Abstract Solvers for Quantified Boolean Formulas and their Applications

Remi Brochenin(✉) and Marco Maratea

DIBRIS, University of Genova, Viale F. Causa 15, Genova, Italy
{remi.brochenin,marco.maratea}@unige.it

Abstract. Abstract solvers are a graph-based representation employed in many research areas, such as SAT, SMT and ASP, to model, analyze and compare search algorithms in place of pseudo-code-based representations. Such an uniform, formal way of presenting the solving algorithms proved effective for their understanding, for formalizing related formal properties and also for combining algorithms in order to design new solving procedures.

In this paper we present abstract solvers for Quantified Boolean Formulas (QBFs). They include a direct extension of the abstract solver describing the DPLL algorithm for SAT, and an alternative formulation inspired by the two-layers architecture employed for the analysis of disjunctive ASP solvers. We finally show how these abstract solvers can be directly employed for designing solving procedures for reasoning tasks which can be solved by means of reduction to a QBF.

1 Introduction

Abstract solvers are a relatively new methodology that have been employed in many research areas, such as Propositional Satisfiability (SAT) [1], Satisfiability Modulo Theories (SMT) [1,2], Answer Set Programming (ASP) [3–5], and Constraint ASP [6], to model, analyze and compare solving algorithms in place of pseudo-code-based representations. Abstract solvers are a graph-based representation, where the states of computation are represented as nodes of a graph, the solving techniques as arcs between such nodes, the solving process as a path in the graph and the formal properties of the algorithms are reduced to related graph's properties. Such a uniform, mathematically simple yet formal way of presenting the solving algorithms proved effective for their understanding, for formalizing properties in a clear way and also for combining algorithms for designing new solutions. However, with the notable exception of the recent work on disjunctive ASP [5], up to now this methodology has been employed to solving procedures for reasoning tasks whose complexity is within the first level of the polynomial hierarchy.

In this paper we present, for the first time, abstract solvers for deciding the satisfiability of Quantified Boolean Formulas (QBFs) [7], the prototypical PSPACE-complete problem, thus showing their potential also to analyze "hard" reasoning tasks. The first abstract solution is an extension of the abstract

M. Gavanelli et al. (Eds.): AI*IA 2015, LNAI 9336, pp. 205–217, 2015.
DOI: 10.1007/978-3-319-24309-2_16

solver [1] for describing the DPLL algorithm for SAT [8], enhanced with theory-specific techniques, modeled as additional arcs in the respective graph. The second abstract solver is, instead, based on the two-layers architecture employed for the analysis of disjunctive ASP solvers, which are characterized by a generating layer for finding candidate solutions, and a test layer for checking candidates's minimality: this second solution employs a "multi-layer" architecture whose number of layers depend on the quantifiers alternation in the QBF formula. We also comment on a third viable approach based on compilation into a SAT formula, on which is applied an abstract solver for SAT. For all abstract solutions, correctness results are formalized by means of related graph's properties. We finally show how abstract solvers for QBFs can be directly employed for solving certain reasoning tasks in other areas such as Answer Set Programming [9] and Abstract Argumentation and Dialectical frameworks (see, e.g. [10,11]), whose solutions are obtained by means of reduction to a QBF, thus where an abstract solver for QBF becomes an abstract solver for the (compiled) respective reasoning task.

To sum up, the main contributions of this paper are:

- We present a first abstract solver for QBFs that extends the abstract solver describing the DPLL algorithm for SAT.
- We present a second abstract solver that employs a multi-layer architecture whose idea comes from the two-layers architecture of disjunctive ASP solvers.
- We show a third solution, based on compilation to SAT and an abstract solver for SAT, and correctness results for all mentioned solutions by means of properties of related graphs.
- We show how the afore-presented abstract solvers can be directly employed for solving certain reasoning tasks in other fields.

The paper is structured as follows. Section 2 introduces needed preliminaries. Section 3 then presents the abstract solvers for backtracking-based procedures. Section 4 shows some applications of the introduced abstract solvers. The paper then ends with a discussion of related work and some conclusions in Section 5.

2 Formal Background

Syntax and Semantics. Consider a set P of variables (also called atoms). The symbols $\bot$ and $\top$ denote *false* and *true*, respectively. A literal is a variable a_0 or its negation $\overline{a_0}$. $\overline{\overline{a_0}}$ is the same of a_0. A clause is a finite set of literals. A SAT formula F' in Conjunctive Normal Form (CNF) is a finite set of clauses.

A QBF formula is an expression F of the form:

$$Q_0 X_0 Q_1 X_1 \dots Q_n X_n F' \tag{1}$$

where:

- every Q_i $(0 \leq i \leq n)$ is a quantifier, either existential $\exists$ or universal $\forall$, and such that for each $0 \leq j \leq n-1$, $Q_j \neq Q_{j+1}$, i.e. we assume an alternation of quantifiers, and assume that the innermost quantifier is $\exists$;
- $X_0, \ldots, X_n$ are variable groups (equivalently seen as sets) which define a *partition* of P (i.e. the formula is closed) such that (i) each $0 \leq i \leq n$ $X_i \neq \emptyset$, (ii) $\bigcup_{i,0\leq i\leq n} X_i = P$, and (iii) for each $0 \leq i,j \leq n$, $i \neq j$, $X_i \cap X_j = \emptyset$.
- F' is a SAT formula over P.

In (1), $Q_0X_0Q_1X_1 \ldots Q_nX_n$ is the *prefix* and F' is the *matrix*. A *level* of a literal l built on a variable $b_i \in X_i$, denoted $level(l)$, is i with $0 \leq i \leq n$. We assume the formula has max alternations of quantifiers, so max is n. If F is (1) and l is a literal $l = b_i$ or $l = \neg b_i$ for $b_i \in X_i$, then F_l is the QBF:

- whose matrix is obtained from F by substituting (i) b_i with $\top$ and $\neg b_i$ with $\bot$ if $l = b_i$, and (ii) b_i with $\bot$ and $\neg b_i$ with $\top$, otherwise;
- whose prefix is $Q_0X_0Q_1X_1 \ldots Q_i(X_i \setminus b_i) \ldots Q_nX_n$.

The *semantics* of a QBF F can be defined recursively as follows:

1. If the prefix is empty, according to the semantics of propositional logic.
2. If F is $\exists bF'$, $b \in P$, F is satisfiable iff F_b is satisfiable or $F_{\neg b}$ is satisfiable.
3. If F is $\forall bF'$, $b \in P$, F is satisfiable iff both F_b and $F_{\neg b}$ are satisfiable.

Example 1. In the QBF (2) below (from [7]), X_0 is $\{a\}$, X_1 is $\{d\}$, X_2 is $\{b,c\}$, $\exists a \forall d \exists bc$ is the prefix and $\{\{\overline{a},\overline{d},b\},\{\overline{d},\overline{b}\},\{b,c\},\{a,\overline{d},\overline{c}\},\{d,b,\overline{c}\}\}$ is the matrix. Note that (2) is unsatisfiable.

$$\exists a \forall d \exists bc \{\{\overline{a},\overline{d},b\},\{\overline{d},\overline{b}\},\{b,c\},\{a,\overline{d},\overline{c}\},\{d,b,\overline{c}\}\} \tag{2}$$

Q-DPLL algorithm. An *assignment* to a set X of atoms is a function from X to $\{\bot,\top\}$. A set of literals is called *consistent* if for any literal l it contains it does not contain $\overline{l}$. We identify a consistent set of literals M with an assignment to $At(M)$: $a \in M$ iff a maps to $\top$, and $\neg a \in M$ iff a maps to $\bot$.

Q-DPLL is an extension of DPLL algorithm for SAT for determining the satisfiability of a QBF. As DPLL exhaustively explores the space of assignments, by assigning literals either deterministically or heuristically, to generate classical models of a propositional formula, also Q-DPLL is a classical backtrack-search algorithm that exhaustively explores the space of assignments to test the satisfiability of a QBF. The main difference is that, when a literal whose atom is universally quantified is heuristically chosen, both branches must be explored.

A pseudo-code description of the Q-DPLL algorithm can be found in [7,12]. Solvers implementing this approach are, e.g. EVALUATE [12] and QUBE [13].

Abstract Q-DPLL for SAT. As said before, a QBF without prefix corresponds to a SAT formula, where all atoms are existentially quantified. Some more preliminaries about abstract solvers and related graphs are needed. A *universal literal* is a literal annotated as $l^{\forall}$. An *existential literal* is a literal annotated as $l^{\exists}$.

A *decision literal* is a universal literal or an existential literal. A *record* is a string of literals and decision literals. The *terminal states* are $Valid$ and $Unsat$. A *state* is a record or a terminal state. The *initial state* is $\emptyset$.

Figure 1 will present the transition rules for the graph describing the Q-DPLL algorithm, but if we restrict to the rules $Unit$, $Decide$, $Backtrack_\exists$, $Fail$ and $Succeed$, and considering that all atoms are on the same level, we obtain a description of the DPLL algorithm for SAT as presented in, e.g. [5]. Given a SAT formula, $Unit$ adds to the current assignment an unassigned literal in a clause where all other literals are contradicted. $Decide$ adds to the current assignment an unassigned literal. $Backtrack_\exists$ restores an inconsistent assignment by going back in the current assignment and flipping the last decision literal. $Fail$ determines the formula to be unsatisfiable, i.e. the current assignment is inconsistent but can not be fixed given that it does not contain any decision literal. Finally $Succeed$ determines the formula to be satisfiable.

Example 2. Below are two possible paths in the graph $QBF_{\exists a,b,c\{\{a,b\},\{\overline{a},c\}\}}$:

Initial state :		$\emptyset$	Initial state :		$\emptyset$
$Decide$	$\Longrightarrow$	$a^\exists$	$Decide$	$\Longrightarrow$	$a^\exists$
$Unit$	$\Longrightarrow$	$a^\exists\ c$	$Decide$	$\Longrightarrow$	$a^\exists\ \overline{c}^\exists$
$Decide$	$\Longrightarrow$	$a^\exists\ c\ b^\exists$	$Unit$	$\Longrightarrow$	$a^\exists\ \overline{c}^\exists\ c$
$Succeed$	$\Longrightarrow$	$Valid$	$Backtrack_\exists$	$\Longrightarrow$	$a^\exists\ c$
			$Decide$	$\Longrightarrow$	$a^\exists\ c\ b^\exists$
			$Succeed$	$\Longrightarrow$	$Valid$

In order to realistically describe the DPLL algorithm, an ordering must be given on the application of the rules, such that a transition rule can not be applied if a rule with higher priority is applicable. In DPLL, the ordering follows how the rules $Unit, Decide, Backtrack_\exists, Fail$, and $Succeed$ are listed in Figure 1. Thus, the path in Example 2 at the left corresponds to a possible path of the DPLL algorithm, while the path at the right does not, given that $Decide$ is applied when $Unit$ is applicable.

Finally, we say that a graph G *verifies* a statement S (e.g. G verifies that F is satisfiable) when all the following conditions hold:

1. G is finite and acyclic;
2. Any terminal state in G is either $Failstate$ or $Valid$;
3. If a state $Valid$ is reachable from the initial state in G then S holds (e.g. F is satisfiable);
4. $Failstate$ is reachable from the initial state in G if and only if S does not hold (e.g. F is not satisfiable).

3 Abstract Solvers for QBFs

In this section we introduce the three abstract solvers for deciding the satisfiability of a QBF mentioned in the introduction.

Q-DPLL on a single layer. We show here a description of the Q-DPLL algorithm, within a single layer of computation. As we already wrote, the Q-DPLL algorithm is an extension of the DPLL algorithm for SAT. As a consequence, its transition system updates some of the transition rules for SAT, and introduces further transition rules to take into account the specific problem. First, unit propagate can now be applied subject to further specific conditions. Second, the decision literal must be chosen such that there is no an unassigned literal at a higher level. Third, there are now two types of backtracking, i.e. through an existentially-quantified literal, whose value is switched after a contradiction, or through an universally-quantified literal, whose value is switched after a successful branch. Finally, monotone rules that take into account situations where only a literal, or its negation, is in the formula, are also added. In the following, we will formalize these updates and modular additions.

The graph QBF_F has the states as nodes, as defined in the previous section, and the transitions of Figure 1 as edges.

The rules *Unit*, *Decide* and $Backtrack_\exists$ extend the ones for SAT in Section 2. *Unit* adds to the current assignment an unassigned literal in a clause if all other assigned literals are contradicted, *and all other unassigned literals are universally quantified*. *Decide* adds to the current assignment an unassigned literal, *either existentially or universally quantified, such that all atoms at lower levels are assigned*. $Backtrack_\exists$ restores an inconsistent assignment by going back in the assignment and flipping the last *existentially-quantified* decision variable.

Specific rules for QBF are $Monotone1$, $Monotone2$ and $Backtrack_\forall$. The rule $Monotone1$ (resp. $Monotone2$) assigns an existential (resp. universal) literal l, and l (resp. $\bar{l}$) appears in some clause while the opposite does not appear in any clause. $Backtrack_\forall$ is a counterpart of $Backtrack_\exists$: it flips the value of the last universal literal after finding a complete and consistent assignment. For that, it goes back in the current assignment and flips the last decision variable that is universally quantified. Note that this rule is applicable only when all other rules but *Succeed* can not be applied, so that it is triggered only when the current assignment both assigns all the atoms in F since *Decide* does not apply and is consistent since *Fail* does not apply. *Fail* and *Succeed* are the same as for SAT, except that *Succeed* can be triggered only when there is no decision variable that is universally quantified so that $Backtrack_\forall$ is not applicable.

Proposition 1. *Given a QBF formula F, the graph QBF_F verifies that F is satisfiable.*

Example 3. Consider the QBF (2) in Example 1, that we call F. A possible path in QBF_F is:

Initial state :	$\emptyset$	$Backtrack_\exists$	$\Longrightarrow a$
Decide	$\Longrightarrow \overline{a}^\exists$	*Decide*	$\Longrightarrow a\overline{d}^\forall$
Decide	$\Longrightarrow \overline{a}^\exists \overline{d}^\forall$	$Monotone1$	$\Longrightarrow a\overline{d}^\forall b$
$Monotone1$	$\Longrightarrow \overline{a}^\exists \overline{d}^\forall b$	$Backtrack_\forall$	$\Longrightarrow a\ d$
$Backtrack_\forall$	$\Longrightarrow \overline{a}^\exists d$	*Unit*	$\Longrightarrow a\ d\ b$
Unit	$\Longrightarrow \overline{a}^\exists d\ \overline{c}$	*Fail*	$\Longrightarrow Unsat$
Unit	$\Longrightarrow \overline{a}^\exists d\ \overline{c}\overline{b}$		

Rules

$$
\begin{array}{llll}
Unit & L & \Longrightarrow Ll & \text{if} \left\{ \begin{array}{l} l \text{ does not occur in } L \text{ and} \\ \text{for some clause } C \text{ in the matrix,} \\ l \text{ occurs in } C \text{ and} \\ \text{each other unassigned literal of } C \text{ is universal and} \\ \text{each assigned literal of } C \text{ is contradicted} \end{array} \right. \\
Monotone1 & L & \Longrightarrow Ll & \text{if} \left\{ \begin{array}{l} \text{the variable of } l \text{ is existential and} \\ l \text{ occurs in some clause } C \text{ and} \\ \bar{l} \text{ does not occur in any clause } C \end{array} \right. \\
Monotone2 & L & \Longrightarrow Ll & \text{if} \left\{ \begin{array}{l} \text{the variable of } l \text{ is universal and} \\ \bar{l} \text{ occurs in some clause } C \text{ and} \\ l \text{ does not occur in any clause } C \end{array} \right. \\
Decide & L & \Longrightarrow Ll^Q & \text{if} \left\{ \begin{array}{l} L \text{ is consistent and} \\ \text{the variable of } l \text{ is unassigned and} \\ \text{the quantifier of the variable of } l \text{ is } Q \text{ and} \\ \text{for all } l' \text{ such that } level(l') < level(l) \\ \text{the variable of } l' \text{ is assigned.} \end{array} \right. \\
Backtrack_\exists & Ll^\exists L' & \Longrightarrow L\bar{l} & \text{if} \left\{ \begin{array}{l} Ll^\exists L' \text{ is inconsistent and} \\ l^\exists \text{ is the rightmost existential literal} \end{array} \right. \\
Fail & L & \Longrightarrow Unsat & \text{if} \left\{ \begin{array}{l} L \text{ is inconsistent and existential free} \end{array} \right. \\
Backtrack_\forall & Ll^\forall L' & \Longrightarrow L\bar{l} & \text{if} \left\{ \begin{array}{l} \text{no other rule applies except } Succeed \text{ and} \\ l^\forall \text{ is the rightmost universal literal} \end{array} \right. \\
Succeed & L & \Longrightarrow Valid & \text{if} \left\{ \begin{array}{l} \text{no other rule applies} \end{array} \right.
\end{array}
$$

Fig. 1. The transition rules of the QBF_F graph.

Q-DPLL with multiple layers. In the previous sub-section we presented the abstract solver that describes the Q-DPLL algorithm. Here we present an alternative solution that uses and extends the two-layers architecture employed in abstract solvers for disjunctive ASP [5]. In disjunctive ASP, solvers are mainly organized in the following way:[1] there is a "generate" layer that computes candidate solutions, and a "test" layer that checks whether it is indeed a solution, by checking minimality. In QBF, we extend this architecture by considering that a layer is the solving process "within" the same quantifier level. Within a layer, a SAT problem is solved, and depending from the past search, the current level, and the related quantifier type, the search is directed through levels with some newly added control states. In sum, there is a basic set of transition rules that corresponds to the rules for SAT (plus monotone rules, that can be used also in SAT), which are called Core rules, and another set of rules to direct the search. In the following, we will formalize these concepts.

[1] A notable exception is the family of ASP solvers based on translation.

Before introducing the related graph, we need some additional definitions. A stack of records S is a (possibly empty) list of records $S = L_1 :: L_2 :: \ldots :: L_k$ which we can also write as $S = S' :: L_k$, where S' is a stack of records.

An *oracle state* $L_{S,k,Q}$ is made of:

- a record L, which accounts for the current assignment computed;
- a stack of records S, an integer k and a quantifier type Q.

A *control state* $Instr(L_{S,k,Q})$ is made of:

- the action $Instr$ that led to this state, $Instr \in \{Failure, Success, Cont\}$;
- a record L, result of the last computation;
- a stack of records S, results of previous computations;
- an integer k equal to the amount of quantifier alternations that precede the quantifiers currently treated in the last computation;
- a type of quantifier Q in $\{\exists, \forall\}$, corresponding to the type of the quantifiers of the last computation.

Control states guide the search through layers. A state is a control state, an oracle state or a terminal state. Intuitively, the core rules from $Unit$ to $Backtrack$, which resemble the ones in the previous sub-section, deal with the computation within a level. Rules $Fail$ and $Succeed$, instead, are the rules at the interface between two layers, going from an oracle to a control state. Result processing rules, on the other hand, direct the computation according to the oracle state they are dealing with. In particular, $Fail$ (resp. $Succeed$) leads to the actions $Failure$ (resp. $Cont$) if the assignment can not (resp. can) be successfully extended at this level, respectively. In consequence of the application of one of these rules, Result processing rules can be triggered. A $Failure$ control state on an existential level triggers $Failure\exists$ and the result is that the action is unchanged, the level is decremented, the quantifier is changed, and the current record L' becomes the last in the stack. Then, $Failure\forall$ is immediately triggered doing similar processing, but leading to an oracle state whose record is inconsistent (and backtrack will be forced). The rational about this behavior is that if we had a failure on an existential level, we must also jump over the previous universal level, because in this branch a solution can not be found. If k becomes 0 and the stack is empty, we can return $Unsat$ through $FailureFinal$, meaning the F is unsatisfiable. Instead, a $Cont$ control state on an existential level triggers the rule $Continue$ that brings (i) to an oracle state at the next level whose assignment is added to the queue, the level is incremented, the quantifier is switched and the current assignment is restarted, or (ii) to a $Success$ state if the maximum level is reached through $FullAssign$. Now, $Success\exists$ is triggered and leads to failing level at the upper quantified level, in order to force a backtrack, or $Success\exists'$ is triggered if the record L at the upper level does contain decision literals: if this happens, the $Success$ state is maintained at the upper level, that immediately triggers $Success\forall$ that goes one further level up, and the search can proceed as before. If a $Success$ is reached at level 0, $Valid$ can be returned, meaning that F is satisfiable. The graph $QBF2_F$ has the states

as nodes, the transitions of Figure 2 as edges, and $\emptyset_{\emptyset,0,Q}$ as initial state, where Q is the type of the outermost quantifier.

Proposition 2. *Given a QBF formula F, the graph $QBF2_F$ verifies that F is satisfiable.*

Note that this graph seems to be more amenable for being the basis for building new abstract procedures for QBFs, which is one of the main advantage of this methodology. In fact, we can replace the Core rules with any other set of rules that solve the same problem (in this case a SAT problem), and add similar rules as *Fail* and *Succeed* that lead to Result processing rules.

Example 4. Consider the QBF (2) in Example 1, that we call F. A possible path in $QBF2_F$ is:

Initial state :	$\emptyset_{\emptyset,0,Q}$	*Backtrack*	$\Longrightarrow a_{\emptyset,0,\exists}$
Decide	$\Longrightarrow \overline{a}^{\exists}{}_{\emptyset,0,\exists}$	*Succeed*	$\Longrightarrow Cont(a_{\emptyset,0,\exists})$
Succeed	$\Longrightarrow Cont(\overline{a}^{\exists}{}_{\emptyset,0,\exists})$	*Continue*	$\Longrightarrow \emptyset_{a,1,\forall}$
Continue	$\Longrightarrow \emptyset_{\overline{a}^{\exists},1,\forall}$	*Decide*	$\Longrightarrow \overline{d}^{\exists}{}_{a,1,\forall}$
Decide	$\Longrightarrow \overline{d}^{\exists}{}_{\overline{a}^{\exists},1,\forall}$	*Succeed*	$\Longrightarrow Cont(\overline{d}^{\exists}{}_{a,1,\forall})$
Succeed	$\Longrightarrow Cont(\overline{d}^{\exists}{}_{\overline{a}^{\exists},1,\forall})$	*Continue*	$\Longrightarrow \emptyset_{a::\overline{d}^{\exists},2,\exists}$
Continue	$\Longrightarrow \emptyset_{\overline{a}^{\exists}::\overline{d}^{\exists},2,\exists}$	*Monotone*1	$\Longrightarrow b_{a::\overline{d}^{\exists},2,\exists}$
*Monotone*1	$\Longrightarrow b_{\overline{a}^{\exists}::\overline{d}^{\exists},2,\exists}$	*Succeed*	$\Longrightarrow Cont(b_{a::\overline{d}^{\exists},2,\exists})$
Succeed	$\Longrightarrow Cont(b_{\overline{a}^{\exists}::\overline{d}^{\exists},2,\exists})$	*FullAssign*	$\Longrightarrow Success(b_{a::\overline{d}^{\exists},2,\exists})$
FullAssign	$\Longrightarrow Success(b_{\overline{a}^{\exists}::\overline{d}^{\exists},2,\exists})$	*Success*$\exists$	$\Longrightarrow \overline{d}^{\exists}\bot_{a,1,\forall}$
Success$\exists$	$\Longrightarrow \overline{d}^{\exists}\bot_{\overline{a}^{\exists},1,\forall}$	*Backtrack*	$\Longrightarrow d_{a,1,\forall}$
Backtrack	$\Longrightarrow d_{\overline{a}^{\exists},1,\forall}$	*Unit*	$\Longrightarrow d\ b_{a,1,\forall}$
Unit	$\Longrightarrow d\ \overline{c}_{\overline{a}^{\exists},1,\forall}$	*Unit*	$\Longrightarrow d\ b\ \overline{b}_{a,1,\forall}$
Unit	$\Longrightarrow d\ \overline{c}\ \overline{b}_{\overline{a}^{\exists},1,\forall}$	*Fail*	$\Longrightarrow Failure(d\ b\ \overline{b}_{a,1,\forall})$
Unit	$\Longrightarrow d\ \overline{c}\ \overline{b}\ b_{\overline{a}^{\exists},1,\forall}$	*Failure*$\forall$	$\Longrightarrow a\bot_{\emptyset,0,\exists}$
Fail	$\Longrightarrow Failure(d\ \overline{c}\ \overline{b}\ b_{\overline{a}^{\exists},1,\forall})$	*Fail*	$\Longrightarrow Failure(a\bot_{\emptyset,0,\exists})$
Failure$\forall$	$\Longrightarrow \overline{a}^{\exists}\bot_{\emptyset,0,\exists}$	*FailureFinal*	$\Longrightarrow Unsat$

Solution Based on Variable Elimination. A third solution for solving a QBF is to rely on an approach which directly follows the semantic of a QBF as presented in the previous section, thus considering that, given a QBF F, $\exists aF$ is logically equivalent to $F_a \vee F_{\overline{a}}$, while $\forall aF$ is logically equivalent to $F_a \wedge F_{\overline{a}}$. The expansion of a variable a is obtained by[2]:

1. adding a variable b' for each variable b having $level(b) < level(a)$;
2. quantifying each variable b' in the same way as b and s.t. $level(b') = level(b)$;
3. for each clause C in the scope of a, adding a new clause C' obtained from C by substituting b' to b; and
4. considering the mentioned clause C (resp. C') , those containing $\overline{a}$ (resp. a) are eliminated, while a (resp. $\overline{a}$) is eliminated from the other clauses.

[2] Optimizations are possible, e.g. *Unit*, *Monotone*1 and *Monotone*2 are applicable, and a concept of "minimal scope" for a variable can be defined (see, e.g. [7]).

Core rules

$$UnitL_{S,k,Q} \quad \Longrightarrow Ll_{S,k,Q} \quad \text{if} \begin{cases} l \text{ does not occur in } L \text{ and} \\ \text{for some clause } C \text{ in the matrix,} \\ l \text{ occurs in } C \text{ and} \\ \text{each other unassigned literal of } C \text{ is universal and} \\ \text{each assigned literal of } C \text{ is contradicted} \end{cases}$$

$$Monotone1 \quad L_{S,k,Q} \Longrightarrow Ll_{S,k,Q} \quad \text{if} \begin{cases} \text{the variable of } l \text{ is existential and} \\ l \text{ occurs in the matrix and} \\ \bar{l} \text{ does not occur in the matrix} \end{cases}$$

$$Monotone2 \quad L_{S,k,Q} \Longrightarrow Ll_{S,k,Q} \quad \text{if} \begin{cases} \text{the variable of } l \text{ is universal and} \\ \bar{l} \text{ occurs in the matrix and} \\ l \text{ does not occur in the matrix} \end{cases}$$

$$Decide \quad L_{S,k,Q} \Longrightarrow Ll^{\exists}{}_{S,k,Q} \quad \text{if} \begin{cases} L \text{ is consistent and} \\ \text{neither } l \text{ nor } \bar{l} \text{ occur in } L \text{ and} \\ level(l) = k \end{cases}$$

$$Backtrack \quad Ll^{\exists}L'_{S,k,Q} \Longrightarrow L\bar{l}_{S,k,Q} \quad \text{if} \begin{cases} Ll^{\exists}L' \text{ is inconsistent and} \\ l^{\exists} \text{ is the rightmost decision literal} \end{cases}$$

$$Fail \quad L_{S,k,Q} \Longrightarrow Failure(L_{S,k,Q}) \quad \text{if} \begin{cases} L \text{ is inconsistent and decision free} \end{cases}$$

$$Succeed \quad L_{S,k,Q} \Longrightarrow Cont(L_{S,k,Q}) \quad \text{if} \begin{cases} \text{no other rule applies} \end{cases}$$

Result processing rules for $k \in \{0..max-1\}$

$$Continue \quad Cont(L_{S,k,Q}) \Longrightarrow \emptyset_{S::L,k+1,\overline{Q}}$$

$$FullAssign \quad Cont(L_{S,max,\exists}) \Longrightarrow Success(L_{S,max,\exists})$$

$$Failure\forall \quad Failure(L_{S::L',k+1,\forall}) \Longrightarrow L'\bot_{S,k,\exists}$$

$$Failure\exists \quad Failure(L_{S::L',k+1,\exists}) \Longrightarrow Failure(L'_{S,k,\forall})$$

$$FailureFinal \quad Failure(L_{\emptyset,0,Q}) \Longrightarrow Unsat$$

$$Success\forall \quad Success(L_{S::L',k+1,\forall}) \Longrightarrow Success(L'_{S,k,\exists})$$

$$Success\exists \quad Success(L_{S::L',k+1,\exists}) \Longrightarrow L'\bot_{S,k,\forall}$$
if L' contains at least a decision literal

$$Success\exists' \quad Success(L_{S::L',k+1,\exists}) \Longrightarrow Success(L'_{S,k,\forall})$$
if L' is decision-free

$$SuccessFinal \quad Success(L_{\emptyset,0,Q}) \Longrightarrow Valid$$

Fig. 2. The transition rules of the $QBF2_F$ graph.

In particular, the process of expanding all universally-quantified variables yields a SAT formula that can be solved with the abstract solver we have seen in Section 2, e.g. by expanding d in (2) we obtain the following SAT formula:

$$\exists abcb'c'\{\{b,c\},\{b,\overline{c}\},\{\overline{a},b'\},\{\overline{b'}\},\{b',c'\},\{a,\overline{c'}\}\}$$

where, e.g. variables b' and c' are added to the prefix (step 1. above) as existentially quantified (step 2.), $\{b',c'\}$ is obtained from $\{b,c\}$ (step 3.) and $\{b,\overline{c}\}$ is $\{d,b,\overline{c}\}$ deprived of d (step 4.). Formal properties for this approach

can be stated by relying on the correctness of variable elimination, and formal properties on the abstract graph for SAT.

Of course, a formal result similar to Proposition 1 and 2 could be added. But the correctness of this potential proposition stems directly from the correctness of variable elimination and of abstract solvers for SAT, that are already proved in other papers. Hence, stating such a proposition is not necessary.

4 Applications

Several reasoning tasks in other fields have been solved by a translation to a QBF formula followed by the application of a QBF solver to this formula. Hence, the abstract solvers we have defined can be used to abstract decision procedures for these problems. Indeed, the states and the transition rules from which the transitions are inferred will be identical to those of the graph QBF_F. But this process will be applied to a specific formula F that corresponds to the translation of an instance of the reasoning task to solve. We review here some applications of the abstract solvers we have defined in previous sections.

We will use the graphs of the type QBF_F, but these ideas could equivalently be stated with $QBF2_F$ graphs. Also, in the articles describing each of the considered solvers, where it is possible to find the resulting formulas, the matrix is not in CNF, and the formulas are not necessarily in prenex from. Hence, they are not defined exactly as we defined QBF formulas in this article. As a consequence, each time we define an abstract solver using a formula F we will use $QBF_{NF(F)}$. The function NF converts a non-prenex non-CNF quantified formula F to a QBF formula matching the definition provided in this article, i.e. with prenex form, all the free variables quantified existentially, and matrix in CNF.[3]

Answer Set Programming. Answer set programming is a declarative language representing problems as logic programs, of which solutions are called answer sets. Determining whether a program has any solution is Σ_2^P-complete in the general case of disjunctive answer set programming. In [9] is defined the quantified formula $\mathcal{T}_{lp}(P)$ for a program P. The formula $\mathcal{T}_{lp}(P)$ is satisfiable if and only if P has at least one answer set. Then, for any program P, the graph $QBF_{NF(\mathcal{T}_{lp}(P))}$ verifies that P has at least one answer set.

There is not yet practical ASP solvers built my means of reduction to QBF. The main obstacle is the encoding, and the fact that the defined encoding is in non-prenex non-CNF form, and an efficient transformation in prenex CNF is required. On the other hand, we believe that, once this obstacle is mitigated, our work can help also in the practical interplay between the encoding and the engine solver.

[3] Also the solvers that implement approaches based on reduction rely on such conversion, given they employ QBF solvers based on the form used in this article.

Abstract Argumentation Frameworks. Abstract argumentation frameworks are directed graphs which are designed to represent conflicting information. Each vertex of the graph is called an argument. The edges of the graph represent the way arguments can attack each other. The graphs are studied under varied semantics for which a set of arguments is a solution; for instance, in general a set of arguments will have to be conflict-free to be admissible. There are two main types of decision problems generally studied for each of the semantics: knowing whether a given argument belongs to at least a solution (i.e. credulous acceptance), and whether a given argument belongs to all solutions (i.e. skeptical acceptance). The article [14] defines first order formulas that have free variables, designed so that the assignments to the free variables that satisfy the formulas have a strict correspondence to the solutions of an argumentation framework under a given semantics. For instance, the first order formula $\mathcal{PE}(\mathcal{A})$ is defined in this article and satisfying assignments to its free variables correspond to the preferred semantics of the argumentation framework A. Obtaining an abstract procedure from such formulas is possible but would require using the techniques of [11], as explained below.

Abstract Dialectical Frameworks. Abstract dialectical frameworks are a generalization of abstract argumentation frameworks which allow to model more complex interactions between arguments. As a result, decision problems are generally one level higher in the polynomial hierarchy, up to the third level. We are here going to rely on the work of Diller et al. [11] which, for an argument $s \in S$ from an abstract dialectical framework (S, C) and a semantics $\sigma \in \{adm, com, prf, grd, mod, stb\}$, defines a QBF formula which is satisfiable if and only if s is skeptically accepted in (S, C) under σ semantics. A similar formula is defined in the case of credulous acceptance.

For instance, we focus on the case of preferred semantics and skeptical acceptance. The formula is denoted $\forall S_3(\mathcal{E}_{prf}(S, C) \Rightarrow s^{\oplus})$. Note that $\forall S_3$ refers to a quantification over all the elements from a set of variables built from S and used in $\mathcal{E}_{prf}(S, C)$, so that the quantification remains in the first order. Hence, for an abstract dialectical framework (S, C) and $s \in S$, the procedure of [11] for solving skeptical acceptance of s in (S, C) under preferred semantics is abstracted by $QBF_{NF(\forall S_3(\mathcal{E}_{prf}(S,C) \Rightarrow s^{\oplus}))}$. Hence $QBF_{NF(\forall S_3(\mathcal{E}_{prf}(S,C) \Rightarrow s^{\oplus}))}$ verifies that s is skeptically accepted in (S, C) under preferred semantics. Similar graphs can be defined so as to abstract the procedure for other semantics, and for credulous acceptance. For instance, credulous acceptance for preferred semantics would be abstracted by $QBF_{NF(\exists S_3(\mathcal{E}_{prf}(S,C) \wedge s^{\oplus}))}$. In $\forall S_3(\mathcal{E}_{prf}(S, C) \Rightarrow s^{\oplus})$, the formula $\mathcal{E}_{prf}(S, C)$ is very similar to $\mathcal{PE}(\mathcal{A})$ in its function. Using a formula similar to $s^{\oplus}$ which represents the argument s and a set of variables similar to S_3, one could define a formula of the form $\forall S_3(\mathcal{PE}(\mathcal{A}) \Rightarrow s^{\oplus})$ so as to abstract a decision procedure for abstract argumentation frameworks. Assuming we call the obtained formula $pe(A, s)$, the graph $QBF_{pe(A,s)}$ would verify that s is skeptically accepted in A under preferred semantics.

Differently from ASP, efficient approaches based on QBF are implemented for Abstract Dialectical Frameworks [11]. We believe that, in this case, abstract solvers can help to more deeply understand the effectiveness of this approach.

5 Related Work and Conclusions

Abstract solvers have been originally employed in [1] first to describe the DPLL algorithm for SAT, and then by extending this graph to deal with certain SMT logics which can be solved by means of a lazy (i.e. SAT-based) approach to SMT solving. Then, abstract solvers have been applied to Answer Set Programming in several papers. Abstract solvers for backtracking-based ASP solvers for non-disjunctive ASP programs (whose complexity is within the first level of the polynomial hierarchy) have been presented in [15], then extended in (*i*) [3] to include backjumping and learning techniques, and in (*ii*) [5] for describing solvers for disjunctive ASP programs (able to express problems up to the second level of the polynomial hierarchy). Another contribution in ASP is presented in [4], where an unifying perspective based on completion (i.e. transforming a logic program into a propositional formula) on some solvers for non-disjunctive ASP programs is given. Finally, abstract solvers for Constraint ASP solvers are presented in [6].

In this paper we have presented, for the first time, abstract procedures for solving reasoning tasks whose complexity is beyond the second level of the polynomial hierarchy, i.e. the satisfiability of QBF, the prototypical PSPACE-complete problem. We have finally shown how these abstract solvers can be used to define abstract procedures for certain reasoning tasks in other fields that can be solved via a translation to a QBF. Other applications are of course possible, e.g. to solving conformant and conditional automated planning problems. However, we do not claim about the efficiency of a new tool built on this basis, given that it usually also requires many iterations of theoretical analysis, practical engineering, and domain-specific optimizations to develop efficient systems. Yet, positive experiences have been already reported for Abstract Dialectial Frameworks [11] and classical planning [16].

Future research includes adding optimization techniques, like backjumping and learning, to our abstract solvers. These are two well-known techniques implemented in several solvers: backjumping is the ability to jump over decision literals that were not directly responsible to the conflict that caused backtracking, while learning adds information (in terms of, e.g. clauses) to the initial formula in order to avoid to follow similar paths in order parts of the search. The presence of two quantifier types enables two different types of transition rules for modeling these techniques, that will be anyway added by means of modular addition of transition rules.

References

1. Leeuwenhoek, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). Journal of the ACM **53**(6), 937–977 (2006)
2. Barrett, C., Shikanian, I., Tinelli, C.: An abstract decision procedure for a theory of inductive data types. JSAT **3**(1–2), 21–46 (2007)
3. Lierler, Y.: Abstract answer set solvers with backjumping and learning. Theory and Practice of Logic Programming **11**, 135–169 (2011)
4. Lierler, Y., Truszczynski, M.: Transition systems for model generators - a unifying approach. Theory and Practice of Logic Programming **11**(4–5), 629–646 (2011)
5. Brochenin, R., Lierler, Y., Maratea, M.: Abstract disjunctive answer set solvers. In: Schaub, T., Friedrich, G., O'Sullivan, B. (eds.) Proc. of ECAI 2014, vol. 263. Frontiers in Artificial Intelligence and Applications, pp. 165–170. IOS Press (2014)
6. Lierler, Y.: Relating constraint answer set programming languages and algorithms. Artificial Intelligence **207**, 1–22 (2014)
7. Giunchiglia, E., Marin, P., Narizzano, M.: Reasoning with quantified boolean formulas. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability. Volume 185 of FAIA, pp. 761–780. IOS Press (2009)
8. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Communications of the ACM **5**(7), 394–397 (1962)
9. Egly, U., Eiter, T., Tompits, H., Woltran, S.: Solving advanced reasoning tasks using quantified boolean formulas. In: Kautz, H.A., Porter, B.W. (eds.) Proc. of AAAI 2000, pp. 417–422, AAAI Press / The MIT Press (2000)
10. Arieli, O., Caminada, M.W.A.: A QBF-based formalization of abstract argumentation semantics. Journal of Applied Logic **11**(2), 229–252 (2013)
11. Diller, M., Wallner, J.P., Woltran, S.: Reasoning in abstract dialectical frameworks using quantified boolean formulas. In: Parsons, S., Oren, N., Reed, C., Cerutti, F. (eds.) Proc. of COMMA 2014. Volume 266 of FAIA, pp. 241–252, IOS Press (2014)
12. Cadoli, M., Giovanardi, A., Schaerf, M.: An algorithm to evaluate quantified boolean formulae. In: Mostow, J., Rich, C. (eds) Proc. of AAAI 1998, pp. 262–267, AAAI Press / The MIT Press (1998)
13. Giunchiglia, E., Narizzano, M., Tacchella, A.: Clause/term resolution and learning in the evaluation of quantified boolean formulas. JAIR **26**, 371–416 (2006)
14. Charwat, G., Dvořák, W., Gaggl, S.A., Wallner, J.P., Woltran, S.: Methods for Solving Reasoning Problems in Abstract Argumentation - A Survey. Artificial Intelligence **220**, 28–63 (2015)
15. Lierler, Y.: Abstract answer set solvers. In: Garcia de la Banda, M., Pontelli, E. (eds.) ICLP 2008. LNCS, vol. 5366, pp. 377–391. Springer, Heidelberg (2008)
16. Cashmore, M., Fox, M., Giunchiglia, E.: Partially grounded planning as quantified boolean formula. In: Borrajo, D., Kambhampati, S., Oddi, A., Fratini, S. (eds.) Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, AAAI (2013)