability.*

After SUCP reaches its fixpoint, either the empty clause has been generated (unsatisfiable formulas), or the empty formula is obtained (satisfiable formulas), or none of these two cases occur. The latter situation is the one we expect in general, as SUCP is not refutationally complete. Nevertheless, a class of formulas can be decided by just SUCP. Let us consider for example formula (23). By noting SUCR steps as simplification rules, we obtain:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{[e,f]_{\{001,100\}}\ [e]_{\{1*1,000\}}\ [\neg c]_{\{01\}}\ [c,f]_{\{1**\}}\ [\neg c,\neg f]_{\{00*\}}\ [\neg e,\neg f]_{\{1*1\}}\ [\neg e]_{\{1*0,0*1\}}}\ (\text{on } [\neg c]_{\{01\}})}{[e,f]_{\{001,100\}}\ [e]_{\{1*1,000\}}\ [c,f]_{\{1**\}}\ [\neg c,\neg f]_{\{00*\}}\ [\neg e,\neg f]_{\{1*1\}}\ [\neg e]_{\{1*0,0*1\}}}\ (\text{on } [e]_{\{1*1,000\}})}{[e,f]_{\{001,100\}}\ [c,f]_{\{1**\}}\ [\neg c,\neg f]_{\{00*\}}\ [\neg f]_{\{1*1\}}\ [\neg e]_{\{1*0,0*1\}}}\ (\text{on } [\neg e]_{\{1*0,0*1\}})}{[f]_{\{001,100\}}\ [c,f]_{\{1**\}}\ [\neg c,\neg f]_{\{00*\}}\ [\neg f]_{\{1*1\}}}\ (\text{on } [f]_{\{001,100\}})}{[c,f]_{\{1*1,110\}}\ [\neg c]_{\{00\}}\ [\neg c,\neg f]_{\{000\}}\ [\neg f]_{\{1*1\}}}\ (\text{on } [\neg c]_{\{00\}})}{[c,f]_{\{1*1,110\}}\ [\neg f]_{\{1*1\}}}\ (\text{on } [\neg f]_{\{1*1\}})}{[c,f]_{\{110\}}\ [c]_{\{1*\}}}\ (\text{on } [c]_{\{1*\}})}{\textit{(empty formula)}}$$

Hence, the formula is satisfiable. Furthermore, by collecting the unit clauses propagated we build a symbolic model for the originating formula, as defined in Section 4.3.

It is interesting to compare what SUCP achieves on $SymbSk(f)$ with the inferences performed on the QBF f via the standard (quantified version of) unit propagation. Every

unit clause in the original formula becomes a symbolic unit after skolemization. Conversely, clauses mentioning exactly one existential literal and at least one universal literal are non-unit in the QBF sense, but they become units after symbolic skolemization.

For example, while the QBFs (1) and (18) contain no unit clause, their skolemized versions mention two and five symbolic units respectively. So, while quantified UCP infers nothing on those QBFs, SUCP achieves substantial results on their skolemized versions, as exemplified by the formula we just decided to be satisfiable via SUCP. That formula is (after a few irrelevant subsumption steps) the symbolic skolemization of (18).

Definition 5.10 (Symbolic Pure Literal Elimination) *Given a formula $\mathcal{F} = \langle \mathcal{Q}, \Pi \rangle$, with $\mathcal{Q} = \langle V, \delta \rangle$ and any literal l with $var(l) \in V$, we pose*

$$\mathcal{I}^+(l) = \bigcup_{\Gamma_{\mathcal{I}} \in \Pi:\, \Gamma \ni l} \mathcal{I}|_{\delta(l)} \qquad\qquad \mathcal{I}^-(l) = \bigcup_{\Gamma_{\mathcal{I}} \in \Pi:\, \Gamma \ni \neg l} \mathcal{I}|_{\delta(l)}$$

and we define the symbolic exclusion of the pure literals on l *as the transformation $SE(\mathcal{F}, l) \doteq [l]_{\mathcal{P}(l)} * \mathcal{F}$, where $\mathcal{P}(l) = \mathcal{I}^+(l) \setminus (\mathcal{I}^+(l) \cap \mathcal{I}^-(l))$. We call* symbolic pure literal elimination *(SPLE) the application of SE repeated until no choice for l produces changes anymore.*

Once again, this symbolic rule is best understood from the ground perspective of $Pexp(\mathcal{F})$. Plenty of pure literals possibly belong to such propositional expansion. Satisfiability is unaffected by the elimination of clauses containing pure literals, even if logical equivalence may be lost because of possible extra-models which contradict those literals. The selection and elimination of all the clauses with pure literals is performed symbolically by SPLE.

Theorem 5.6 *SPLE preserves satisfiability.*

SPLE shares many properties with SUCP (apart from not preserving all models). It can incrementally generate new pure literals to be eliminated, it is a confluent operation, and it is not refutationally complete (actually, it can prove no inconsistency).

As for SUCP, the SPLE rule is more powerful than the homologous QBF pure literal rule (should the SQPL formula come directly from a QBF). Pure literals in a propositional expansion are indeed not necessarily associated to pure literals in the originating QBF, so they cannot be recognized at the QBF level. In addition, it is not necessary for a pure literal to appear that its variable is only mentioned in one polarity. It is from the interplay between literals and scenarios in symbolic clauses that ground pure literals originate.

As an example, let us consider again (23), which comes through symbolic skolemization from a formula with no pure literals (18). The first (symbolic) pure literals we compute are $[c]_{\mathcal{P}(c)} = [c]_{\{1*\}}$ and $[\neg c]_{\mathcal{P}(\neg c)} = [\neg c]_{\{0*\}}$, after whose assignment we remain with

$$[e, f]_{\{001,100\}} \wedge [e]_{\{1*1,000\}} \wedge [\neg e, \neg f]_{\{1*1\}} \wedge [\neg e]_{\{1*0,0*1\}}$$

Next, pure literals $[f]_{\{001,100\}}$ and $[\neg f]_{\{1*1\}}$ are computed. They assignment brings us to

$$[e]_{\{1*1,000\}} \wedge [\neg e]_{\{1*0,0*1\}}$$

where all the literals are pure, hence the formula becomes empty in one more SE step.

5.5 Binary reasoning

It is well-known that 2-SAT (i.e. SAT under the restriction that all the clauses contains two literals) can be decided in linear time, for example by extracting all the *strongly connected components* (SCC) of the *implication graph* (IG) associated with the formula, and checking that no such component includes complementary literals (Aspvall, Plass, & Tarjan, 1979).

The techniques used to decide 2-SAT are helpful even if the formula is not in 2-SAT, provided some significant portion of binary clauses is present anyway. Two classical examples are the use of the IG to derive *equivalent literals* (used to simplify the formula through literal substitution), and to identify *failed literals* (equivalent to implied unit clauses).

In this section, we extend to SQPL some of these techniques.

Definition 5.11 (SIG) *Given a symbolic formula* $\mathcal{F} = \exists[e_1]^{\delta_1} \dots \exists[e_m]^{\delta_m}.\widetilde{\mathcal{F}}$*, the* symbolic implication graph $SIG(\mathcal{F})$ *constructed over* $\mathcal{F}$ *is a directed graph with* $2m$ *nodes, labeled by* $[e_1]^{\delta_1}$, $[\neg e_1]^{\delta_1}$, $\dots$, $[e_m]^{\delta_m}$, $[\neg e_m]^{\delta_m}$*. For each* symbolic binary clause $[l_i, l_j]_{\mathcal{I}} \in \widetilde{\mathcal{F}}$ *the graph contains two edges labeled by the scenario* $\mathcal{I}$*, one from* $[\neg l_i]^{\delta_i}$ *to* $[l_j]^{\delta_j}$*, the other from* $[\neg l_j]^{\delta_j}$ *to* $[l_i]^{\delta_i}$*. We note these labeled edges as* $[\neg l_j]^{\delta_j} \xrightarrow{\mathcal{I}} [l_i]^{\delta_i}$ *and* $[\neg l_i]^{\delta_i} \xrightarrow{\mathcal{I}} [l_j]^{\delta_j}$*.*

For example, the SIG associated with the formula

$$\exists[a]^2\exists[b]^2\exists[c]^2\exists[d]^2. \quad \begin{array}{l}[\neg c, \neg d]_{\{*0\}} \wedge [a, c]_{\{0*\}} \wedge [\neg a, \neg b]_{\{00,*1\}} \wedge \\ \wedge\, [b, d]_{\{*0\}} \wedge\, [b, c]_{\{**\}} \wedge [a, \neg c]_{\{*1,10\}}\end{array} \tag{27}$$

is depicted in Figure 3. This graph contains a single strongly connected component, which includes all the nodes. As opposed to the standard 2-SAT case, the presence of complementary literals in such component is not a sufficient criterion to infer unsatisfiability (the above formula is in fact satisfiable), yet the opposite implication still holds.

Lemma 5.3 *If no SCC in the SIG associated to a 2-SQPL formula* $\mathcal{F}$ *contains complementary literals, then* $\mathcal{F}$ *is satisfiable.*

As usual, the scenarios attached to edges are recognized by a forest of BDDs, so a SIG is a couple of graphs plus a mapping from edges in the first graph to nodes in the second one.

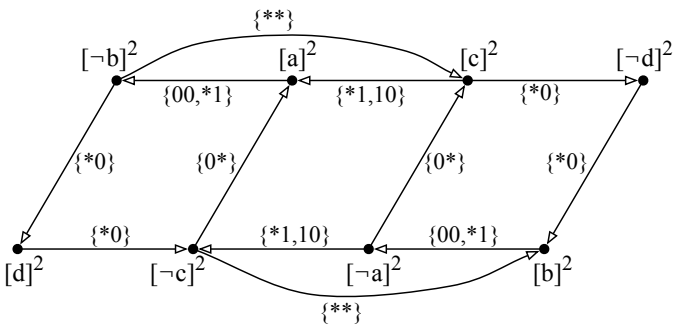


Figure 3: Symbolic implication graph associated with the 2-SQPL formula (27).

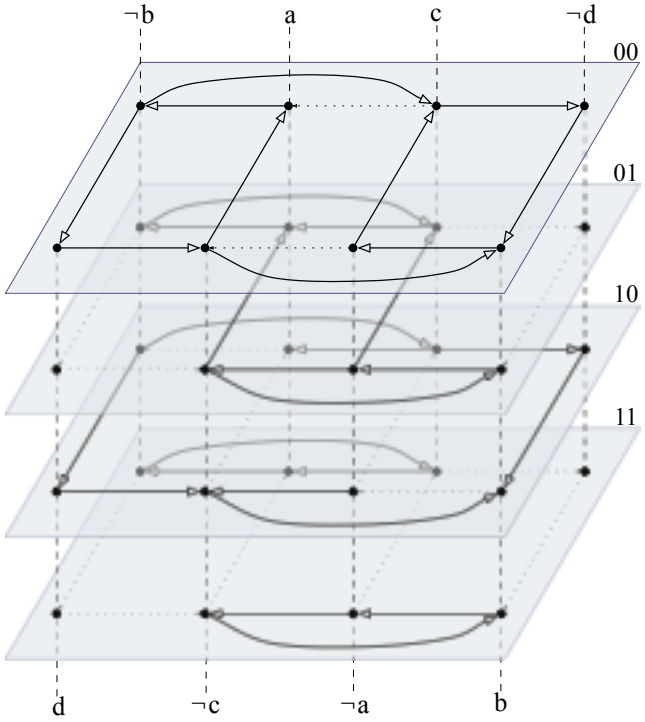


Figure 4: The IG for the expansion of (27), obtained by unfolding the SIG in Figure 3.

A SIG can be viewed as a compressed representations of the implication graph constructed over the propositional expansion of the formula. The expanded graph is depicted in Figure 4 for our sample case. The goal is to design a binary reasoning mechanism which works at the symbolic level, without resorting to the expanded implication graph. We start by extending to SQPL the notions of variable substitution and literal equivalence.

Definition 5.12 (Symbolic Variable Substitution) *A* symbolic variable substitution *for a formula $\langle \mathcal{Q}, \Pi \rangle$, with $\mathcal{Q} = \langle V, \delta \rangle$, is a quadruple $\langle v, w, \phi, \mathcal{J} \rangle$ where $v \in V$ is the* substituting *variable, $w \in V$, $\delta(v) \leq \delta(w)$ is the* substituted *variable, $\phi \in \mathbb{B}$ is the* phase *of the substitution, and $\mathcal{J} \subseteq \mathbb{B}^{\gamma(w)}$ is the* scenario *of the substitution. We denote a substitution $\langle v, w, \phi, \mathcal{J} \rangle$ by writing $[v/l]_{\mathcal{J}}$, with $l = \phi \oplus w$. The result of applying a substitution $[v/l]_{\mathcal{J}}$ to a clause $C = \Gamma_{\mathcal{I}} \in \Pi$, written $C[v/l]_{\mathcal{J}}$, is defined as follows:*

$$\Gamma_{\mathcal{I}}[v/l]_{\mathcal{J}} \doteq \begin{cases} \{\Gamma_{\mathcal{I}}\} & \text{if } v \notin var(\Gamma),\ var(l) \notin var(\Gamma) \\ \{\Gamma'_{\mathcal{I}'}, \Gamma_{\mathcal{I} \cap \overline{\mathcal{J}}}\} & \text{if } v \notin var(\Gamma),\ \phi \oplus l \in \Gamma \text{ for some } \phi \in \mathbb{B} \\ \{\Gamma_{\mathcal{I} \cap \overline{\mathcal{J}}}\} & \text{if } \phi \oplus v \in \Gamma, \overline{\phi} \oplus l \in \Gamma \text{ for some } \phi \in \mathbb{B} \\ \{\Gamma'_{\mathcal{I}'}, \Gamma_{\mathcal{I} \cap \overline{\mathcal{J}}}\} & \text{if } \phi \oplus v \in \Gamma, \phi \oplus l \in \Gamma \text{ for some } \phi \in \mathbb{B} \end{cases}$$

where (second and fourth cases) $\Gamma' = \Gamma \setminus \{\phi \oplus l\} \cup \{\phi \oplus v\}$ and $\mathcal{I}' = (\mathcal{I} \cap \mathcal{J})|_{\gamma(\Gamma')}$. The result $\mathcal{F}[v/l]_{\mathcal{J}}$ of applying $[v/l]_{\mathcal{J}}$ to $\mathcal{F} = \langle \mathcal{Q}, \Pi \rangle$ is the formula $\langle \mathcal{Q}, \Pi' \rangle$ where $\Pi' = \cup_{C \in \Pi} C[v/l]_{\mathcal{J}}$.

Let us consider, for example, how the matrixes of two sample formulas $\mathcal{F}$ and $\mathcal{F}'$ over the quantification structure $\exists[a]^1\exists[b]^2\exists[c]^3$ relate to each other if $\mathcal{F}' = \mathcal{F}[c/a]_{\{1*0\}}$. The four cases in the above definition are respectively exemplified by the four clauses in $\mathcal{F}$:

$$\begin{aligned}\widetilde{\mathcal{F}} &= \overbrace{[b]_{\{1*\}}} \wedge \overbrace{[b,\neg c]_{\{*1*\}}} \wedge \overbrace{[\neg a,\neg b,c]_{\{**0\}}} \wedge \overbrace{[a,\neg b,c]_{\{***\}}}\\ \widetilde{\mathcal{F}'} &= \overbrace{[b]_{\{1*\}}} \wedge \overbrace{[\neg a,b]_{\{11\}} \wedge [b,\neg c]_{\{01*,111\}}} \wedge \overbrace{[\neg a,\neg b,c]_{\{0*0\}}} \wedge \overbrace{[a,\neg b,c]_{\{0*1,1*1,0*0\}} \wedge [a,\neg b]_{\{1*\}}}\end{aligned}$$

The symbolic substitutions we are interested in are those we obtain from literal equivalences.

Definition 5.13 (equivalent literals) *A literal equivalence over a quantification structure $\mathcal{Q} = \langle V, \delta\rangle$ is a set of literals $L = \{l_1, l_2, \ldots, l_m\}$ over V coupled with a scenario $\mathcal{I} \subseteq \mathbb{B}^{\delta(L)}$. It is noted $[l_1 \equiv l_2 \equiv \cdots \equiv l_m]_{\mathcal{I}}$. An equivalence over $\mathcal{Q}$ is valid for $\mathcal{F} = \mathcal{Q}.\widetilde{\mathcal{F}}$ if*

$$\forall M \in \mathcal{M}(\mathcal{F})\ \forall p, q \in L : \delta(p) \leq \delta(q)\ \forall \mathcal{J} \subseteq \mathcal{I}|_{\delta(q)}. \quad [p]_{\mathcal{J}} \in M \ \ \textit{iff} \ \ [q]_{\mathcal{K}} \in M$$

where $\mathcal{K} = (\mathcal{I}|_{\delta(q)} \cap \mathcal{J})|_{\delta(p)}$.

Lemma 5.4 *If the equivalence $[l_1 \equiv l_2 \equiv \cdots \equiv l_m]_{\mathcal{I}}$ is valid for $\mathcal{F}$, then for all $p, q \in \{l_1, \ldots, l_m\}$ such that $\delta(p) \leq \delta(q)$ the formula $\mathcal{F}[q/p]_{\mathcal{I}|_{\delta(p)}}$ is satisfiability-equivalent to $\mathcal{F}$.*

The identification of some valid equivalences can be performed on the SIG, and the same is true of *implied units*, i.e. symbolic unit clauses $[l]_{\mathcal{I}}$ such that $[\neg l]_{\mathcal{I}} * \mathcal{F}$ is unsatisfiable. Let us call *flat* a path which only traverses literals at the same universal depth.

Definition 5.14 (continuous scenario) *Given a flat path $\pi = [l_1]^{\delta_l} \xrightarrow{\mathcal{I}_1} [l_2]^{\delta_l} \cdots \xrightarrow{\mathcal{I}_m} [l_{m+1}]^{\delta_l}$ in a SIG, the* continuous scenario *associated to π is defined as $\mathcal{I}(\pi) \doteq \cap_{j=1,\ldots,m}\mathcal{I}_j$.*

Lemma 5.5 *Let G be the SIG for the symbolic formula $\mathcal{F}$. Then*

- *Implied unit: if π is a flat path from $[l]^{\delta_l}$ to $[\neg l]^{\delta_l}$ in G with a non-empty continuous scenario, then $[\neg l]_{\mathcal{I}(\pi)}$ is an implied unit for $\mathcal{F}$;*
- *Valid equivalence: if π is a flat cycle in G traversing $[l_0]^{\delta_l}, [l_1]^{\delta_l}, \ldots, [l_m]^{\delta_l}$ then $[l_1 \equiv l_2 \equiv \cdots \equiv l_m]_{\mathcal{I}(\pi)}$ is a valid equivalence for $\mathcal{F}$.*

For example, the path $[\neg c]^2 \xrightarrow{\{0*\}} [a]^2 \xrightarrow{\{00,1*\}} [\neg b]^2 \xrightarrow{\{**\}} [c]^2$ in Figure 3 allows us to derive and propagate the symbolic unit $[c]_{\{0*\}}$, consistently with the two ground implied literals c_{00} and c_{01} we find by analyzing the IG in Figure 4. At the same time, the cycles $\pi_1 = [a]^2 \xrightarrow{\{00,*1\}} [\neg b]^2 \xrightarrow{\{**\}} [c]^2 \xrightarrow{\{*1,10\}} [a]^2$ and $\pi_2 = [a]^2 \xrightarrow{\{00,*1\}} [\neg b]^2 \xrightarrow{\{*0\}} [d]^2 \xrightarrow{\{*0\}} [\neg c]^2 \xrightarrow{\{0*\}} [a]^2$ allows us to conclude that $[a \equiv \neg b \equiv c]_{\{*1\}}$ and $[a \equiv \neg b \equiv d \equiv \neg c]_{\{00\}}$ are valid equivalences.

To make the above mechanism refutationally complete for 2-SQBF (hence, for 2ex-QBF by virtue of symbolic skolemization), we need to reason on formulas whose SIGs include non-flat paths. While the notions of equivalence and substitution are general enough, the definition of continuos scenario associated to non-flat paths is substantially more complicated. The major obstacle is, intuitively, that the plans representing slices of scenarios, such as those in Figure 4, are not parallel. Rather, they collapse on each other forming a tree-like structure shaped by the prefix. The computation of continuous scenarios over similar structures demands involved details, which will be presented in a forthcoming paper. The key idea stays the same, i.e. equivalences and failed literals are obtained by performing BDD-based operations on the SIG, without decompressing its meaning.

Algorithm 1: SymbDecide($\mathcal{F}$)

input : A symbolic formula $\mathcal{F} = \mathcal{Q}.\ \widetilde{\mathcal{F}}$, where $\mathcal{Q} = \langle V, \delta \rangle$
output: The evaluation of $\mathcal{F}$ into $\{\textsf{sat}, \textsf{unsat}\}$

```
1  F ← simplify(F); F' ← F;
2  while continueElimination(F, F') do
3      if [] ∈ F̃ then
4          return unsat;
5      else if F̃ = ∅ then
6          return sat;
7      else if expandable(F) then
8          F_exp ← Pexp(F);
9          return SAT(F_exp);
10     else
11         e ← a variable in V with δ(e) = δ(F);
12         F ← simplify(SVE(F, e));
13         if |F| < |F'| then
14             F' ← F;
15 return SymbSplit(F');
```

6. A decision procedure for SQPL

We propose a complete decision procedure for SQPL which attempts to fruitfully combine the reasoning tools developed in the previous section. Beyond merging different decision techniques, we narrow the degrees of freedom stemming from their non-deterministic nature. We are also concerned with practical computational issues, such as the space requirements of expansion compared to the amount of physically available memory.

Overall structure. The entry point for our procedure is Algorithm 1, which implements an inference strategy based on the symbolic elimination of variables. Despite being complete in principle, the elimination of variables is interrupted if the condition recognized by *continueElimination* (discussed later) fails before the formula is decided. In such case, we resort to Algorithm 2, which implements and alternative complete strategy based on search.

The overall decision procedure thus consists in a space-intensive resolution-based approach, used as a preprocessing step for a search-based procedure. This particular arrangement has been selected as it produces the best experimental performance on a wide variety of cases (see Section 7), and it is consistent with the findings reported in the literature on the combination of resolution and search in propositional settings (see Section 8).

Algorithm 1. Variable elimination in the spirit of (Davis & Putnam, 1960; Dechter & Rish, 1994; Biere, 2004) is adapted to SQPL. An essential parameter of this technique is the method used to decide the order of elimination (line 11). Rather than computing some global (possibly optimal) ordering as in (Rish & Dechter, 2000), we schedule variables in a greedy way. The best variable to eliminate next is heuristically assumed to be the one with the minimum elimination cost. As in (Biere, 2004), the cost of a variable e is defined as the

increment of the number of literals generated by its elimination, i.e. $|SVE(\mathcal{F}, e)| - |\mathcal{F}|$. If variables associated to negative costs exist, they are chosen first, and the formula is shrunk.

The number of literals may strongly fluctuate as SVE goes on. Thus, the variable $\mathcal{F}'$ is introduced to record the smallest formula obtained by Algorithm 1 during its life span (lines 13-14). If *continueElimination* fails, $\mathcal{F}'$ is passed to Algorithm 2 as a replacement for the formula with the smallest number of variables (line 15).

This option has two advantages: relieves Algorithm 2 of redundant clauses and allows us to modulate the effort devoted to variable elimination. A fairly simple heuristics we can apply to limit resolution is to give up as soon as no variable choice shrinks the formula, in the spirit of (Subbarayan & Pradhan, 2004; Eén & Biere, 2005). This option makes it unnecessary to keep track of the smallest formula encountered. We apply a more permissive condition by granting variable elimination the possibility to continue working even if intermediate results do not shrink monotonically. Only when a certain threshold on the length of the enlarging elimination subsequence is exceeded we turn back to the global size minimum (which lays at an arbitrary position in such sequence) and switch to search.

The resulting criterion applied by *continueElimination* is as follows: *return false if* $|\mathcal{F}'|$ *is different from* $|\mathcal{F}|$ *since more than* k_{res} *rounds.* A second circumstance under which variable elimination is interrupted exists, depending on the specific computational infrastructure: *return false if the memory required to store both* $\mathcal{F}$ *and* $\mathcal{F}'$ *exceeds the available main memory.* The reason for this empirical rule is discussed in the "SAT-based reasoning" paragraph. In all the other cases, variable elimination is prosecuted.

Algorithm 2: SymbSplit($\mathcal{F}$)

input : A symbolic formula $\mathcal{F} = \mathcal{Q}.\ \widetilde{\mathcal{F}}$, where $\mathcal{Q} = \langle V, \delta \rangle$
output: The evaluation of $\mathcal{F}$ into $\{\mathsf{sat}, \mathsf{unsat}\}$

16 $\mathcal{F} \leftarrow simplify(\mathcal{F})$;
17 **if** $[\,] \in \mathcal{F}$ **then**
18 **return unsat**;
19 **else if** $\mathcal{F} = \emptyset$ **then**
20 **return sat**;
21 **else if** *expandable*($\mathcal{F}$) **then**
22 $F_{exp} \leftarrow Pexp(\mathcal{F})$;
23 **return** SAT(F_{exp});
24 **else**
25 **if** $\mathcal{F}$ *is an* α*-formula* **then**
26 $i \leftarrow$ a positive integer smaller than $\delta_{min}(\mathcal{F})$;
27 **if** SymbSplit($\mathcal{F}{\downarrow_0^i}$)=**sat** *and* SymbSplit($\mathcal{F}{\downarrow_1^i}$)=**sat then**
28 **return sat**
29 **else**
30 $l \leftarrow$ a literal over V such that $\delta(l) = 0$
31 **if** SymbSplit($[l] * \mathcal{F}$)=**sat** *or* SymbSplit($[\neg l] * \mathcal{F}$)=**sat then**
32 **return sat**
33 **return unsat**;

Algorithm 2. A recursive DLL-like decision procedure based on Theorem 5.4 is adapted to SQPL, in the spirit of (Cadoli et al., 1998). The base cases of the recursion are possibly decided using propositional expansion and a SAT solver as look-ahead tools (lines 22-23).

The split of α-formulas (line 27) is performed symbolically and it only involves BDD manipulations. Contrariwise, β-formulas (line 31) are split by a symbolic assignment which leaves BDDs unchanged as the universal depth of the assigned literal is zero.

Algorithm 2 implements a basic DP-like search scheme. Sophisticated variants have been developed in the literature, and they can be smoothly adapted to our framework. For example, conflict-directed and solution-directed *backjumping* can be used to replace chronological backtracking (Giunchiglia, Narizzano, & Tacchella, 2001b), and learning of models and implied clauses can be adopted (Letz, 2002; Zhang & Malik, 2002).

Simplification. Both algorithms apply the procedure "*simplify*" before attempting anything else (lines 1 and 16). Such procedure normalizes the formula (Section 5.2), propagates symbolic unit clauses and eliminate pure literals (Section 5.4), and performs binary reasoning (Section 5.5). All these operations are backtrack-free inferences aiming to reduce the size of the ground expansion of the formula, according to the following result.

Lemma 6.1 *If $\mathcal{F}'$ is obtained from $\mathcal{F}$ by applying an assignment, a substitution, or a normalization, then $|\mathcal{F}'|_{grn} \leq |\mathcal{F}|_{grn}$.*

Simplification is pursued with the highest priority as incomplete rules yield in experiments the best tradeoff between computational effort and inference power.

As opposed to the usual propositional framework, the effort required to reach the fixpoint of simplification may be significant. It becomes relevant to properly schedule the relative order of simplification rules, and to regulate the instantiation of each rule (e.g. the order of propagation for symbolic unit clauses). The best inference policy strongly depends on the formula at hand, so we employ an adaptive round-robin scheme with preemptive priorities and variable time-windows, fully described in (Benedetti, 2005d).

Simplification may enlarge the size of the symbolic representation in favor of a smaller footprint for its expansion, as exemplified by the substitution on page 28, where the symbolic size is enlarged from 4 to 6, while the ground size is reduced from 18 to 15. Such tradeoff is accepted as the overall evaluation procedure is ultimately oriented to promote the application of SAT-based reasoning through the reduction of the ground size.

SAT-based reasoning. Both algorithms are designed to check the expandibility of simplified SQPL formulas into equivalent SAT instances as often as possible. While Algorithm 1 resorts to SAT at most once, Algorithm 2 may incur expansion repeatedly, up to one for each search branch traversed. In both cases, expansion is a three-step process:

1. *Check for expandibility (lines 7, 21).* Any SQPL formula can in principle be expanded to SAT, but we need to fix reasonable limits to the maximum allowed size of the expansion. State-of-the-art SAT procedures allocate memory to maintain and manipulate original and learned clauses in the form of *watched data structures*. Given the modus operandi of such structures, it is reasonable to assume that the solving engine keeps on referring to the whole allocated memory. This means that no part of it can be swapped to secondary memory without incurring I/O trashing conditions.

So, memory requirements are modelized using the simple expression $\beta + (\alpha + \gamma) \cdot s$, where s is the size of the input formula (number of clauses), β is a parameter (MB) that estimates the formula-independent amount of memory required, α is a parameter (MB/clause) that approximates the linear component of the dependence, and γ is used to modelize the enlargement due to the accumulation of learnt clauses. We pose

$$expandable(\mathcal{F}) \doteq \frac{M_{free}}{\beta + (\alpha + \gamma) \cdot |\mathcal{F}|_{grn}} > 1$$

where M_{free} is the amount of available central memory that can be allocated (and continuously referred to) without incurring in trashing conditions.

Expandability is checked at each iteration of Algorithm 1 and at each recursion of Algorithm 2, so its evaluation—which essentially consists in calculating $|\mathcal{F}|_{grn}$—is assumed not to constitute a bottleneck. Lemma 4.3 ensures that such computation is linear in the size of the formula. We employ two additional techniques to amortize the estimation cost: (i) Only the clauses that have been introduced or modified since the last estimation need an update. So, $|\mathcal{F}|_{grn}$ can be computed incrementally on the basis of its previous known value. Furthermore, (ii) the up-to-date size estimations, tough incomplete, may induce a lower bound on $|\mathcal{F}|_{grn}$ which is sufficient to falsify the expandability condition. In this case, no computation at all is required.

2. *Propositional expansion (lines 8, 22).* Two steps are involved: (i) the generation of a namespace for ground variables; (ii) the generation of the ground clauses over such space. Variables are created by mapping the structured namespace of symbolic literals onto a flat, SAT-solver friendly namespace for ground literals, through a function:

$$V_{map} : V_\exists \times \mathbb{B}^{|V_\forall|} \to [1, n]$$

To prevent the SAT solver from suffering unnecessarily large data structures, the co-domain of V_{map} is the smallest interval $[1, n]$ to allow a bijection. In particular, rather than $n = \sum_{e \in V_\exists} 2^{|dom(e)|}$, we pose $n = \sum_{e \in V_\exists} |\mathcal{I}(e)|$, with $\mathcal{I}(e) = \{i \in \mathcal{I} | \Gamma_\mathcal{I} \in \Pi, e \in var(\Gamma)\}$, and we don't define the mapping on indexes that never appear in the expansion of the matrix (though they belong to the expansion of the prefix).

A data structure that allows to efficiently query V_{map} is crucial to make the expansion practical. Indeed, each single expansion of a symbolic formula (which is expected to nearly fill the main memory, and can be requested several times) will repeatedly query V_{map} on every point of its domain (whose size may easily reach the millions). In practice, V_{map} is queried tens of millions of times during the solution of large instances. An efficient computation of such function is obtained through the precomputation of a list of hash tables, one for each variable in $V_\exists$.

3. *SAT solving (lines 9, 23).* The "SAT" procedure is meant to work in a black-box way, by engaging an external state-of-the-art SAT solver, such as (Moskewicz et al., 2001; Een & Sorensson, 2003). This architecture allows to profit transparently from improvements in SAT technologies. In the basic procedure we present, no information (other than a **sat**/**unsat** answer) flows through the interface between the main recursive search and the nested calls to the SAT solver. A tighter interaction along the guidelines described in (Samulowitz & Bacchus, 2005) can be achieved.

7. Experimental evaluation

We present some experiments in which QBF instances are decided by (i) symbolically skolemizating them, and (ii) applying Algorithm 1 to the resulting SQPL formula. We are interested in measuring how many instances such method can solve compared to other state-of-the-art QBF provers, given identical computational resources (time and memory). The benchmark set is constituted as explained in Section 7.1. Competing QBF solvers are introduced in Section 7.2. The experimental results are commented in Section 7.3.

7.1 Benchmark set

We aim to assemble a benchmark set representative of the variety of real-world applications and reasoning tasks the QBF language is applied to. For this reason, we exclude instances generated according to random syntactic models, and focus on applications. Our test set is of large but not overwhelming size, and it takes comprehensively account of applications as different as model checking, automated synthesis, automated diagnosis, non-monothonic reasoning, equivalence checking, planning, two-player games, etc.

A total of 1,567 instances (contributed by 18 different research groups and arranged into 128 families) have been gathered from public domain sources, or requested to the respective authors. The substantial differences in the encoded tasks (and in the encoding techniques) result in a rich assortment of cases, and in a variety of syntactic structures. In particular[7]:

Ansotegui-Selman's instances (46 instances, 7 families) encode a two-player game, namely the evader-pursuer problem on fixed size checkerboards (Ansótegui, Gomes, & Selman, 2005). There are from 2 ($\exists\forall\exists$) to 18 alternations in the prefix.

Ayari's benchmarks (72 instances, 5 families) come from real-world verification problems on circuits (*adder*, *DFlipFlop*, and *VonN*) and protocol descriptions (*MutexP* and *SzymanskyP*). Encodings with fixed alternations (from $\forall\exists$ to $\exists\forall\exists\forall\exists\forall\exists$) are used (Ayari & Basin, 2000). The biggest formula mentions more than one million variables.

Biere's benchmarks (64 instances, 4 families) represent model checking problems stating invalid safety properties over n-bit counters, with $n = 1, \ldots, 16$. The encoding uses an iterative-squaring method in which the complexity of the prefix grows with the depth of analysis (Biere, 2004). Up to 32 alternations are involved.

Castellini's instances (169 instances, 3 families) encode *conformant* variants of a planning problem, called "bomb-in-the-toilet", where non-determinism in the initial state is handled by universal quantification (Castellini, Giunchiglia, & Tacchella, 2003).

Faber-Leone-Maratea-Ricca's instances (100 instances, 5 families) express a strategic problem (with $\forall\exists$ alternation) in which subset-minimal production-preserving clusters of companies (called *strategic* sets) are determined (Cadoli, Eiter, & Gottlob, 1997).

Gent-Rowley's problems (59 instances, 8 families) encode the existence of a winning strategy in variants of the "Connect4" game (Gent & Rowley, 2003).

Herbstritt's family (28 instances) is concerned with bounded model checking of partially specified circuit designs. 6 to 8 alternations are used (Herbstritt & Becker, 2007).

7. For simplicity, the benchmarks are named after the author(s) who made available the related QBF encoding, even if the underlying problem was originally encoded/proposed by further/different authors.

Katz's benchmarks (16 instances, 2 families) capture the (symbolic) reachability problem for some hardware circuits using both fixed and increasing alternation encodings.

Lairi-Seshia's family (3 instances) encodes convergence checking problems.

Ling's benchmarks (2 families, 8 instances) encode FPGA (Field Programmable Gate Array) logic synthesis problems. The aim is to determine whether a specific logic function can be implemented in a given programmable circuit (Ling et al., 2005).

Mangassarian-Veneris' benchmarks (12 families, 132 instances) encode bounded model checking problems over real-word circuits, using non-classical fixed-alternation encoding techniques (Mangassarian, Veneris, Safarpour, Benedetti, & Smith, 2007).

Mneimneh and Sakallah's benchmarks (12 families, 90 instances) encode the problem of computing the diameter (or sequential depth) for twelve of the ISCAS89 circuits (Mneimneh & Sakallah, 2003). For each circuit, a sequence of $\exists\forall\exists$ instances is generated, the n-th of which checks if that circuit has a new state at depth n.

Narizzano's problems (4 families, 32 instances) encode a robot navigation problem described in (Castellini et al., 2003).

Pan's benchmark (19 families, 378 instances) is mainly comprised (18 families) of encodings of modal-K formulas satisfiability (Pan & Vardi, 2003). The last family (*q-shifter*) encodes the existence of a suited output configuration for any input to a barrel-shifter with n control bits and 2^n input lines, using a single $\forall\exists$ alternation.

Remshagen-Truemper's benchmark (12 families, 144 instances) encodes the existence of a winning strategy in a two-player game using an alternative technique that does not introduce one quantifier alternation per game ply (Remshagen & Truemper, 2005).

Rintanen's benchmarks (17 families, 131 instances) include several conformant/conditional planning, hand-made, and sorting network problems (Rintanen, 1999b).

Scholl-Becker's benchmarks (8 families, 64 instances) encode formal equivalence checking of partial implementations of real-world designs in which random faults have been inserted (Scholl & Becker, 2000).

Palacios-Geffner's problems (5 families, 32 instances) encode planning and navigation tasks with unknown initial state or disallowed sensing (Palacios & Geffner, 2005).

Most of these families are quite challenging for modern solvers, and many include instances have never been solved (Le Berre, Narizzano, Simon, & Tacchella, 2004; Narizzano, Pulina, & Tacchella, 2006). The complete test set is made available at (Benedetti, 2005e).

7.2 Solvers

We experiment with the following implementations of QBF decision procedures:

QuBE (Giunchiglia, Narizzano, & Tacchella, 2001c), version 1.3-LRN: a search-based solver featuring watched data structures for unit clause propagation and pure literal detection, plus backjumping and conflict/solution learning.

Semprop (Letz, 2002), version 2004-01-06, a search-based solver which includes dependency-directed backjumping and mechanisms to cache lemmas/models.

	sKizzo	quantor	QuBE	yquaffle	semprop	SQBF
total	**1206** (**77**%)	1037 (66%)	817 (52%)	835 (53%)	915 (58%)	802 (51%)
best	**503** (**38**%)	358 (27%)	136 (10%)	94 (7%)	166 (13%)	57 (4%)
unique	**108** (**63**%)	15 (9%)	5 (3%)	15 (9%)	24 (14%)	4 (2%)

Table 1: Number of instances solved by different QBF solvers (under the experimental conditions described in the text). The row labeled "total" gives the number of instances solved within the allotted timeout (in absolute value and as percentage of the total number of instances). The number of cases in which a solver answered first is accounted for in the "best" row (absolute value and percentage of the instances solved by at least one solver). Similarly, the "unique" row reports on cases that were only decided by the considered solver (absolute value and percentage of the instances only decided by one solver).

yQuaffle (Zhang & Malik, 2002), version 2006-02-10, a search-based solver featuring multiple conflict-driven learning, inversion of quantifiers and solution-based backjumping.

Quantor (Biere, 2004), version 2.11, a resolution-based solver employing q-resolution and expansion to eliminate quantifiers, subsumption control, a careful variable-elimination schedule, and a SAT solver as a back-end to solve the final existential theory.

SQBF (Samulowitz & Bacchus, 2005), version 2006-12-18, a solver based on a bi-directional cooperation with a SAT solver (to exchange learnt clauses, etc.), achieved through a tight integration at the data-structure level.

These solvers have been selected as they are publicly available implementations of state-of-the-art techniques yielding the best results in recent evaluations (Le Berre et al., 2004; Narizzano et al., 2006), and they cover a large spectrum of different approaches to QBF. More details on the key differences among these solvers are given in Section 8.

The following incarnation of the procedure described in Section 6 is employed:

sKizzo (Benedetti, 2005d), version 0.10, available at (Benedetti, 2005e).

7.3 Experimental results

We fed the 6 selected solvers with all the instances in our test set, allowing up to 1,000 seconds for each of the resulting runs[8]. Table 1 presents a synthetic overview of the results. The approach based on symbolic skolemization appears to solve significantly more instances than any other method. If we limit our attention to instances solved by at least one solver (second row) we observe that symbolic skolemization is the fastest technique more often than any other method. The number of instances uniquely solved by symbolic skolemization (third row) testifies to a clear contribution in improving the state of the art in the field. It exceeds by more than four times the result of the second best solver (semprop).

8. A cluster of 8 identical machines has been employed for the experiments. Each node is equipped with a 2.66GHz Intel Xeon processor and 2GB of main memory, and runs Gentoo Linux 2.6.19. The full traces of all the experiments are made available at (Benedetti, 2005e).

The pace at which these results are obtained as a function of the timeout is depicted in Figure 5. This information is represented graphically as described in (Giunchiglia, Maratea, Tacchella, & Zambonin, 2001a): The Y-value of a point in the plot gives the number of instances that can be solved within an amount of time represented by the X-value of the same point (given on a logarithmic scale). The symbolic skolemization method dominates all the other techniques in terms of number of instances solved within any given timeout. The second-best solver is quantor, which dominates, in turn, all the other approaches.

To assess whether our technique is contributing in a balanced way across different types of problems, we present a breakdown of the data on a per-benchmark basis (Table 2). We preliminarily observe that only 3 of the 18 benchmarks have been completely solved by any specific solver, and this confirms that the test-set is challenging for the state of the art.

sKizzo is the best solver on 12 of the 18 benchmarks, both in terms of total number of solved instances and in terms of instances solved quicker. It is the second-best solver as to solved instances in all but one of the remaining cases. Thus, it can be credited with a robust cross-benchmark performance.

We note that in 11 benchmarks the method based on symbolic skolemization decides at least one instance that no other technique can solve within the same time bound. The improvement is conspicuous in at least 6 benchmarks. This proves that the contribution to the state of the art ranges across different classes of problems. Some results at the family level are presented in Table 3. These families have been selected to evidence cases in which symbolic skolemization clearly improves over all the other approaches we tested.

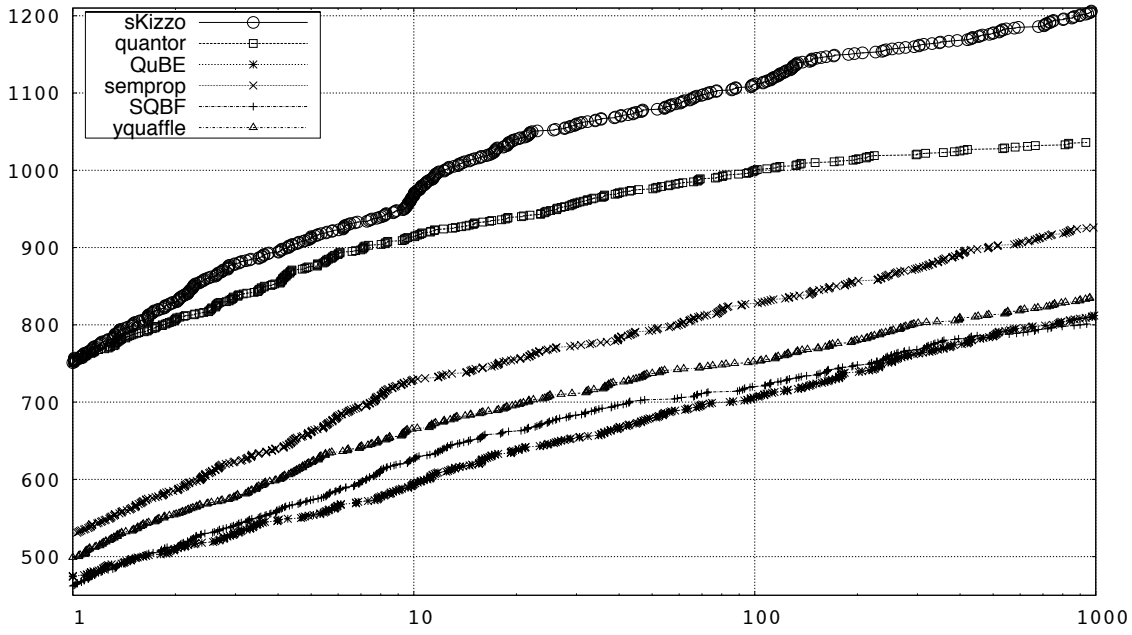


Figure 5: Performance of several QBF solvers over our test set (1,567 instances). The number of instances solved (y axis) is plotted against the allotted timeout in seconds (x axis).

	sKizzo	quantor	QuBE	yquaffle	semprop	SQBF	
total	**15** (**33**%)	7 (15%)	8 (17%)	8 (17%)	12 (26%)	11 (24%)	ansotegui selman
best	**12** (**55**%)	0 (0%)	0 (0%)	0 (0%)	3 (14%)	7 (32%)	
unique	**3** (**43**%)	0 (0%)	0 (0%)	0 (0%)	1 (14%)	**3** (**43**%)	
total	**41** (**63**%)	32 (49%)	26 (40%)	24 (37%)	28 (43%)	19 (29%)	ayari
best	**19** (**45**%)	4 (10%)	15 (36%)	0 (0%)	4 (10%)	0 (0%)	
unique	**9** (**90**%)	0 (0%)	0 (0%)	0 (0%)	1 (10%)	0 (0%)	
total	**49** (**77**%)	39 (61%)	20 (31%)	29 (45%)	26 (41%)	27 (42%)	biere
best	**31** (**63**%)	12 (24%)	4 (8%)	0 (0%)	1 (2%)	1 (2%)	
unique	**10** (**100**%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	
total	**169** (**100**%)	168 (99%)	167 (99%)	166 (98%)	164 (97%)	156 (92%)	castellini
best	**41** (**24**%)	33 (20%)	38 (22%)	6 (4%)	36 (21%)	15 (9%)	
unique	**1** (**100**%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	
total	77 (77%)	60 (60%)	87 (87%)	85 (85%)	**88** (**88**%)	54 (54%)	faber-leone maratea-ricca
best	12 (13%)	1 (1%)	14 (15%)	**36** (**38**%)	30 (32%)	2 (2%)	
unique	0 (0%)	0 (0%)	**2** (**29**%)	**2** (**29**%)	**2** (**29**%)	1 (14%)	
total	**48** (**81**%)	41 (69%)	32 (54%)	45 (76%)	47 (80%)	43 (73%)	gent-rowley
best	8 (17%)	**19** (**40**%)	18 (38%)	0 (0%)	3 (6%)	0 (0%)	
unique	**1** (**100**%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	
total	**1** (**4**%)	**1** (**4**%)	**1** (**4**%)	0 (0%)	**1** (**4**%)	0 (0%)	herbstritt
best	**1** (**33**%)	**1** (**33**%)	0 (0%)	0 (0%)	**1** (**33**%)	0 (0%)	
unique	**1** (**50**%)	**1** (**50**%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	
total	**4** (**25**%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	katz
best	**4** (**100**%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	
unique	**4** (**100**%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	
total	**8** (**100**%)	**8** (**100**%)	7 (88%)	7 (88%)	5 (62%)	4 (50%)	ling
best	**4** (**50**%)	1 (12%)	1 (12%)	1 (12%)	1 (12%)	0 (0%)	
unique	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	
total	**110** (**83**%)	109 (83%)	72 (55%)	72 (55%)	31 (23%)	72 (55%)	mangassarian veneris
best	38 (33%)	**43** (**38**%)	11 (10%)	11 (10%)	7 (6%)	4 (4%)	
unique	1 (20%)	1 (20%)	**2** (**40**%)	1 (20%)	0 (0%)	0 (0%)	
total	**34** (**34**%)	19 (19%)	10 (10%)	2 (2%)	10 (10%)	1 (1%)	mneimneh sakallah
best	**32** (**73**%)	10 (23%)	0 (0%)	1 (2%)	1 (2%)	0 (0%)	
unique	**24** (**92**%)	1 (4%)	0 (0%)	1 (4%)	0 (0%)	0 (0%)	
total	28 (88%)	10 (31%)	28 (88%)	21 (66%)	16 (50%)	**29** (**91**%)	narizzano
best	**24** (**75**%)	0 (0%)	8 (25%)	0 (0%)	0 (0%)	0 (0%)	
unique	**2** (**67**%)	0 (0%)	1 (33%)	0 (0%)	0 (0%)	0 (0%)	
total	**354** (**94**%)	286 (76%)	148 (39%)	137 (36%)	246 (65%)	152 (40%)	pan
best	141 (40%)	**175** (**49**%)	5 (1%)	6 (2%)	27 (8%)	2 (1%)	
unique	**48** (**98**%)	0 (0%)	0 (0%)	0 (0%)	1 (2%)	0 (0%)	
total	136 (94%)	117 (81%)	129 (90%)	134 (93%)	132 (92%)	**144** (**100**%)	remshagen truemper
best	**69** (**48**%)	16 (11%)	23 (16%)	12 (8%)	17 (12%)	7 (5%)	
unique	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	
total	76 (58%)	**85** (**65**%)	52 (40%)	62 (47%)	71 (54%)	51 (39%)	rintanen
best	**40** (**34**%)	23 (19%)	6 (5%)	18 (15%)	21 (18%)	11 (9%)	
unique	0 (0%)	9 (24%)	0 (0%)	11 (29%)	**18** (**47**%)	0 (0%)	
total	36 (56%)	**37** (**58**%)	25 (39%)	36 (56%)	34 (53%)	33 (52%)	scholl-becker
best	6 (14%)	**17** (**39**%)	0 (0%)	1 (2%)	15 (34%)	5 (11%)	
unique	1 (25%)	**2** (**50**%)	0 (0%)	0 (0%)	1 (25%)	0 (0%)	
total	**20** (**62**%)	18 (56%)	5 (16%)	8 (25%)	6 (19%)	6 (19%)	verdes-geffner
best	**16** (**76**%)	2 (10%)	0 (0%)	0 (0%)	3 (14%)	0 (0%)	
unique	**3** (**75**%)	1 (25%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)	

Table 2: Number of instances solved by each solver on each benchmark of our test set (in total, as the fastest solver, and as the unique solver). The best results are in bold.

family	size	sKizzo	quantor	QuBE	yquaffle	semprop	SQBF
4x4-std	7	**14**	0	0	0	0	0
6x6-std	8	**25**	0	0	0	0	0
Adder2-true	8	**62**	38	12	12	25	12
Connect6	8	**62**	50	38	50	50	50
RobotsD3	8	**88**	12	75	38	25	75
adder-true	8	**75**	38	12	12	12	12
circuit2-property1	11	**73**	64	45	18	0	27
circuit2-property2	11	**100**	91	18	36	0	27
cnt	16	**100**	81	31	69	62	69
cnt_e	16	**62**	56	38	38	25	25
cnt_r	16	**81**	50	31	38	44	44
cnt_re	16	**62**	56	25	38	31	31
cube	10	**60**	50	20	30	20	20
emptyroom	6	**83**	67	17	33	17	17
k_branch_p	21	**86**	19	19	19	57	19
k_d4_n	21	**100**	24	19	14	29	14
k_d4_p	21	**100**	76	29	24	81	24
k_t4p_n	21	**100**	19	10	5	24	10
k_t4p_p	21	**100**	33	19	10	38	14
mutex	5	**100**	60	60	40	60	0
quant	8	**38**	0	0	0	0	0
quant_squaring	8	**12**	0	0	0	0	0
s298	8	**100**	12	0	0	0	0
s386	8	**25**	12	0	0	0	0
s499	8	**100**	38	0	0	0	0
s510	8	**25**	0	0	0	0	0
szymanski	10	**50**	30	10	0	30	0
toilet-c	85	**100**	99	99	96	98	96

Table 3: Percentage of solved instances in selected families.

8. Related works and discussion

We used many techniques and results developed in the literature on propositional reasoning, ranging from BDD-based representations to the combination of resolution and search in propositional decision procedures. We briefly review the literature related to each of these topics, aiming to point out the distinguishing features of our approach.

Resolution versus search in SAT and QBF. Our decision procedure combines resolution and search in a specific way, namely by first eliminating variables through directional resolution and then, if (in)consistency isn't detected within given computational resources, by performing search on the residual theory. An analysis of the alternating fortunes of search and resolution in propositional solvers accounts for this architectural choice. We briefly review the findings of works which compare or combine the two approaches.

The seminal paper (Davis & Putnam, 1960) first proposed to evaluate SAT instances using an ordering-based restricted resolution which progressively eliminates all the propositional symbols until the formula is decided (the "DP" procedure). This approach was empirically found to require an excessive amount of space, a fact later on analyzed and explained theoretically (Galil, 1977; Goerdt, 1992). A variant which runs in linear space was obtained by adopting a split (or branching) rule instead of resolution (Davis et al., 1962). Such branching variant (named "DLL" or, sometimes, "DPLL" procedure) has ever since been at the core of the most competitive SAT solvers.

The strengths and weaknesses of DP versus DLL were reassessed in (Dechter & Rish, 1994), where it was shown that many classes of instances are tractable for *variable elimination* (a new name for DP) and that under some natural structural restrictions such method is capable to largely outperform DLL. Nevertheless, steady improvements in the theory and implementation of search-based solvers—such as CSP-inspired look-back mechanisms to escape the chronological order of backtracks (Bayardo & Schrag, 1997) and learning techniques to avoid revisiting inconsistent assignments (Silva & Sakallah, 1996)—contributed to consolidate the reputation of DLL as the most effective decision procedure for SAT.

Hybrid decision schemes attempting to profit from the strengths of both methods were proposed. In one hybrid algorithm, a restricted form of variable elimination was used at each search node of a DLL solver to simplify the current subformula (Gelder, 2001). A different approach used the directional extension produced during variable elimination (made tractable but refutationally incomplete by bounding the length of resolvents) as a mechanism that implicitly compiles a branching heuristics, used to reduce the number of backtracks performed by a subsequent run of the DLL procedure (Rish & Dechter, 2000).

Though provably effective, these hybrid schemes were somewhat overshadowed by a newer generation of search-based solvers demonstrating impressive performance gains, due to the adoption of (i) dynamic branching heuristics that take into account the search history (Moskewicz et al., 2001), (ii) *lazy* data structures based on "watched literals" that do not require updates during backtracks (Moskewicz et al., 2001), and (iii) randomized restarts (Gomes, Selman, & Kautz, 1998).

The use of quantified resolution to evaluate QBFs was first considered in (Kleine-Buning et al., 1995), within a FOL-inspired approach aiming to generate resolvents until a contradiction is found (or it can be proved that no contradiction arises). Such technique has never been implemented (to the best of our knowledge). Contrariwise, the first efficient implementation of a QBF solver extended the competing DLL procedure (Cadoli et al., 1998). Most of the subsequent QBF reasoners preserve the search-based approach and build upon it (Feldmann, Monien, & Schamberger, 2000; Rintanen, 2001; Giunchiglia et al., 2001c; Letz, 2002; Zhang & Malik, 2002).

A surprisingly competitive procedure to evaluate QBFs has been recently obtained by carefully complementing the elimination of existential variables via quantified resolution with the elimination of universal variables via *expansion* (Biere, 2004). In this framework, the focus is on good heuristics to dynamically schedule the elimination of variables, as opposed to previous attempts where static (possibly optimal) orderings were considered.

The unexpected success of this method—which in essence turns back to the original DP approach—has once again lead to reconsider the role of variable elimination in SAT solvers. A resource-bounded version of variable elimination with greedy elimination order (used as a preprocessing step) has been recently shown to consistently improve the response of DLL solvers on the residual theory (Subbarayan & Pradhan, 2004; Eén & Biere, 2005).

A similar sequential scheme was independently adopted by the first hybrid prover for QBF, which combines search and resolution (Benedetti, 2005d). The procedure presented in Section 6 is basically a cleaned version of such hybrid scheme.

Finally, both resolution and search have been used in conjunction with compressed representations for clauses or assignments (based on BDDs or variants thereof). Such coupling opens further algorithmic possibilities, discussed in the next paragraph.

BDDs in SAT and QBF. Any SAT instance can in principle be decided by constructing the OBDD which represents its evaluation function (Bryant, 1986), obtained for example through a clause-by-clause conjunction of the meaning of each clause (Uribe & Stickel, 1994). The result encodes all the satisfying assignments of that formula. Furthermore, OBDDs are canonic, hence the formula is unsatisfiable if and only if such result is the constant function "0". Partial results of this incremental compilation may become unmanageably large. A possible remedy is to use (the diagram-based BDD version of) existential quantification to dynamically remove disposable variables (San Miguel Aguirre & Vardi, 2001). The focus then shifts on good heuristics to decide the ordering of conjunction operations, of BDD decision levels, and of variable elimination steps (Ranjan, Aziz, Brayton, Plessier, & Pixley, 1995; Pan & Vardi, 2004a).

An alternative possibility is to use *zero-suppressed* BDDs to represent clause sets (rather than sets of models) on top of which either search (Motter & Markov, 2002; Ghasemzadeh, 2005) or variable elimination (Chatalic & Simon, 2000) can be operated.

BDD-supported variable elimination has been readily extended to decide QBFs (Pan & Vardi, 2004b), as BDDs (resp. ZBDDs) support *universal* variable elimination as easily as (resp. more easily than) existential variable elimination. The effectiveness of these procedures (versus classical search-based ones) is comparatively larger than it is for SAT.

A QBF decision procedure which exploits BDDs to cumulate sets of models of the matrix obtained by an all-solution SAT solver is presented in (Audemard & Sais, 2005).

The role of BDDs in SQPL substantially deviates from all the above ones. Namely, BDDs are not used as a compressed representation for either QBF matrixes or sets of assignments. They are rather meant to capture interesting subsets of the definition domains of certain skolem symbols (which we have called "scenarios"). This approach is expressly designed for SQPL formulas, and no adaptation of previous SAT techniques is involved.

In SQPL, BDDs are not burdened with expressing the entire semantics (nor the syntax) of the original formula. Taken apart from the rest of the data structure they belong to, they fail to deliver any self-contained "uncompressed meaning". As a positive consequence, they never incur any exponential blow-up in the symbolic-skolemization step.

Nevertheless, they act as a compact data structure able to support efficiently both search and resolution, and contrarily to ZBDD-based compressions of the syntactic structure of formulas, they allow to easily detect and exploit both unit clauses and pure literals.

Binary clause reasoning. The ideas underlying the classical linear-time algorithm to detect unsatisfiability in binary theories (Aspvall et al., 1979) have been extended to find implied equivalent literals, implied unit clauses, and implied binary clauses in general propositional theories. This inferred information can be used to simplify the SAT problem by reducing the size of the space to be explored by DLL procedures.

In early approaches, binary reasoning was meant to be exercised at each node of the search tree (Gelder & Tsuji, 1996; Li, 2000). As soon as the efficiency of data structures for manipulating clause sets in DLL search increased, a critical trade-off emerged between the time saved by reducing the search space and the time spent in reasoning about how to shrink it. Most subsequent approaches use binary reasoning as a preprocessing step only (Brafman, 2001; Bacchus & Winter, 2003).

Binary clause reasoning in the framework of QBF decision procedures was introduced in (Benedetti, 2004, 2005d), where implied and equivalent literals are derived in a symbolic way, working in the space of skolemized QBF instances. The SQPL logic described in the present paper provides a simple theoretical foundation for such usage.

Recently, classical binary reasoning has been extended to deal with matrixes of QBFs right in their propositional clausal form. Both one-time static preprocessing methods (Samulowitz, Davies, & Bacchus, 2006) and on-the-fly dynamic approaches (Samulowitz & Bacchus, 2006) have been studied. A detailed comparison of the classical and symbolic binary reasoning in QBF is still to be carried out. However, the two methods have incomparable inference power. For example, any clause with two existential and more than zero universal literals is translated into a binary symbolic clause in the SQPL approach, hence it contributes to build the symbolic implication graph. The same clause would be ignored by classical binary reasoning. Contrariwise, a clause with one universal and one existential literal plays no role in the SIG, whereas it is exploited by classical methods. Clauses with one existential and one universal literal are the only ones to escape symbolic but not classical binary reasoning. By their structure, these clauses are necessarily captured and propagated as symbolic unit SQPL clauses. So, it is reasonable to conjecture that SUCP+SER strictly subsumes classical binary reasoning in QBF.

The exploitation of SAT solvers in QBF provers. The validity of any QBF can be expressed as the satisfiability of a proper SAT instance, though this process in general provokes an exponential blowup (unless PSPACE=NP). As shown in the present paper, one possible way to obtain a SAT instance equivalent to the QBF f is to evaluate $Pexp(SymbSk(f))$.

If the SAT expansion happens to be of manageable size, it can be addressed by state-of-the-art SAT solvers. However, this is almost never the case for non-trivial QBFs. A relaxation of this technique is to generate SAT instances which only capture a part of the meaning of the QBF, thus resulting in semi-decision procedures. This practice was first proposed in (Cadoli et al., 1998) in the form of *trivial truth/falsity* tests: If the SAT instance obtained from a QBF by considering all its variables as existentially quantified is unsatisfiable, then the originating QBF is false; if the instance obtained by discarding all the universal literals is satisfiable, then the QBF is true. Though no interesting QBF can be decided by this method alone, its recursive usage as a look-ahead tool in DLL-like search is advantageous, as it partly relaxes the left-to-right constraint on the branching order.

A data-structure level integration between a QBF and a SAT solver has been recently proposed to maximize the benefits of such relaxation (Samulowitz & Bacchus, 2005), within a mechanism that also allows the two engines to seamlessly exchange learnt clauses.

Other ways of exploiting SAT solvers in QBF provers have been proposed. For example, after all the variables but the ones in the most external (existential) scope have been eliminated through resolution and expansion as described in (Biere, 2004), the consistency of the remaining purely-existential theory is decided by a SAT solver (Biere, 2004).

A more sophisticated approach is presented in (Ayari & Basin, 2002), where subformulas of non-clausal quantified theories are processed through miniscoping, quantifier expansion, and simplifications, until they contain only one kind of quantification. At that point, the result is converted to clausal form and given to an off-the-shelf SAT solver.

In SQPL, SAT solving assumes a more important role than in other approaches. Our decision procedure is indeed essentially concerned with the consistency of an underlying SAT instance, equivalent to the original QBF, which is represented in a "symbolically" compressed way. Nothwidstanding the extreme compression, such representation admits a few key operations: a cheap but exact estimation of the size of the SAT expansion, and a number of tractable simplifications and manipulations (including the basic steps required to perform a complete search or resolution). All these operations can be performed without ever uncompressing the problem. Even if the originating QBF instance can almost never be inflated straighaway to a SAT problem, symbolic simplifications can substantially reduce the size of its expansion. By all means, this is the primary goal of the decision procedure: Normalize, split or simplify, in whatever way, the formula, so to trim the size of its expansion down to a combinatorial core which can be SAT-solved.

The usages of SAT solving in QBF we mentioned are somehow related to our approach. For example, a behavior similar to the one described in (Biere, 2004) is obtained as a special case from our procedure, namely when all the variables but the ones in the outermost existential scope are (symbolically) eliminated; at that point, the SQPL formula and its SAT expansion coincide (all the BDDs point to the "1" constant) so the expanded formula is immediately recognized as manageable. The mechanism used in (Ayari & Basin, 2002) manipulates and expand formulas explicitly, until they "loose" all the quantifiers of one kind; this perspective is reversed in our approach, as simplifications are operated symbolically, and expansion is realized as soon as the expanded version is known to be small.

Finally, the approach described in (Samulowitz & Bacchus, 2005) uses SAT solvers to decide (sometimes) sub-instances obtained at nested levels of a DLL-like recursive procedure. This look-ahead mechanism is similar to the one described in Algorithm 2, while the exchange of learned clauses is a peculiar feature of (Samulowitz & Bacchus, 2005).

Further work on SQPL. The present paper formalizes and extends ideas originally presented in (Benedetti, 2004, 2005a), while (Benedetti, 2005d) describes a (publicly available[9]) software architecture in which the decision procedure of Section 6 is implemented.

The extension from linear to tree-like quantification structures, and its positive impact on symbolic skolemization and decision procedures thereof, is studied in (Benedetti, 2005c).

As briefly mentioned in Section 3, SQPL models can be used to certify the validity of QBFs, as they encode a strategy (or policy) to satisfy the matrix under the quantification structure given in the prefix. Ways and means to represent, compute, and validate such models have been studied in (Benedetti, 2005b), while an account of possible applications is outlined in (Benedetti & Mangassarian, 2007). A publicly available implementation of the certification and strategy-extraction technology exists[9].

A search-based decision procedure for SQPL which is basically different from the one built after Theorem 5.4 has been published in (Benedetti, 2006). In such branching scheme, only β-formulas originate backtrack points, while α-formulas are decided by inference.

Finally, the SQPL framework has been used to experiment with the concept of restricted quantification in QBF. Preliminary results are in (Benedetti, Lallouet, & Vautard, 2007).

9. The solver is publicly available from the web site (Benedetti, 2005e), which also collects up-to-date experiments, documents, and papers concerning SQPL and its applications.

9. Conclusions and Future Work

We introduced a new way to decide QBFs. Our work is motivated by the contrast between the remarkable potential of QBF in applications, and the relatively weak performances of present solvers, most of which are based on techniques derived from SAT solvers. Our approach, by contrast, is inspired to FOL provers and has skolemization at the heart.

In order to overcome the issues involved in reasoning on skolemized QBFs, we introduced a purpose-built formalism, in which acceptable interpretations for the skolem terms are captured by means of a peculiar blending of BDDs with clause sets. The whole point of this representation is that it allows us to operate inferences and build decision procedures by combining a few primitive manipulations over decision diagrams and clauses. BDDs are exclusively concerned with the "universal side" of the reasoning, while clauses are used on the "existential side". The resulting method differs not only from previous QBF solvers, but also from all the previous ways of using BDDs in propositional logic we are aware of.

Classical decision procedures (such as DLL and DP) and specific reasoning techniques (such as symbolic binary reasoning, pure literal elimination, and propositional expansion) coexist in our framework, an implementation of which has been shown to significantly outperform other state-of-the-art approaches on a large set of benchmarks from applications.

Our future work on the topic will move in three directions.

First, many techniques have been proposed in the QBF literature (such as model caching, watched data structures, cooperative QBF/SAT learning, etc.) which are adaptable to our framework, but have been left out of the current implementation to focus on the main topic. Experimental results could further benefit from implementing such techniques.

Second, many propositional inference rules exist—such as hyper-binary resolution and other powerful form of binary reasoning—whose SQPL version has not yet been studied.

Finally, SQPL opens novel perspectives on the way problems can be expressed and solved, which are worth systematic exploration. Examples include the possibility to construct unconventional decision procedures on top of SQPL—such as (Benedetti, 2006)—and the valuable capability to extend the modeling language with a small effort, for example by means of restricted quantification (Benedetti et al., 2007).

Acknowledgments. I thank Sylvain Jubertie for his patience in assisting me during the interminable experimental tests I performed on the cluster he manages at LIFO. Amedeo Cesta deserves a special thought for his indefatigable encouragement. I'm deeply grateful to Marco Cadoli for discussing symbolic skolemization with me, and for urging me to put ideas into practice. This paper is dedicated to his memory.

References

Ansótegui, C., Gomes, C. P., & Selman, B. (2005). The Achilles' Heel of QBF.. In *Proceedings of the 20th AAAI National Conference on Artificial Intelligence (AAAI-05)*. AAAI Press.

Aspvall, B., Plass, M. F., & Tarjan, R. E. (1979). A linear-time algorithm for testing the truth of certain quantified boolean formulas.. *Inf. Process. Lett.*, *8*(3), 121–123.

Audemard, G., & Sais, L. (2005). A Symbolic Search Based Approach for Quantified Boolean Formulas. In T. Walsh, F. B. (Ed.), *eighth International Conference on Theory and Applications of Satisfiability Testing (SAT2005)*, Vol. 3569 of *LNCS*, pp. 16–30. Springer Verlag.

Ayari, A., & Basin, D. (2000). Bounded Model Construction for Monadic Second-order Logics. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV-00)*, No. 1855 in LNCS. Springer.

Ayari, A., & Basin, D. (2002). QUBOS: Deciding Quantified Boolean Logic Using Propositional Satisfiability Solvers. In *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD-02)*, No. 2517 in LNCS. Springer.

Baaz, M., & Leitsch, A. (1994). On skolemization and proof complexity. *Fundamenta Informaticae*, pp. 353–379.

Bacchus, F., & Winter, J. (2003). Effective Preprocessing with Hyper-Resolution and Equality Reduction. In *Proc. of SAT-03*.

Bayardo, R., & Schrag, R. (1997). Using CSP look-back rechniques to solve real-world SAT instances. In *Proceedings of the 14th AAAI National Conference on Artificial Intelligence (AAAI-97)*. AAAI Press.

Benedetti, M. (2004). sKizzo: a QBF Decision Procedure based on Propositional Skolemization and Symbolic Reasoning. Tech. rep. 04-11-03, ITC-irst, Centre for Scientific and Technological Research.

Benedetti, M. (2005a). Evaluating QBFs via Symbolic Skolemization. In *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-04)*, No. 3452 in LNCS. Springer.

Benedetti, M. (2005b). Extracting Certificates from Quantified Boolean Formulas. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*.

Benedetti, M. (2005c). Quantifier Trees for QBFs. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-05)*, No. 3569 in LNCS. Springer.

Benedetti, M. (2005d). sKizzo: A Suite to Evaluate and Certify QBFs. In *Proceedings of the 20th International Conference on Automated Deduction (CADE-05)*, No. 3632 in LNCS. Springer.

Benedetti, M. (2005e). sKizzo's web site, `http://sKizzo.info`..

Benedetti, M. (2006). Abstract Branching for Quantified Formulas. In *Proc. of 21st AAAI National Conference on Artificial Intelligence (AAAI-06)*. AAAI Press.

Benedetti, M., Lallouet, A., & Vautard, J. (2007). QCSP made Practical by Virtue of Restricted Quantification. *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*.

Benedetti, M., & Mangassarian, H. (2007). Experience and Perspectives in QBF-based Formal Verification. *Submitted to the Journal of Satisfiability*.

Biere, A. (2004). Resolve and Expand. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT-04)*, No. 3542 in LNCS, pp. 238–246. Springer.

Biere, A., Cimatti, A., Clarke, E. M., Fujita, M., & Zhu, Y. (1999). Symbolic Model Checking without BDDs. In *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS-99)*, No. 1579 in LNCS, pp. 193–207. Springer.

Brafman, R. I. (2001). A simplifier for propositional formulas with many binary clauses. In *Proc. of IJCAI'01*.

Bryant, R. E. (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transaction on Computing*, *C-35*(8), 677–691.

Bryant, R. E., Lahiri, S. K., & Seshia, S. A. (2003). Convergence testing in term-level bounded model checking. In *Proc. of CHARME-03*, Vol. 2860 of *LNCS*.

Büning, H. K., & Zhao, X. (2004). On Models for Quantified Boolean Formulas. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT-04)*, No. 3542 in LNCS. Springer.

Cadoli, M., Eiter, T., & Gottlob, G. (1997). Default logic as a query language. *IEEE Transactions on Knowledge and Data Engineering*, *9*(3), 448–463.

Cadoli, M., Giovanardi, A., & Schaerf, M. (1998). An algorithm to evaluate quantified boolean formulae. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pp. 262–267. American Association for Artificial Intelligence.

Cadoli, M., & Schaerf, A. (2005). Compiling problem specifications into SAT. *Artificial Intelligence*, *1-2*(162), 89–120.

Castellini, C., Giunchiglia, E., & Tacchella, A. (2003). SAT-based planning in complex domains: Concurrency, constraints and nondeterminism. *Artificial Intelligence*, *147*(1-2), 85–117.

Chatalic, P., & Simon, L. (2000). Multi-Resolution on compressed sets of clauses. In *Proceedings of the 12th International Conference on Tools with Artificial Intelligence (ICTAI-00)*.

Cook, B., Kroening, D., & Sharygina, N. (2004). Accurate Theorem Proving for Program Verification. In *Proc. of ISoLA-04 (Leveraging Applications of Formal Methods)*.

Cook, B., Kroening, D., & Sharygina, N. (2005). Symbolic model checking for asynchronous Boolean programs. In *Proc. of the 12th SPIN Workshop on Model Checking Software*.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, pp. 151–158.

Crawford, J. M., & Baker, A. D. (1994). Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the 12th AAAI National Conference on Artificial Intelligence (AAAI-94)*, pp. 1092–1097. AAAI Press.

Davis, M., Logemann, G., & Loveland, D. (1962). A machine program for theorem proving. *Journal of the ACM*, *5*, 394–397.

Davis, M., & Putnam, H. (1960). A computing procedure for quantification theory. *Journal of the ACM*, *7*.

Dechter, R., & Rish, I. (1994). Directional resolution: The Davis-Putnam procedure, revisited. In Doyle, J., Sandewall, E., & Torasso, P. (Eds.), *KR'94: Principles of Knowledge Representation and Reasoning*, pp. 134–145. Morgan Kaufmann, San Francisco, California.

Dershowitz, N., Hanna, Z., & Katz, J. (2005). Bounded Model Checking with QBF. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-05)*, No. 3569 in LNCS. Springer.

Een, N., & Sorensson, N. (2003). An Extensible SAT-solver. In *Proc. of SAT-03*.

Eén, N., & Biere, A. (2005). Effective preprocessing in sat through variable and clause elimination.. In Bacchus, F., & Walsh, T. (Eds.), *SAT*, Vol. 3569 of *Lecture Notes in Computer Science*, pp. 61–75. Springer.

Egly, U. (1998). Quantifiers and the System KE: Some Surprising Results. In *Proc. of CSL'98*.

Egly, U., Eiter, T., Tompits, H., & Woltran, S. (2000). Solving advanced reasoning tasks using quantified Boolean formulas. In *Proc. of AAAI-00*.

Egly, U., Seidl, M., Tompits, H., Woltran, S., & Zolda, M. (2003). Comparing Different Prenexing Strategies for Quantified Boolean Formulas. In *Proc. of SAT'03*.

Egly, U., Seidl, M., & Woltran, S. (2006). A Solver for QBFs in Nonprenex Form. In *Proc. of ECAI*.

Feldmann, R., Monien, B., & Schamberger, S. (2000). A Distributed Algorithm to Evaluate Quantified Boolean Formulas. In *Proceedings of the 17th AAAI National Conference on Artificial Intelligence (AAAI-00)*, pp. 285–290. AAAI Press.

Galil, Z. (1977). On the Complexity of Regular Resolution and the Davis-Putnam Procedure. *Theor. Computer Science*, *4*(1), 23–46.

Gambin, A., Lasota, S., & Rutkowski, M. (2006). Analyzing stationary states of gene regulatory network using petri nets. *Silico Biology*, *6*, 93–109.

Gelder, A. V. (2001). Combining preorder and postorder resolution in a satisfiability solver. In *Proc. of the IEEE/ASL LICS Workshop on Theory and Applications of Satisfiability Testing (SAT)*.

Gelder, A. V., & Tsuji, Y. K. (1996). Satisfiability testing with more reasoning and less guessing. Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*.

Gent, I., & Rowley, A. (2003). Encoding connect-4 using quantified boolean formulae. Tech. rep. APES-68-2003, APES Research Group.

Ghasemzadeh, M. (2005). *A new algorithm for the quantified satisfiability problem, based on zero-suppressed binary decision diagrams and memoization.* Ph.D. thesis, University of Potsdam, Germany.

Giunchiglia, E., Maratea, M., Tacchella, A., & Zambonin, D. (2001a). Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR-01)*, No. 2083 in LNAI, pp. 347–363. Springer.

Giunchiglia, E., Narizzano, M., & Tacchella, A. (2001b). Backjumping for Quantified Boolean Logic Satisfiability. In *Proc. of IJCAI-01.*

Giunchiglia, E., Narizzano, M., & Tacchella, A. (2001c). QuBE: A system for deciding Quantified Boolean Formulas Satisfiability. In *Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR-01)*, No. 2083 in LNAI. Springer.

Giunchiglia, E., Narizzano, M., & Tacchella, A. (2006). Quantifier Structure in Search Based Procedures for QBFs. In *Proceedings of the conference on Design, automation and test in Europe (DATE-06).*

Goerdt, A. (1992). Davis-Putnam Resolution versus Unrestricted Resolution. *Annals of Mathematics and Artificial Intelligence*, *6*(1-3), 169–184.

Gomes, C. P., Selman, B., & Kautz, H. (1998). Boosting combinatorial search through randomization. In *Proc. of AAAI-98.*

Gopalakrishnan, G., Yang, Y., & Sivaraj, H. (2004). QB or Not QB: An Efficient Execution Verification Tool for Memory Orderings. In *Proc. of CAV-04.*

Herbstritt, M., & Becker, B. (2007). On Combining 01X-Logic and QBF. In *Proc. of the 11th International Conference on Computer Aided Systems Theory (EuroCAST-07).*

Jussila, T., & Biere, A. (2006). Compressing BMC Encodings with QBF. In *Proceedings of the 4th International Workshop on Bounded Model Checking (BMC-06), Seattle (USA).*

Katz, J., Hanna, Z., & Dershowitz, N. (2005). Space-Efficient Bounded Model Checking. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE-05).* IEEE Computer Society.

Kautz, H., & Selman, B. (1992). Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92)*, pp. 359–363.

Kim, J., Whittemore, J., Silva, J. P. M., & Sakallah, K. A. (2000). On Applying Incremental Satisfiability to Delay Fault Problem. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE-00)*, pp. 380–384. IEEE Computer Society.

Kleine-Buning, H., Karpinski, M., & Flogel, A. (1995). Resolution for quantified Boolean formulas. *Information and Computation*, *117*(1), 12–18.

Larrabee, T. (1992). Test pattern generation using boolean satisfiability. In *IEEE Transaction on Computer-aided Design*, Vol. 11, pp. 6–22.

Le Berre, D., Narizzano, M., Simon, L., & Tacchella, A. (2004). Second QBF solvers evaluation, avaliable on-line at `www.qbflib.org`..

Letz, R. (2002). Lemma and model caching in decision procedures for quantified boolean formulas. In *Proc. of the Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods*, pp. 160–175. Springer-Verlag.

Li, C.-M. (2000). Integrating equivalency reasoning into Davis-Putnam procedure. In *Proceedings of the 17th AAAI National Conference on Artificial Intelligence (AAAI-00)*, pp. 291–296. AAAI Press.

Ling, A., Singh, D. P., & Brown, S. D. (2005). FPGA Logic Synthesis using Quantified Boolean Satisfiability. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-05)*, No. 3569 in LNCS. Springer.

Mangassarian, H., Veneris, A., Safarpour, S., Benedetti, M., & Smith, D. (2007). Performance-Driven Bounded Sequential Modeling with QBF. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD-07)*.

Massacci, F., & Marraro, L. (2000). Logical Cryptanalysis as a SAT Problem. *Journal of Automated Reasoning*, *24*.

Mneimneh, M., & Sakallah, K. (2003). Computing Vertex Eccentricity in Exponentially Large Graphs: QBF Formulation and Solution. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT-03)*, No. 2919 in LNCS. Springer.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., & Malik, S. (2001). Chaff: Engineering an Efficient SAT Solver. In *proceedings of the 38th Design Automation Conference*.

Motter, D. B., & Markov, I. L. (2002). A compressed, breadth-first search for satisfiability. In *Proc. of the 4th Workshop on Algorithm Engineering and Experiments (ALENEX-02)*, Vol. 2409 of *LNCS*, pp. 29–42. Springer.

Narizzano, M., Pulina, L., & Tacchella, A. (2006). Report of the Third QBF solvers evaluation. *Journal of Satisfiability*, pp. 145–164.

Oetsch, J., Seidl, M., Tompits, H., & Woltran, S. (2006). A Tool for Advanced Correspondence Checking in Answer-Set Programming: Preliminary Experimental Results. In *Proc. of WLP-06*.

Palacios, H., & Geffner, H. (2005). Mapping Conformant Planning into SAT Through Compilation and Projection. In *Proc. of 11th Conference of the Spanish Association for Artificial Intelligence (CAEPIA-05)*.

Pan, G., & Vardi, M. (2003). Optimizing a BDD-based Modal Solver. In *Proceedings of CADE 2003*.

Pan, G., & Vardi, M. (2004a). Search vs. Symbolic Techniques in Satisfiability Solving. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT-04)*, No. 3542 in LNCS. Springer.

Pan, G., & Vardi, M. (2004b). Symbolic Decision Procedures for QBF. In *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP-04)*, No. 3258 in LNCS. Springer.

Papadimitriou, C. H. (1994). *Computational Complexity*. Addison Wesley.

Ranjan, R. K., Aziz, A., Brayton, R. K., Plessier, B., & Pixley, C. (1995). Efficient BDD algorithms for FSM synthesis and verification. In *Proceedings of IEEE/ACM International Workshop on Logic Synthesis*.

Remshagen, A., & Truemper, K. (2005). An Effective Algorithm for the Futile Questioning Problem. *Journal of Automated Reasoning*, *1*(34), 31–47.

Rintanen, J. (1999a). Construction Conditional Plans by a Theorem-prover. *Journal of Artificial Intelligence Research*, *10*, 323–352.

Rintanen, J. (1999b). Improvements to the Evaluation of Quantified Boolean Formulae. In *Proc. of IJCAI-99*.

Rintanen, J. (2001). Partial implicit unfolding in the Davis-Putnam procedure for quantified boolean formulae.. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-01)*, No. 2250 in LNCS. Springer.

Rish, I., & Dechter, R. (2000). Resolution versus search: Two strategies for sat. In et al., I. G. (Ed.), *SAT2000*, pp. 215–259. IOS Press.

Samulowitz, H., & Bacchus, F. (2005). Using SAT in QBF. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP-05)*, No. 3709 in LNCS. Springer.

Samulowitz, H., & Bacchus, F. (2006). Binary Clause Reasoning in QBF. In *Proc. of 9th Int. Conference on Theory and Applications of Satisfiability Testing (SAT-06)*, No. 4121 in LNCS. Springer.

Samulowitz, H., Davies, J., & Bacchus, F. (2006). Preprocessing QBF. In *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP-06)*, No. 4204 in LNCS. Springer.

San Miguel Aguirre, A., & Vardi, M. Y. (2001). Random 3-SAT and BDDs: The plot thickens further. In *Proc. of the 7th International Conference on Principle and Practice of Constraint Programming*.

Scholl, C., & Becker, B. (2000). Checking equivalence for partial implementation. Tech. rep., Institute of Computer Science, Albert-Ludwigs University.

Scholl, C., & Becker, B. (2001). Checking equivalence for partial implementations. In *Proc. of DAC-01*.

Silva, J. P., & Sakallah, K. A. (1996). Grasp: a new search algorithm for satisfiability. *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 220–226.

Staber, S., & Bloem, R. (2007). Fault Localization and Correction with QBF. In *Proc. of SAT-07*.

Stockmeyer, L. J., & Meyer, A. R. (1973). Word Problems Requiring Exponential Time. In *In 5th Annual ACM Symposium on the Theory of Computing*.

Subbarayan, S., & Pradhan, D. K. (2004). Niver: Non increasing variable elimination resolution. In *Proceedings of The Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT '04)*, pp. 351–356.

Uribe, T. E., & Stickel, M. E. (1994). Ordered binary decision diagrams and the Davis-Putnam procedure. In Jouannaud, J. P. (Ed.), *1st International Conference on Constraints in Computational Logics*, Vol. 845, pp. 34–49.

Wegener, I. (2000). *Branching Programs and Binary Decision Diagrams.* Monographs on Discrete Mathematics and Applications. SIAM.

Zhang, L., & Malik, S. (2002). Towards Symmetric Treatment of Conflicts And Satisfaction in Quantified Boolean Satisfiability Solver. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP-02)*, No. 2470 in LNCS. Springer.