Ann Math Artif Intell (2017) 80:21–45
DOI 10.1007/s10472-016-9501-2

Conformant planning as a case study of incremental QBF solving

Uwe Egly[1] · Martin Kronegger[1] · Florian Lonsing[1] · Andreas Pfandler[1,2]

Published online: 24 March 2016

Abstract We consider planning with uncertainty in the initial state as a case study of *incremental* quantified Boolean formula (QBF) solving. We report on experiments with a workflow to incrementally encode a planning instance into a sequence of QBFs. To solve this sequence of successively constructed QBFs, we use our *general-purpose incremental* QBF solver DepQBF. Since the generated QBFs have many clauses and variables in common, our approach avoids redundancy both in the encoding phase as well as in the solving phase. We also present experiments with incremental preprocessing techniques that are based on blocked clause elimination (QBCE). QBCE allows to eliminate certain clauses from a QBF in a satisfiability preserving way. We implemented the QBCE-based techniques in DepQBF in three variants: as preprocessing, as inprocessing (which extends preprocessing by taking into account variable assignments that were fixed by the QBF solver), and as a novel dynamic approach where QBCE is tightly integrated in the solving process. For DepQBF, experimental results show that incremental QBF solving with incremental QBCE outperforms incremental QBF solving without QBCE, which in turn outperforms

✉ Florian Lonsing
florian.lonsing@tuwien.ac.at

Uwe Egly
uwe.egly@tuwien.ac.at

Martin Kronegger
martin.kronegger@tuwien.ac.at

Andreas Pfandler
andreas.pfandler@tuwien.ac.at

[1] Institute of Information Systems, TU Wien, Vienna, Austria

[2] School of Economic Disciplines, University of Siegen, Siegen, Germany

nonincremental QBF solving. For the first time we report on incremental QBF solving with incremental QBCE as inprocessing. Our results are the first empirical study of *incremental* QBF solving in the context of planning and motivate its use in other application domains.

Keywords Quantified Boolean formulas (QBFs) · Conformant planning · Incremental solving · Preprocessing · Blocked clause elimination

Mathematics Subject Classifications (2010) 68T15 · 68T20 · 68T27

1 Introduction

Many workflows in formal verification and model checking rely on certain logics as languages to model verification conditions or properties of the systems under consideration. Examples for such logics are propositional logic (SAT), quantified Boolean formulas (QBFs), and decidable fragments of first order logic in terms of satisfiability modulo theories (SMT). A tight integration of decision procedures to solve formulas in these logics is crucial for the overall performance of the workflows in practice.

In the context of SAT, *incremental solving* has become a state-of-the-art approach [1, 12, 30, 40]. Given a sequence of related propositional formulas $S = \langle \phi_0, \phi_1, \ldots, \phi_n \rangle$ an incremental SAT solver reuses information that was gathered when solving ϕ_i in order to solve the next formula ϕ_{i+1}. Since this incremental approach avoids some redundancy in the process of solving the sequence S, it is desirable to integrate incremental solvers in practical workflows. In contrast, in *nonincremental solving* the solver does not keep any information from previously solved formulas and always starts from scratch.

QBFs are an extension of propositional formulas which allow for explicit universal ($\forall$) and existential ($\exists$) quantification over Boolean variables. The problem of checking the satisfiability of QBFs is PSPACE -complete. We consider QBFs as a natural modelling language for planning problems with uncertainty in the initial state. In *conformant planning* we are given a set of state variables over a specified domain, a set of actions (each action with a precondition and an effect), an initial state where some values of the variables may be unknown, and a specification of the goal. The task is to find a sequence of actions, i.e., a plan, that leads from the initial state to a state where the goal specification is satisfied. Many natural problems, such as repair and therapy planning [47], can be formulated as conformant planning problems. When restricted to plans of length polynomial in the input size this form of planning is Σ_2P -complete [3], whereas classical planning with complete information is NP -complete. Therefore, transforming conformant planning problems to QBFs is a natural approach. Rintanen [43] presented such transformations. It is also possible to transform an instance of conformant planning to propositional logic (SAT), for example. However, QBF encodings are potentially exponentially more succinct than SAT encodings because universal quantifiers have to be flattened in the SAT encodings. The compactness of QBF encodings has been illustrated in the context of classical planning problems [10].

Recently, Kronegger et al. [28] showed that solving a conformant planning instance by transformation into a sequence of QBFs can be competitive. In this approach, they generated a QBF for every plan length under consideration and invoked an external QBF solver on each generated QBF. However, the major drawback is that the QBF solver cannot reuse information from previous runs. All information necessary to solve the QBF has to be learned again because the QBF solver is called in a nonincremental way.

In this work we present an *incremental* workflow to solve planning problems with uncertainty in the initial state which tightly integrates a *general-purpose incremental QBF solver*. To obtain a better picture of the performance gain through the incremental approach, we implemented the workflow in a planning tool and perform a case study where we compare incremental QBF solving and nonincremental QBF solving on benchmarks for conformant planning. Notice that the basic workflow as implemented by our planning tool is not limited to conformant planning problems but also applies to arbitrary reachability problems like bounded model checking (BMC) [39].

The **main contributions** of this work are as follows.[1]

- *Planning tool.* We present a planning tool based on the transformation of planning instances with uncertainty in the initial state to QBFs. This tool implements an incremental and exact approach, i.e., it is guaranteed to find a plan whenever a plan exists and – if successful – it returns a plan of *minimal* length. Furthermore, our tool allows for the use of arbitrary QBF preprocessors and (incremental) QBF solvers.
- *QBF solving.* We apply the general-purpose incremental QBF solver DepQBF[2] [34, 35] in the workflow implemented by the planning tool. DepQBF is a search-based QBF solver with clause and cube learning [15, 31, 36, 50]. We have integrated incremental preprocessing techniques into DepQBF to combine preprocessing and incremental solving. For this purpose, we have implemented a nonincremental and an incremental version of blocked clause elimination for QBF (QBCE) [6, 17]. QBCE allows to eliminate certain clauses from a QBF in a satisfiability preserving way. The nonincremental and incremental version of QBCE is applied for preprocessing and inprocessing, which extends preprocessing by taking into account variable assignments that were fixed by the QBF solver. Additionally, QBCE is applied in a novel dynamic approach where it is tightly coupled with the solving process [32]. These techniques are explained in Section 2. Apart from DepQBF, we have integrated Nenofex [33] and RAReQS [23] as additional nonincremental QBF solvers and Bloqqer [6] as a QBF preprocessor into our planning tool.
- *Experimental evaluation.* We evaluate the performance of the incremental and the nonincremental QBF-based approach to solve planning instances with uncertainty in the initial state. For that matter, we rely on the nonincremental QBF solvers Nenofex, RAReQS, and incremental and nonincremental variants of DepQBF. For the nonincremental approach we also analyze the performance of the QBF solvers when the QBF preprocessor Bloqqer is applied prior to solving. For both the incremental and nonincremental approach we analyze the performance of DepQBF when combined with QBCE as a preprocessing and inprocessing technique. We also evaluate the novel dynamic application of QBCE [32]. In addition, we briefly report on experiments with heuristic approaches to solve the considered planning problems.

This article is organized as follows. In Section 2 we give an overview on QBF solving, preprocessing by QBCE, and incremental QBF solving together with incremental QBCE. In Section 3 we briefly describe the necessary background on conformant planning and the two benchmark types we used in our experimental evaluation. Then, in Section 4 we discuss our planning tool that takes planning instances as input and encodes them as a sequence of QBFs. In Section 5 we report in detail on the experimental evaluation of our approach. Finally, in Section 6 we conclude and give directions for future work.

[1] This article is an extended and enhanced version of the conference article [13].

[2] DepQBF is free software: http://lonsing.github.io/depqbf/.

2 QBF solving and preprocessing

We focus on quantified Boolean formulas (QBFs) $\psi = \hat{Q}.\phi$ in *prenex conjunctive normal form (PCNF)*. All quantifiers occur in the *prefix* $\hat{Q} = Q_1 B_1 \ldots Q_n B_n$ and the *matrix* ϕ is a quantifier-free propositional formula in CNF. A CNF consists of a conjunction of clauses. A *clause* (*cube*) is a disjunction (conjunction) of literals. For simplicity, we identify a clause (cube) as a set of literals. A *literal* is either a variable x or a negated variable $\bar{x}$. The negation of a literal l is denoted by $\bar{l}$. A clause C is *tautological* if $x \in C$ and $\bar{x} \in C$ for some variable x. Let $i, j \in \{1, \ldots, n\}$ and $1 \leq k \leq n-1$. The prefix consists of quantifiers $Q_i \in \{\forall, \exists\}$ where $Q_k \neq Q_{k+1}$ and pairwise disjoint sets B_i of Boolean variables, where $Q_i B_i$ is called a *quantifier set*. For simplicity, we omit parentheses when we write quantifier sets. The prefix gives rise to a linear ordering of the variables: we define $x \leq y$ if $x \in B_i$, $y \in B_j$ and $i \leq j$. A QBF $\psi = \hat{Q}.\phi$ is *closed* if all variables which occur in the matrix ϕ also occur in the prefix $\hat{Q}$. We consider only closed QBFs.

Given a QBF ψ, an *assignment* is a mapping from the variables in ψ to truth values *true* or *false*. We identify an assignment $A := \{l_1, \ldots, l_n\}$ as a set of literals l_i such that if some variable x is assigned *true* then $l_i \in A$ and $l_i = x$, and if x is assigned *false* then $l_i \in A$ and $l_i = \bar{x}$. Given a QBF ψ and an assignment A, the *QBF ψ under A*, denoted by $\psi[A]$, is obtained by replacing every occurrence of a variable x in ψ such that $l = x$ ($l = \bar{x}$) and $l \in A$ by the truth constant $\top$ ($\bot$). Simplifications with respect to truth constants may result in a new matrix without truth constants and in a smaller quantifier prefix.

The semantics of QBFs is defined recursively based on the quantifier types and the prefix ordering of the variables. In a semantical evaluation, a QBF ψ is split into subcases. The QBF consisting only of the truth constant *true* ($\top$) or *false* ($\bot$) is satisfiable or unsatisfiable, respectively. The QBF $\psi = \forall x.\, \psi'$ with the universal quantification $\forall x$ at the leftmost position in the prefix is satisfiable if the subcases $\psi[\{\bar{x}\}]$ and $\psi[\{x\}]$ are satisfiable. The QBF $\psi = \exists x.\, \psi'$ with the existential quantification $\exists x$ is satisfiable if at least one of the subcases $\psi[\{\bar{x}\}]$ or $\psi[\{x\}]$ is satisfiable.

Search-based QBF solving [9] is a generalization of the *DPLL* algorithm [11] for SAT. DPLL solves a propositional formula by systematically splitting it into subcases based on assignments to the variables. Search-based QBF solvers implement a QBF-specific variant of *conflict-driven clause learning (CDCL)* for SAT, called *QCDCL* [15, 31, 36, 50]. In QCDCL as implemented in DepQBF, assignments A to the variables in a given closed QBF ψ are successively generated. In general, the variables have to be assigned starting from the left end of the quantifier prefix. After assignments have been made, the QBF $\psi[A]$ is analyzed.

The case where $\psi[A] = \bot$ constitutes an unsatisfiable subcase, also called a *conflict*. The subcase is analyzed and a new clause C, called a *learned clause*, is derived from $\psi[A]$ by *Q-resolution* [27]. Q-resolution is a variant of *resolution* for propositional logic which takes the quantification of variables into account and which combines resolution and *universal reduction*, defined as follows. Given two nontautological clauses C_1 and C_2 such that $l \in C_1, \bar{l} \in C_2$, and the variable of l is existential, the *resolvent* of C_1 and C_2 is the nontautological clause $C := (C_1 \setminus \{l\}) \cup (C_2 \setminus \{\bar{l}\})$. The variable of l is the *pivot variable*. If C is tautological then by definition no resolvent of C_1 and C_2 exists. Given a clause $C = C' \vee l$ where the variable $var(l)$ of l is universal and for all variables $var(l')$ of existential literals $l' \in C'$ it holds that $var(l') \leq var(l)$, *universal reduction* of C results in the clause C'. For simplicity, we consider a clause a *resolvent* if it was derived by a single application of resolution followed by universal reduction. The actual selection of the clauses in ψ to be

resolved in order to produce a learned clause C depends on the current assignment A. A learned clause C is added conjunctively to ψ.

The case where $\psi[A] = \top$ constitutes a satisfiable subcase, also called a *solution* because every clause in ψ is satisfied by A. A cube $C = (\bigwedge_{l \in A} l)$ comprising the literals in A is constructed and added disjunctively to ψ as a learned cube. Hence the set of learned cubes (clauses) appear in a formula in disjunctive (conjunctive) normal form. Dual to the derivation of clauses, Q-resolution is applied to derive new cubes from previously learned ones, where pivot variables must be universally quantified. Dual to universal reduction, *existential reduction* allows to remove existential literals from cubes which are maximal with respect to the prefix ordering.

After a learned clause or cube has been added to ψ, assignments are retracted by backtracking. Assignment generation proceeds until the next (un)satisfiable subcase is encountered. QCDCL terminates if an empty clause or cube is derived. QCDCL derives an empty clause (cube) if and only if the given QBF is unsatisfiable (satisfiable).

Example 1 (Adapted from Example 1 in [32]) Consider the QBF ψ with prefix $\exists z, z' \forall u \exists y$ and matrix $(u \vee \bar{y}) \wedge (\bar{u} \vee y) \wedge (z \vee u \vee \bar{y}) \wedge (z \vee u \vee y) \wedge (z' \vee \bar{u} \vee y) \wedge (\bar{z} \vee \bar{u} \vee \bar{y}) \wedge (\bar{z}' \vee u \vee y)$.

We solve ψ by QCDCL. Suppose that all the variables in ψ are assigned to true following the prefix ordering, resulting in the assignment $A := \{z, z', u, y\}$. The clause $(\bar{z} \vee \bar{u} \vee \bar{y})$ is falsified under assignment A, which constitutes an unsatisfiable subcase. In the course of conflict-driven clause learning, resolving the clauses $(\bar{z} \vee \bar{u} \vee \bar{y})$ and $(\bar{u} \vee y)$ by Q-resolution produces the learned unit clause $(\bar{z})$. The learned unit clause $(\bar{z})$ can only be satisfied by assigning variable z to false. During backtracking, all assignments in A are retracted and the search process starts over with the new assignment $A := \{\bar{z}\}$.

Assume that A is further extended to $A := \{\bar{z}, \bar{z}', \bar{u}, \bar{y}\}$ by following the prefix ordering. The clause $(z \vee u \vee y)$ is falsified under A. The learned clause (z) is produced by resolving the falsified clause $(z \vee u \vee y)$ and the clause $(u \vee \bar{y})$. Further, the learned clause (z) is resolved with the previously learned clause $(\bar{z})$, resulting in the empty clause and proving that ψ is unsatisfiable.

The purpose of the clauses and cubes learned in QCDCL is to prune the search space and hence speed up the search. Additionally, QCDCL-based QBF solvers can produce *Q-resolution proofs of the (un)satisfiability* of a given QBF. A Q-resolution proof Π consists of all the Q-resolution steps involved in the generation of clauses or cubes needed to derive the empty clause or cube. This allows us to verify the result of a QBF solver independently [49].

Given a Q-resolution proof Π of the unsatisfiability of a QBF ψ, a *countermodel* [2] or *strategy* [16] can be extracted from Π in terms of a set of *Herbrand functions*. Intuitively, an Herbrand function $g_y(x_1, \ldots, x_n)$ represents the values that a *universal* variable y must take to *falsify* ψ with respect to the values of all *existential* variables $x_1, \ldots, x_n$ with $x_i < y$ in the prefix ordering. Given an unsatisfiable QBF ψ, the process of *Herbrandization* of ψ replaces the universal variables in ψ by their respective Herbrand functions and results in an unsatisfiable propositional formula ψ' not necessarily in CNF and containing only existential variables.

Example 2 (Continues Example 1) Consider the QBF ψ from Example 1 and the derivation of the empty clause in terms of resolving the learned clauses $C := (\bar{z})$ and $C' := (z)$. The learned clause C was obtained by resolving the clauses $(\bar{z} \vee \bar{u} \vee \bar{y})$ and $(\bar{u} \vee y)$ and the learned clause C' by resolving the clauses $(z \vee u \vee y)$ and $(u \vee \bar{y})$, respectively. The Q-resolution

steps needed to derive the learned clauses up to the empty clause constitute a Q-resolution proof Π of the unsatisfiability of ψ.

By inspecting Π it is possible to construct Herbrand functions for the universal variables in ψ. We refer to the literature [2] for details on the construction of Herbrand functions. Suppose that we have constructed the Herbrand function $g_u(z, z') := z$ for the single universal variable u in ψ. Note that $g_u(z, z')$ technically depends on both existential variables z and z' which are smaller than u in the prefix ordering. However, actually $g_u(z, z')$ depends only on z since z' does not occur in its definition.

Herbrandization replaces all occurrences of u in ψ by $g_u(z, z')$, resulting in the propositional formula $\psi' := \exists z, z', y.(z \vee \bar{y}) \wedge (\bar{z} \vee y) \wedge (z \vee y) \wedge (z' \vee \bar{z} \vee y) \wedge (\bar{z} \vee \bar{y}) \wedge (\bar{z}' \vee z \vee y)$, which contains only existential variables. The formula ψ' is unsatisfiable, which shows that $g_u(z, z')$ is indeed a correct Herbrand function, thus confirming the unsatisfiability of the original QBF ψ.

In order to check the correctness of Herbrand functions as illustrated in Example 2, the satisfiability of the propositional formula ψ' resulting from Herbrandization must be checked, which is an NP -complete problem.

Dual to Q-resolution proofs of unsatisfiability and Herbrand functions, *Skolem functions* can be extracted from a Q-resolution proof Π of the *satisfiability* of a QBF ψ. A Skolem function $f_y(x_1, \ldots, x_n)$ represents the values that an *existential* variable y must take to *satisfy* ψ with respect to the values of all *universal* variables $x_1, \ldots, x_n$ with $x_i < y$ in the prefix ordering. Dual to Herbrandization, *Skolemization* of ψ replaces the existential variables in ψ by their respective Skolem functions and results in a satisfiable QBF ψ' not necessarily in CNF and containing only universal variables.

Example 3 (Taken from Example 1 in [32]) Consider the QBF ψ with prefix $\exists z, z' \forall u \exists y$ and matrix $(u \vee \bar{y}) \wedge (\bar{u} \vee y) \wedge (z \vee u \vee \bar{y}) \wedge (z' \vee \bar{u} \vee y) \wedge (\bar{z} \vee \bar{u} \vee \bar{y}) \wedge (\bar{z}' \vee u \vee y)$. We solve ψ by QCDCL. Suppose that assignment $A := \{\bar{z}, \bar{z}', \bar{u}, \bar{y}\}$ has been enumerated. All clauses are satisfied under A, which corresponds to a solution, and hence the cube $C_0 = (\bar{z} \wedge \bar{z}' \wedge \bar{u} \wedge \bar{y})$ is learned. Likewise, the cube $C_1 = (\bar{z} \wedge \bar{z}' \wedge u \wedge y)$ is learned given assignment $A := \{\bar{z}, \bar{z}', u, y\}$, which is also a solution. By existential reduction, the cubes $C_2 = (\bar{z} \wedge \bar{z}' \wedge \bar{u})$ and $C_3 = (\bar{z} \wedge \bar{z}' \wedge u)$ are derived from C_0 and C_1, respectively. Resolving the cubes C_2 and C_3 produces the cube $C_4 = (\bar{z} \wedge \bar{z}')$. Finally, applying existential reduction to C_4 results in the empty cube. The Q-resolution steps needed to derive the learned cubes up to the empty cube constitute a Q-resolution proof Π of the satisfiability of ψ.

Similar to Example 2, Skolem functions for the existential variables in ψ may be constructed by inspecting Π [2]. Suppose that we have constructed Skolem functions $f_z := \bot$, $f_{z'} := \bot$, and $f_y(u) := u$ for z, z', and y in ψ. Skolemization replaces all occurrences of existential variables in ψ by their respective Skolem function, resulting in the QBF $\psi' := \forall u.(u \vee \bar{u}) \wedge (\bar{u} \vee u) \wedge (\bot \vee u \vee \bar{u}) \wedge (\bot \vee \bar{u} \vee u) \wedge (\top \vee \bar{u} \vee \bar{u}) \wedge (\top \vee u \vee u)$. Further simplification reduces ψ' to $\top$, showing that ψ' is satisfiable. Hence, f_z, $f_{z'}$, and $f_y(u)$ are correct Skolem functions, which confirms that the original QBF ψ is satisfiable.

In order to check the correctness of Skolem functions as illustrated in Example 3, the satisfiability of the QBF ψ' resulting from Skolemization must be checked, a problem which is in the complexity class co-NP .

Skolem and Herbrand functions provide a more detailed explanation of the (un)satisfiability of a QBF than Q-resolution proofs. The functions express concrete values the variables must take whereas Q-resolution proofs indicate only (un)satisfiability.

To encode an instance of conformant planning as a QBF, we build upon the work of Rintanen [43]. We encode the problem whether a plan of a particular length k exists as a QBF $\psi = \exists B_1 \forall B_2 \exists B_3 . \phi$ with a prefix having two quantifier alternations. Notice that, although the computational complexity of this planning problem is on the second level of the polynomial hierarchy, we use a more natural encoding that simplifies the PCNF transformation which is on the third level of the polynomial hierarchy. If ψ is satisfiable then a plan of length k can be extracted from the assignments to the variables in the leftmost existential quantifier set $\exists B_1$. To this end, however, it is not necessary to explicitly produce Q-resolution proofs and Skolem functions from a run of QCDCL on ψ. The Skolem function f_y of a variable y in B_1 does not depend on any universal variables and hence has an arity of zero (as illustrated in Example 3). For this special case, the value of f_y is a truth constant and can be computed by a QCDCL-based QBF solver during a run and be represented in the QDIMACS output format.[3] The correctness of the extracted plan can be checked by *partial* Skolemization of ψ based on the constant Skolem functions of the variables in B_1, which must result in a satisfiable QBF $\psi' = \forall B_2 \exists B_3 . \phi'$ in PCNF containing universal and existential variables. Checking the correctness of the constant Skolem functions amount to checking the satisfiability of ψ', which is Π_2P-complete.

Example 4 (Continues Example 3) Consider the QBF ψ with prefix $\exists z, z' \forall u \exists y$ and matrix $(u \vee \bar{y}) \wedge (\bar{u} \vee y) \wedge (z \vee u \vee \bar{y}) \wedge (z' \vee \bar{u} \vee y) \wedge (\bar{z} \vee \bar{u} \vee \bar{y}) \wedge (\bar{z}' \vee u \vee y)$ from Example 3 and the constant Skolem functions $f_z := \bot$ and $f_{z'} := \bot$ for the existential variables z and z' from the leftmost existential quantifier set. Partial Skolemization with respect to f_z and $f_{z'}$ and simplification results in the satisfiable QBF $\psi' := \forall u \exists y . (u \vee \bar{y}) \wedge (\bar{u} \vee y)$.

Extracting constant Skolem functions of variables in the leftmost existential quantifier set only amounts to partial certification of the satisfiability of ψ. Partial certification is appealing for QBF-based workflows as it avoids to generate Q-resolution proofs. In practice, the generation of Skolem functions of arbitrary existential variables may be prohibitive in terms of run time and memory footprint [41]. Moreover, partial certification is compatible with preprocessing and incremental solving.

Checking the correctness of constant Skolem functions as illustrated in Example 4 is PSPACE -complete in general if the number of quantifier alternations in ψ' resulting from partial Skolemization is unbounded. However, a Q-resolution proof Π' of the satisfiability of ψ' may be obtained from a Q-resolution proof Π of the original QBF ψ and constant Skolem functions f_y of existential variables y in the leftmost quantifier set of ψ by interpreting Π under f_z [16].

2.1 Expansion-based QBF solving

In contrast to search-based QBF solving, expansion-based solvers successively eliminate variables in the formula (e.g., [5, 8, 33]). In general, variables are eliminated from right to left with respect to the prefix ordering. In the following, we briefly illustrate the idea of eliminating variables by expansion.

Let $\psi := Q_1 x_1 \ldots Q_{n-1} x_{n-1} Q_n x_n . \phi$ be a QBF where $Q_i \in \{\forall, \exists\}$, x_i are variables, and ϕ is a quantifier-free propositional formula not necessarily in CNF. If the rightmost variable x_n is existentially quantified, i.e., $Q_n = \exists$, then x_n is expanded by replacing the original

[3] QDIMACS output format definition: http://www.qbflib.org/qdimacs.html.

QBF $\psi := Q_1 x_1 \ldots Q_{n-1} x_{n-1} Q_n x_n.\phi$ by $Q_1 x_1 \ldots Q_{n-1} x_{n-1}.(\phi[\bar{x}_n] \vee \phi[x_n])$. In the two copies $\phi[\bar{x}_n]$ and $\phi[x_n]$ of ϕ, all occurrences of x_n are replaced by the truth constants $\bot$ and $\top$, respectively. Expansion of variables may stop early as soon as the formula after expansion reduces to either $\bot$ or $\top$ under Boolean simplifications, meaning that the original QBF ψ is unsatisfiable or satisfiable, respectively.

If the rightmost variable x_n is universally quantified, i.e., $Q_n = \forall$, then x_n is expanded by joining the two copies of ϕ conjunctively ($\wedge$). Expansion can be generalized to variables which are not rightmost, which requires to duplicate certain variables, add new quantifier sets, and rename variables in one copy of the formula.

In the worst case, the size of the formula doubles each time a variable is expanded. In practice, expansion-based solvers like Nenofex [33] and RAReQS [23], which we consider in our experimental study in Section 5, apply sophisticated techniques like integrated SAT solving, redundancy removal, and abstraction to limit the increase of the formula size.

From a theoretical point of view, expansion of variables is different from Q-resolution, which is applied in QCDCL. On certain classes of QBFs, expansion allows for proofs which are exponentially more succinct than any Q-resolution proof, and vice versa [4, 24]. Hence solvers based on variable expansion and QCDCL have individual strengths depending on the QBFs to be solved.

2.2 Preprocessing by blocked clause elimination

Preprocessing has been found crucial for the performance of QBF-based workflows (see, e.g., the results of the QBF Gallery 2013 and 2014).[4] Preprocessing transforms a given QBF ψ into a QBF ψ' by adding or removing clauses or variables in a satisfiability preserving way. The goal of preprocessing is to speed up solving ψ' (compared to ψ) when also taking the time for preprocessing into account. Techniques applied for preprocessing include failed literal detection, variable expansion, equivalence reasoning, and variable elimination [5, 8, 14, 45, 48]. In contrast to expansion-based QBF solving as outlined in the previous section, in the context of preprocessing variable expansion is applied only in restricted fashion to avoid an increase of the formula size.

In our incremental QBF-based workflow to solve conformant planning problems, we focus on *blocked clause elimination (QBCE)* [6, 17] as a QBF preprocessing technique. Among other techniques, QBCE is implemented in the QBF preprocessor Bloqqer.[5]

Definition 1 (**Blocked clause** [6, 17]) Let $\psi = \hat{Q}.\phi$ be a QBF and $C \in \phi$ be a clause. An existential literal $l \in C$ is a *blocking literal* if for all clauses $C' \in \phi$ with $\bar{l} \in C'$, a literal l' with $l' \leq l$ exists such that $l', \bar{l}' \in C \cup (C' \setminus \{\bar{l}\})$. A clause is *blocked* if it contains a blocking literal.

Note that blocking literal l in Definition 1 must be existential whereas literals $l', \bar{l}'$ can be existential or universal. An *informal* definition of blocked clauses based on resolution is as follows. Consider an existential literal l in a clause C of a QBF and all clauses C' containing the literal $\bar{l}$. Let $C'' := (C \setminus \{l\}) \cup (C' \setminus \{\bar{l}\})$ be the potential resolvent of C and C' using the variable of l as the pivot variable. Then l is a blocking literal in C if, for all C', (1) the clause C'' contains complementary literals l' and $\bar{l}'$ (hence C'' is not a resolvent by definition) and

[4] http://www.kr.tuwien.ac.at/events/qbfgallery2013/ and http://qbf.satisfiability.org/gallery/.

[5] http://fmv.jku.at/bloqqer/.

(2) the literals l' and $\bar{l}'$ are smaller than l according to the prefix ordering. Thus, checking whether $l \in C$ is a blocking literal amounts to inspecting all potential resolvents that can be generated by selecting the variable of l as pivot variable. Originally, blocked clauses were presented in the context of extended resolution and propositional logic [29].

Definition 2 (**Blocked clause elimination** [6, 17]) Given a QBF ψ, *blocked clause elimination (QBCE)* produces the QBF ψ' by removing all blocked clauses from ψ until completion.

QBCE has a unique fixpoint, preserves unsatisfiability and can be carried out in time which is polynomial in the size of ψ [6]. The following example illustrates the elimination of blocked clauses by QBCE.

Example 5 Consider the satisfiable QBF $\psi := \exists x_1, x_2 \forall y_3 \exists x_4.(\bigwedge_{i:=1}^{5} C_i)$ where the clauses C_i are defined as follows:

$$C_1 = (y_3 \vee \bar{x}_4)\ \ C_3 = (x_1 \vee y_3 \vee \bar{x}_4)\ \ C_5 = (\bar{x}_1 \vee \bar{y}_3 \vee \bar{x}_4)$$
$$C_2 = (\bar{y}_3 \vee x_4)\ \ C_4 = (x_2 \vee \bar{y}_3 \vee x_4)$$

QBCE successively removes all clauses in ψ and maintains a sequence BC of blocked clauses in the order they were identified as blocked. Initially, $BC := \emptyset$. Consider C_4 and literal $x_4 \in C_4$, which is not a blocking literal due to $\bar{x}_4 \in C_5$. However, literal $x_2 \in C_4$ is a blocking literal because there is no clause containing $\bar{x}_2$ and hence $BC := \langle C_4 \rangle$. Now $\bar{x}_4 \in C_1$ is a blocking literal since $\bar{y}_3 \in C_2$ and C_4 with $x_4 \in C_4$ is already blocked. For the same reasons $\bar{x}_4 \in C_3$ is a blocking literal. Hence BC is updated to $BC := \langle C_4, C_1, C_3 \rangle$. Then $\bar{x}_1 \in C_5$ is a blocking literal since C_3 with $x_1 \in C_3$ has already been identified as blocked, and thus $BC := \langle C_4, C_1, C_3, C_5 \rangle$. Finally, $x_4 \in C_2$ is a blocking literal since every clause containing $\bar{x}_4$ is blocked and $BC := \langle C_4, C_1, C_3, C_5, C_2 \rangle$.

A partial certificate in terms of Skolem functions of the existential variables in the left-most quantifier set of a QBF ψ can be extracted from a partial certificate of a QBF ψ' obtained from ψ by QBCE. To this end, we apply the following known approach [20] related to blocked clause elimination for propositional logic [25]. Let ψ be a QBF, ψ' be the QBF obtained from ψ by QBCE and $BC = \langle C_1, \ldots, C_n \rangle$ be the sequence of blocked clauses in the order they were identified as blocked when applying QBCE to ψ. Assume that A' is a partial certificate of $\psi' = \exists B_1 \hat{Q}'.\phi'$ where all variables in B_1 are assigned. To reconstruct a partial certificate A of ψ from A', we initially set $A := A'$ and consider each clause C_i in BC for $i = n, \ldots, 1$ in reverse order. If the variable v of the blocking literal of C_i appears in B_1 and if C_i is not satisfied by the current assignment of v in A, then we flip the assignment of v in A. The following example illustrates the reconstruction of a partial certificate.

Example 6 (Continues Example 5) Consider the QBF ψ from Example 5, where all clauses are blocked, and let $BC := \langle C_4, C_1, C_3, C_5, C_2 \rangle$ be the sequence of clauses in ψ in the order they were found blocked. Applying QBCE to ψ results in the empty QBF $\psi' := \top$ containing no clauses. Any assignment to the existential variables x_1, x_2 is a partial certificate of ψ'. However, the assignment $A' = \{x_1, x_2\}$ where both x_1 and x_2 are assigned true is not a partial certificate of the original QBF ψ because replacing x_1 and x_2 by $\top$ in ψ results in the unsatisfiable QBF $\forall y_3, \exists x_4.(y_3 \vee \bar{x}_4) \wedge (\bar{y}_3 \vee x_4) \wedge (\bar{y}_3 \vee \bar{x}_4)$. We reconstruct a partial certificate A from A' as follows. Let $A := A'$ and consider $C_i \in BC$ in reverse ordering. No assignment is flipped for C_2. The assignment of x_1 is flipped from true to false since $\bar{x}_1 \in C_5$

is a blocking literal, it appears in the leftmost existential quantifier set and the previous assignment of true to x_1 does not satisfy C_5. No assignments are flipped when inspecting the remaining clauses in BC. Finally, we obtain the partial certificate $A = \{\bar{x}_1, x_2\}$ of the original QBF ψ.

In our incremental QBF-based workflow to solve instances of conformant planning, we simplify the QBFs ψ that encode the existence of a plan of length k by QBCE to obtain a QBF ψ'. We apply the reconstruction procedure illustrated in Example 6 to extract a plan for the planning instance from a partial certificate of ψ' returned by the QBF solver. We remark that the QRAT proof system [18, 19] allows to generate Skolem functions of satisfiable QBFs which have been preprocessed by common techniques other than QBCE like elimination or expansion of variables, for example. Since we consider only QBCE for preprocessing and partial certificates, the generation of Skolem functions based on QRAT is possible but not necessary in our workflow.

In addition to QBCE as a preprocessing technique, we also apply QBCE for *inprocessing* [26] within DepQBF in our workflow. Inprocessing has been introduced in the context of CDCL-based SAT solving and combines preprocessing with formula simplifications based on unit clauses that have been learned. Inprocessing has not been presented for QCDCL-based QBF solvers so far. After a unit clause (l) has been learned in QCDCL when solving a QBF ψ, all clauses containing the literal l are removed from ψ, and all literals $\bar{l}$ are removed from clauses in ψ. After simplifications by learned unit clauses, further simplifications by QBCE may become possible. Whereas preprocessing is a static approach which is applied once before a QBF is solved, inprocessing is more dynamic since it takes unit clauses learned during a run of QCDCL into account.

Example 7 (Related to Examples 1 and 3) Consider the QBF ψ with prefix $\exists z, z' \forall u \exists y$ and matrix $(u \vee \bar{y}) \wedge (\bar{u} \vee y) \wedge (z \vee u \vee \bar{y}) \wedge (z' \vee \bar{u} \vee y) \wedge (\bar{z} \vee \bar{u} \vee \bar{y}) \wedge (\bar{z}' \vee u \vee y)$ from Example 3. No clause is blocked in ψ and hence applying QBCE as a preprocessing technique to ψ has no effect.

We solve ψ by QCDCL including inprocessing. Suppose that all the variables in ψ are assigned to true following the prefix ordering, resulting in the assignment $A := \{z, z', u, y\}$. The clause $(\bar{z} \vee \bar{u} \vee \bar{y})$ is falsified under assignment A, which constitutes an unsatisfiable subcase. The learned unit clause $(\bar{z})$ is produced in the same way as in Example 1. The learned unit clause $(\bar{z})$ can only be satisfied by assigning variable z to false. Hence during backtracking, all assignments in A are retracted and the search process starts over with the new assignment $A := \{\bar{z}\}$.

At this point, inprocessing simplifies ψ under A by removing the clause $(\bar{z} \vee \bar{u} \vee \bar{y})$, which is satisfied under A, by removing the literal z from the clause $(z \vee u \vee \bar{y})$, and by removing the variable z from the prefix, resulting in the simplified QBF $\psi' := \exists z' \forall u \exists y.(u \vee \bar{y}) \wedge (\bar{u} \vee y) \wedge (z' \vee \bar{u} \vee y) \wedge (\bar{z}' \vee u \vee y)$. When applying QBCE as part of inprocessing to ψ', literal y in $(z' \vee \bar{u} \vee y)$ is a blocking literal. After the blocked clause $(z' \vee \bar{u} \vee y)$ has been removed, also $\bar{z}'$ in $(\bar{z}' \vee u \vee y)$ is a blocking literal. Finally the two remaining clauses $(u \vee \bar{y})$ and $(\bar{u} \vee y)$ are also blocked with blocking literals $\bar{y}$ and y, respectively. Thus simplifications in inprocessing based on the learned unit clause $(\bar{z})$ and QBCE reduce the original QBF ψ to the empty QBF, showing that ψ is satisfiable.

In this example, clauses became blocked during inprocessing only because the original QBF ψ has been simplified by taking into account the learned unit clause $(\bar{z})$ and the corresponding necessary assignment to variable z. Hence in general, inprocessing is more powerful than preprocessing.

In DepQBF we also implemented a novel, *fully* dynamic application of QBCE [32].[6] Thereby, QBCE is used to simplify the given QBF $\psi[A]$ interpreted under the assignment A that has currently been generated in QCDCL. If QBCE simplifies $\psi[A]$ to the empty QBF, then a new cube can be learned with respect to A without the need to make further assignments. This dynamic application of QBCE is in contrast to its usual application as a preprocessing and inprocessing technique. As illustrated by Example 7, with inprocessing QBCE is applied only after ψ has been simplified according to a new learned unit clause C. With the novel dynamic application, however, QBCE may be applied to $\psi[A]$ each time after variables have been assigned to obtain a new assignment A. We refer to related literature [32] on dynamic QBCE for further details and examples.

In our experiments with the QBF-based conformant planning workflow, we evaluate all configurations of DepQBF implementing QBCE as preprocessing, inprocessing, and the novel dynamic variant of QBCE. Further, we compare these configurations combined with incremental and nonincremental solving.

2.3 Incremental QBF solving and incremental QBCE

In the QBF-based conformant planning workflow, a sequence of QBFs must be solved. Each QBF in the sequence encodes the existence of a plan of a particular length. Let $S := \langle \psi_0, \psi_1, \ldots, \psi_n \rangle$ be a sequence of QBFs. Each QBF ψ_i in S is obtained from the previous QBF ψ_{i-1} in S by removing and adding clauses and variables. In *incremental QBF solving based on QCDCL*, clauses and cubes that were learned when solving the previous QBF ψ_{i-1} may be reused in order to speed up the solving process of the current QBF ψ_i and any forthcoming QBF in S. The set of learned clauses and cubes which can be reused depends on the actual modifications of the previous QBF ψ_{i-1} to obtain the current ψ_i. An approach to incremental QBF solving was first presented in the context of bounded model checking [38]. For our experiments on conformant planning based on incremental QBF solving, we rely on our general-purpose incremental QBF solver DepQBF [34, 35].

A novel feature of DepQBF is incremental preprocessing by QBCE. In our approach, QBCE is carried out in incremental fashion when solving the current QBF ψ_i. Thereby, clauses which are blocked in the previously solved QBF ψ_{i-1} may still be blocked in the current QBF ψ_i. Such blocked clauses are ignored when applying QBCE to ψ_i, thus avoiding redundant checks when searching for blocking literals. Clauses which are blocked in ψ_{i-1} but not in ψ_i must be restored in a backtracking phase. If a clause is blocked in ψ_i then it is also blocked if other clauses are only deleted from ψ_i. Hence only adding clauses to ψ_i may turn clauses which have been formerly blocked in ψ_i into nonblocked ones. In general, incremental QBCE may result in smaller computational overhead compared to its nonincremental application, where always all clauses in a QBF are considered. We illustrate our approach to incremental QBCE by the following example.

Example 8 Consider the satisfiable QBF ψ from Example 5 with prefix $\hat{Q} := \exists x_1, x_2 \forall y_3 \exists x_4$ and clauses C_i:

$$C_1 = (y_3 \vee \bar{x}_4) \quad C_3 = (x_1 \vee y_3 \vee \bar{x}_4) \quad C_5 = (\bar{x}_1 \vee \bar{y}_3 \vee \bar{x}_4)$$
$$C_2 = (\bar{y}_3 \vee x_4) \quad C_4 = (x_2 \vee \bar{y}_3 \vee x_4)$$

[6] Dynamic QBCE is part of DepQBF version 5.0 or later.

We construct the sequence $S = \langle \psi_0, \psi_1, \psi_2 \rangle$ by successively adding clauses C_i and apply QBCE incrementally to ψ_i as follows. Let $\psi_0 := \hat{Q}.C_2 \wedge C_4 = \hat{Q}.(\bar{y}_3 \vee x_4) \wedge (x_2 \vee \bar{y}_3 \vee x_4)$, where $x_4 \in C_2$ and $x_2 \in C_4$ are blocking literals ($x_2 \in C_4$ is identified as blocking literal before $x_4 \in C_2$). By adding C_1 and C_3 to ψ_0, we obtain $\psi_1 := \hat{Q}.C_2 \wedge C_4 \wedge C_1 \wedge C_3 = \hat{Q}.C_2 \wedge C_4 \wedge (y_3 \vee \bar{x}_4) \wedge (x_1 \vee y_3 \vee \bar{x}_4)$, where $\bar{x}_4 \in C_1$ and $\bar{x}_4 \in C_3$ are blocking literals. The blocked clause C_2 with blocking literal $x_4 \in C_2$ must be checked again when QBCE is applied to ψ_1 since the literal $\bar{x}_4$ occurs in both added clauses C_1 and C_3. However, the blocked clause C_4 does not have to be checked again because the negation of its blocking literal x_2 neither occurs in C_1 nor in C_3. All clauses are blocked in ψ_1. Finally, by adding C_5 to ψ_1 we obtain $\psi_2 := \hat{Q}.C_2 \wedge C_4 \wedge C_1 \wedge C_3 \wedge C_5 = \hat{Q}.C_2 \wedge C_4 \wedge C_1 \wedge C_3 \wedge (\bar{x}_1 \vee \bar{y}_3 \vee \bar{x}_4)$, where $\bar{x}_1 \in C_5$ is a blocking literal. Only C_2 with blocking literal x_4 has to be checked again during QBCE since $\bar{x}_4 \in C_5$. Like before, C_4 does not have to be checked again.

From the perspective of a user, our implementation of incremental solving and incremental QBCE in DepQBF is a black box. A related approach to incremental preprocessing has been presented in the context of SAT solving [40]. With DepQBF, no user interaction is necessary to control incremental applications of QBCE. This is in contrast to incremental preprocessing based on *don't touch variables* [39, 46]. These variables must not be affected by preprocessing, e.g., the solver must not eliminate them. Either the user is responsible to declare variables as don't touch variables via the solver API or the solver determines them by itself according to certain criteria. With incremental preprocessing based on don't touch variables, the effects of preprocessing never have to be undone in a backtracking phase. However, don't touch variables restrict the potential benefits when preprocessing the current QBF ψ_i since certain parts of ψ_i are locked. In our approach, however, incremental QBCE is applied to the entire QBF ψ_i, with the need to backtrack some of its effects when tackling the next QBF ψ_{i+1} .

In the following, we present a case study of QBF-based conformant planning. Our goal is to evaluate incremental and nonincremental variants of QBF solving in a common application framework to allow for a fair comparison. To this end we first discuss conformant planning and two types of benchmarks we use in the experimental analysis.

3 Conformant planning and benchmark domains

A conformant planning problem consists of a set of state variables over a specified domain, a set of actions (each action with a precondition and an effect), an initial state where some values of the variables may be unknown, and a specification of the goal. The task is to find a sequence of actions, i.e., a plan, that leads from the initial state to a state where the goal specification is satisfied. The plan has to reach the goal for all possible values of unknown variables, i.e., it has to be fail-safe. This problem can nicely be encoded into QBFs, e.g., by building upon the encodings by Rintanen [43]. Conformant planning naturally arises, e.g., in repair and therapy planning [47], where a plan needs to succeed even if some obstacles arise.

The length of a plan is the number of actions in the plan. As one is usually looking for short plans, the following strategy is used. Starting at a lower bound k on the minimal plan length, we iteratively increment the plan length k until a plan is found or a limit on the plan length is reached. This strategy is readily supported by an incremental QBF solver because a large number of clauses remains untouched when moving from length k to $k + 1$. Furthermore, this strategy always leads to *optimal* plans with respect to the plan length.

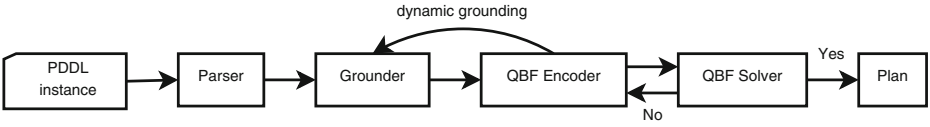

Fig. 1 Architecture of the planning tool

The two benchmark types we consider in our case study are called "Dungeon". These benchmarks are inspired by adventure computer-games and were first presented at the QBF workshop 2013 [28]. In this setting a player wants to defeat monsters living in a dungeon. Each monster requires a certain configuration of items to be defeated. In the beginning, the player picks at most one item from each pool of items. In addition, the player can exchange several items for one more powerful item if she holds all necessary "ingredients". Eventually, the player enters the dungeon. When entering the dungeon, the player is forced to pick additional items. The dilemma is that the player does not know which items she will get, i.e., the additional items are represented by variables with *unknown* values in the initial state. It might also happen that the new items turn out to be obstructive given the previously chosen item configuration. The goal is to pick items such that irrespective of the additional items the player defeats at least one monster.

We consider two variants of the Dungeon benchmark. In variant *v0* the player is only allowed to enter the dungeon once, thus has to pick the items and build more powerful items in advance. In contrast, in variant *v1* the player might attempt fighting the monsters several times and pick/build further items in between if the player was unsuccessful.

Despite the simple concept, these benchmarks are well suited for our case study. First, they capture the full hardness of $\Sigma_2\mathsf{P}$ -complete problems. Second, it is natural to reinterpret the game setting as a configuration or maintenance problem.

4 QBF planning tool

We briefly describe our planning tool that takes planning instances as input and encodes them as a sequence of QBFs. This tool generates a plan of minimal length for a given conformant planning instance with uncertainty in the initial state (if it exists).

Figure 1 illustrates the architecture of our planning tool which was used for the experiments. The tool takes a planning instance given in PDDL (Planning Domain Definition Language) format as input. PDDL is a format to represent planning problems and has been used in the international planning competitions.[7] After parsing the input, the grounder analyzes the given planning instance and calculates a lower bound ℓ on the plan length. Starting with a plan length of $k = \ell$, the grounder then grounds only relevant parts of the instance, i.e., the grounder systematically eliminates variables from the PDDL instance. In a next step, the QBF encoder takes the ground representation as input and transforms it into a QBF that is satisfiable if and only if the planning problem has a plan of length k. The encoding which is used for this transformation to QBFs builds upon the $\exists\forall\exists$-encoding described in the work of Rintanen [43]. We decided to employ the $\exists\forall\exists$-encoding rather than a $\exists\forall$-encoding as this gives a more natural encoding and simplifies the PCNF transformation. Since in this work we focus on a comparison of the incremental and nonincremental approach, we do not go

[7] http://www.icaps-conference.org/index.php/Main/Competitions.

into the details of the encoding. After the transformation into a QBF, the QBF encoder then invokes a QBF solver on the generated QBF. If the generated QBF is satisfiable, our system extracts the optimal plan from the assignment of the leftmost ∃-block. If the QBF is unsatisfiable, the plan length k is incremented, additional relevant parts of the problem may need grounding, and the subsequent QBF is passed to the solver. Below, we give an overview of the features and optimizations of our planning tool. Notice that in this simplified picture, preprocessing a QBF is considered to be part of QBF solving.

Since grounding the planning instance can cause an exponential blow-up in the size of the input, we have implemented a dynamic grounding algorithm. This algorithm uses ideas from the concept of the planning graph [7] to only ground actions that are relevant for a certain plan length. With this optimization, we are able to make the grounding process feasible. Although the planning tool provides several methods to compute lower bounds on the plan length, in our experiments we always started with plan length $k := 0$ to allow for a better comparison of the incremental and nonincremental approach.

Our *incremental* QBF solver DepQBF is written in C whereas the planning tool is written in Java. To integrate DepQBF in our tool and to employ its features for incremental solving, we implemented a Java interface for DepQBF, called *DepQBF4J*.[8] This way, DepQBF can be integrated into arbitrary Java applications and its API functions can then be called via the Java Native Interface (JNI).

In our planning tool, the use of DepQBF's API is crucial for incremental solving, because we have to avoid writing the generated QBFs to a file. Instead, we add and modify the QBFs to be solved directly via the API of DepQBF. The API provides *push* and *pop* functions to add and remove *frames*, i.e., sets of clauses, in a stack-based manner. The CNF part of a QBF is represented as a stack of frames.

Given a planning instance, the workflow starts with plan length $k = 0$. The QBF ψ_k for plan length k can be encoded naturally in an incremental fashion by maintaining two frames f_0 and f_1 of clauses: clauses which encode the goal state are added to f_1. All other clauses are added to f_0. If ψ_k is unsatisfiable, then f_1 is deleted by a *pop* operation, i.e., the clauses encoding the goal state of plan length k are removed. The plan length is increased by one and additional clauses encoding the possible state transitions from plan length k to $k + 1$ are added to f_0. The clauses encoding the goal state for plan length $k + 1$ are added to a new f_1. Note that in the workflow clauses are added to f_0, but this frame is never deleted.

The workflow terminates if (1) the QBF ψ_k is satisfiable, indicating that the instance has a plan with *optimal* length k, or (2) ψ_k is unsatisfiable and $k + 1$ exceeds a user-defined upper bound, indicating that the instance does not have a plan of length k or smaller, or (3) the time or memory limits are exceeded. In cases (1) and (2), we consider the planning instance as solved. For the experimental evaluation, we impose an upper bound of 200 on the plan length.

The Dungeon benchmark captures the full hardness of problems on the second level of the polynomial hierarchy. Therefore, as shown in the following section, already instances with moderate plan length might be hard for QBF solvers as well as for planning-specific solvers [28]. We considered an upper bound of 200 of the plan length to be sufficient to show the difference between the incremental and the nonincremental QBF-based approach. The hardness is due to the highly combinatorial nature of the Dungeon instances, which also applies to configuration and maintenance problems. Further, configuration and maintenance

[8] DepQBF4J is part of the release of DepQBF version 3.03 or later.

problems can be encoded easily into conformant planning as the Dungeon benchmark is essentially a configuration problem.

Our planning tool can also be combined with any *nonincremental* QBF solver (supporting QDIMACS as input format) to determine a plan of minimal length in a *nonincremental* fashion. This is done by writing the QBFs which correspond to the plan lengths $k = 0, 1, \ldots$ under consideration to separate files and solving them with a standalone QBF solver.

5 Experimental evaluation

We evaluate the incremental workflow described in the previous section using planning instances from the Dungeon benchmark. The purpose of our experimental analysis is to compare incremental and nonincremental QBF solving including incremental and nonincremental variants of pre- and inprocessing by QBCE in the context of conformant planning. Thereby, we provide the first empirical study of *incremental* QBF solving in the planning domain. In addition to QBF-based bounded model checking [37, 38], our results independently motivate the use of incremental QBF solving in other application domains.

Apart from DepQBF, which is based on QCDCL, in our study we also consider the *nonincremental* QBF solvers Nenofex [33] and RAReQS [23], which are based on variable expansion. As outlined in Section 2.1, variable expansion is a state-of-the-art approach to QBF solving next to QCDCL. For a comprehensive comparison to incremental QCDCL-based QBF solving as implemented in DepQBF, it would be interesting to also consider an incremental application of variable expansion. However, we are not aware of incremental expansion-based QBF solvers which we could have included in our experimental study. In general, it is unclear how to implement incremental expansion efficiently in a solver.

From the Dungeon benchmark described in Section 3, we selected 144 planning instances from each variant *v0* and *v1*, resulting in 288 planning instances. Given a planning instance, we allowed 900 seconds wall clock time and 7 GB of memory for the entire workflow, which includes grounding, QBF encoding and QBF solving. All experiments reported were run on AMD Opteron 6238, 2.6 GHz, 64-bit Linux.

5.1 Incremental and nonincremental solving

In a first step, we compare the performance of incremental and nonincremental QBF solving in the planning workflow. To this end, we used incremental and nonincremental variants of our QBF solver DepQBF, referred to as incDepQBF and DepQBF, respectively. For nonincremental solving, we called the standalone solver DepQBF by system calls from our planning tool. Thereby, we generated the QBF encoding of a particular planning instance and wrote it to a file on the hard disk. DepQBF then reads the QBF from the file. For incremental solving, we called incDepQBF through its API via the DepQBF4J interface. This way, the QBF encoding is directly added to incDepQBF by its API within the planning tool (as outlined in the previous section), and no files are written. The solvers incDepQBF and DepQBF have the *same* codebase. Therefore, differences in their performance are due to whether incremental solving is applied or not.

In a next step, to evaluate the impact of QBCE on the performance of the planning workflow, we combined both incDepQBF and DepQBF with preprocessing, inprocessing, and the dynamic variant of QBCE [32]. These combinations result in eight configurations of DepQBF shown in the leftmost column of Table 1. Note that applications of QBCE in incDepQBF are always incremental in the sense that redundant work in QBCE is avoided in

Table 1 Overall statistics for the planning workflows implementing incremental and nonincremental QBF solving by incDepQBF and DepQBF, respectively. Both incDepQBF and DepQBF were tested with and without QBCE for preprocessing (-pre), inprocessing (-inp) and applied dynamically in QCDCL (-dyn) [32], respectively. The columns show the total time for the planning workflow on all 288 instances (including time outs), total solved planning instances, solved instances where a plan was found and where no plan with length 200 or shorter exists, total solved QBFs in the workflow on all solved and unsolved planning instances (TS-QBFs) and on those planning instances which were not solved (US-QBFs) when using any variant of DepQBF shown in the table. Lines are sorted by numbers of total solved planning instances (column "Solved")

	288 Planning Instances (Dungeon Benchmark: *v0* and *v1*)					
	Time	Solved	Plan found	No plan	TS-QBFs	US-QBFs
Incremental QBCE:						
incDepQBF-inp:	101,060	181	165	16	5020	1148
incDepQBF-pre:	101,347	180	164	16	4948	1108
incDepQBF-dyn:	100,756	180	163	17	5111	1098
Nonincremental QBCE:						
DepQBF-inp:	103,330	179	166	13	4433	995
DepQBF-pre:	103,804	178	165	13	4427	990
DepQBF-dyn:	104,585	177	164	13	4440	1020
No QBCE:						
incDepQBF:	103,643	176	163	13	3923	679
DepQBF:	112,648	165	162	3	2146	677

a sequence of incremental solver runs as illustrated by Example 8. In contrast to that, applications of QBCE in DepQBF are nonincremental, where QBCE is always carried out from scratch on each QBF to be solved.

DepQBF comes with an optional advanced analysis of variable dependencies in terms of *dependency schemes* [44]. Dependency schemes are binary relations over the set of variables in a QBF expressing independence of variables. Independence of variables allows to relax the linear prefix ordering given by the quantifier prefix of a QBF. This way, QCDCL may benefit from increased freedom in assigning variables. It is an open problem how to efficiently combine dependency schemes with incremental solving. To allow for a fair comparison, we disabled advanced dependency schemes in all variants of (inc)DepQBF. Instead, we used the linear ordering of the quantifier prefix of the QBFs.

The statistics in Table 1 illustrate that incremental QBF solving by incDepQBF outperforms nonincremental solving by DepQBF in the planning workflow in terms of solved planning instances and solved QBFs in the planning workflow (section at the bottom of Table 1). Each of the 165 planning instances solved by DepQBF was also solved by incDepQBF. The application of QBCE increases the numbers of solved planning instances and solved QBFs in both incDepQBF and DepQBF. Note that the numbers of solved QBFs (columns TS-QBFs and US-QBFs) provides a more fine grain performance measure than the numbers of solved planning instances. The benefits of QBCE are reflected by the ranking of the solvers in Table 1 by the numbers of solved planning instances, where incremental

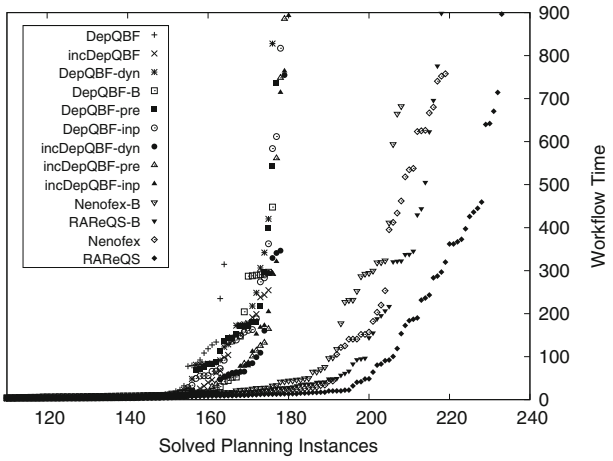

Fig. 2 Related to Tables 1 and 4. Sorted run times of the planning workflow with incremental QBF solving by incDepQBF and nonincremental solving by various solvers, including combinations of QBF solvers with QBCE (-pre, -inp, -dyn) and preprocessing by Bloqqer (-B)

solving with incremental QBCE appears in the section at the top, nonincremental solving with nonincremental QBCE appears in the middle and solving without QBCE appears at the bottom. As illustrated in the plot shown in Fig. 2, the run times of the variants of (inc)DepQBF are close to each other, except for nonincremental solving by DepQBF.

Incremental solving performs particularly well on instances for which no plan exists. Considering the 13 instances solved using incDepQBF which do not have a plan (Table 1), on average the total workflow took 83.35 seconds and incDepQBF spent 35,729 assignments and 135 backtracks per solved QBF. Tables 2 and 3 show an in-depth comparison of DepQBF, incDepQBF, and incDepQBF-inp with QBCE for inprocessing on commonly solved planning instances. Incremental solving with QBCE for inprocessing by incDepQBF-inp results in shorter times, fewer backtracks and assignments than incremental solving without QBCE (Table 3). Incremental solving without QBCE outperforms nonincremental solving except on the instances from the set Dungeon-*v1* (Table 2).

The different calling principles of incDepQBF (by the API) and DepQBF (by system calls) may have some influence on the overall run time of the workflow, depending on the underlying hardware and operating system. In general, the use of the API avoids I/O overhead in terms of hard disk accesses and thus might save run time. Due to the timeout of 900 seconds and the relatively small number of QBF solver calls in the workflow (at most 201, for plan length 0 up to the upper bound of 200), we expect that the influence of the calling principle on the overall time statistics in Table 1 and Fig. 2 is only marginal. The number of backtracks and assignments as shown in Tables 2 and 3 independently illustrate the benefits of incremental solving and QBCE.

Let P_i, $i \in \mathbb{N}$, denote the set of planning instances where a plan of length i was found using both incDepQBF and DepQBF without QBCE. Figure 3 shows how the number of backtracks evolves if the plan length is increased. On the instances in P_k, which have a plan with optimal length k, we observed peaks in the number of backtracks by incDepQBF as well as DepQBF on those QBFs which correspond to the plan length $k - 1$. Thus empirically the final unsatisfiable QBF for plan length $k - 1$ is harder to solve than the QBF for the

Table 2 Total, average, and median number of assignments (a, $\overline{a}$, and $\tilde{a}$, respectively), backtracks (b, $\overline{b}$, $\tilde{b}$), and workflow time (t, $\overline{t}$, $\tilde{t}$) for planning instances from Dungeon-*v0* (left) and Dungeon-*v1* (right) where both workflows using DepQBF and incDepQBF found the optimal plan. On the set Dungeon-*v0*, incremental solving by incDepQBF results in shorter times and fewer backtracks and assignments than nonincremental solving by DepQBF

	Dungeon-v0 (81 solved instances)		*Dungeon-v1 (81 solved instances)*	
	DepQBF	incDepQBF	DepQBF	incDepQBF
Total:				
a:	149,436,140	122,233,046	93,707,457	126,344,508
b:	1,472,156	1,237,384	777,630	1,083,059
t:	951.22	686.75	515.32	581.92
Per instance:				
$\overline{a}$:	1,844,890	1,509,049	1,156,882	1,559,808
$\overline{b}$:	18,174	15,276	9,600	13,371
$\overline{t}$:	11.74	8.47	6.36	7.18
$\tilde{a}$:	1,388	1,391	1,553	1,499
$\tilde{b}$:	13	11	15	15
$\tilde{t}$:	1.25	0.65	1.29	0.59
Per solved QBF:				
$\overline{a}$:	549,397	449,386	349,654	471,434
$\overline{b}$:	5,412	4,549	2,901	4,041
$\overline{t}$:	3.49	2.52	1.92	2.17
$\tilde{a}$:	828	833	805	806
$\tilde{b}$:	1	1	1	1
$\tilde{t}$:	1.25	0.65	1.29	0.59

optimal plan length k or for shorter plan lengths. Figure 3 (right plot) shows notable exceptions. For P_6, the number of backtracks by DepQBF increases in contrast to incDepQBF. For P_5 and P_7, incDepQBF spent more backtracks than DepQBF. We attribute this difference to the heuristics in (inc)DepQBF. The same QBFs has to be solved by incDepQBF and DepQBF in one run of the workflow. However, the heuristics in incDepQBF might be negatively influenced by previously solved QBFs. We made similar observations on instances not solved with either incDepQBF or DepQBF where DepQBF reached a longer plan length than incDepQBF within the given time limit.

5.2 Impact of preprocessing

The results of QBF solver evaluations carried out in the context of QBFEVAL[9] and the (previously mentioned) QBF Gallery have shown empirically that preprocessing is vital when solving QBFs from many application domains. To analyze the impact of preprocessing on the planning workflow, we combined the preprocessor Bloqqer [6] with the

[9] http://www.qbflib.org/index_eval.php.

Table 3 Like Table 2 but comparing incDepQBF and incDepQBF-inp, the variant of DepQBF which solved the largest number of planning instances according to the results in Table 1. Incremental solving by incDepQBF-inp with incremental QBCE for inprocessing outperforms incDepQBF without QBCE in terms of time and numbers of assignments and backtracks on both sets Dungeon-*v0* (left) and Dungeon-*v1*

	Dungeon-v0 (81 solved instances)		*Dungeon-v1 (82 solved instances)*	
	incDepQBF	incDepQBF-inp	incDepQBF	incDepQBF-inp
Total:				
a:	122,233,046	82,109,291	164,131,257	100,010,912
b:	1,237,384	889,450	1,459,655	1,023,084
t:	686.75	558.09	818.63	661.20
Per instance:				
$\overline{a}$:	1,509,049	1,013,694	2,001,600	1,219,645
$\overline{b}$:	15,276	10,980	17,800	12,476
$\overline{t}$:	8.47	6.89	9.98	8.06
$\tilde{a}$:	1,391	1,332	1,641	1,356
$\tilde{b}$:	11	4	15	5
$\tilde{t}$:	0.65	0.64	0.61	0.60
Per solved QBF:				
$\overline{a}$:	449,386	301,872	596,840	363,676
$\overline{b}$:	4,549	3,270	5,307	3,720
$\overline{t}$:	2.52	2.05	2.97	2.40
$\tilde{a}$:	833	755	828	748
$\tilde{b}$:	1	0	1	0
$\tilde{t}$:	0.65	0.64	0.61	0.60

nonincremental variant of DepQBF. Bloqqer implements QBCE and more advanced techniques like variable expansion. In addition to DepQBF, we also integrated Nenofex [33] and RAReQS [23] in the planning workflow, which are based on variable expansion as outlined in Section 2.1. We selected these two solvers because in the experimental results of the QBF Gallery events expansion-based solvers performed well on QBFs generated from the Dungeon benchmark[10] and from other planning[11] benchmarks.

Our implementation of QBCE amounts to partial preprocessing in DepQBF and incDepQBF, which is in contrast to full preprocessing as provided by Bloqqer. In order to extract plans for planning instances solved by (inc)DepQBF with QBCE, it is necessary to reconstruct partial certificates as illustrated in Example 6. Full preprocessing, however, requires to either declare certain variables as don't touch variables [39, 46], which may limit the effects of preprocessing, or to combine preprocessing with the extraction of full certificates [19, 22]. To focus on the impact of preprocessing on the planning workflow, in the experiments reported in the following, we ignore the extraction of plans from solved planning instances.

[10] http://www.kr.tuwien.ac.at/events/qbfgallery2013/sc_apps/conf_planning_dungeon.html.

[11] http://www.kr.tuwien.ac.at/events/qbfgallery2013/sc_apps/planning_CTE.html.

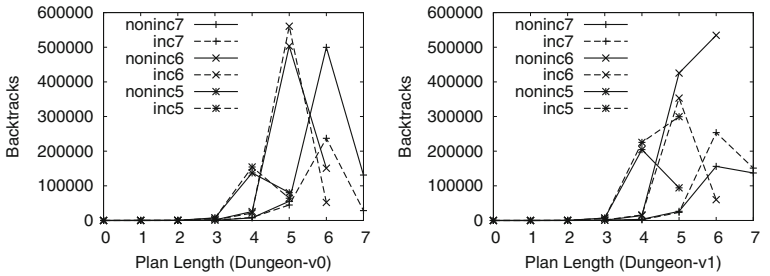

Fig. 3 Related to Table 2. The data points on the lines "inc5" (*dashed*) and "noninc5" (*solid*) show the total numbers of backtracks spent by incDepQBF and DepQBF on the QBFs corresponding to the plan lengths $i = 0, \ldots, 5$ for all instances in P_5. The data points for P_6 and P_7 were computed similarly for the plan lengths $i = 0, \ldots, 6$ and $i = 0, \ldots, 7$, respectively, and are shown on the lines "inc6", "noninc6" and "inc7", "noninc7"

Table 4 shows the performance of the planning workflow when using nonincremental solving with and without preprocessing by Bloqqer. The same limits for wall clock time and memory as for Table 1 were used (900 seconds, 7 GB). Note that while columns TS-QBFs, i.e., the total number of solved QBFs in solved and unsolved planning instances, in Tables 1 and 4 are comparable, columns US-QBFs are not since these statistics are computed based on those planning instances which were not solved by any solver shown in the respective table. Figure 2 shows the sorted run times of the workflow related to Table 4. To integrate Bloqqer into the workflow, we preprocess the current QBF and write the preprocessed formula to a file on the hard disk. The nonincremental solvers then read the file and solve the preprocessed formula.

Interestingly, both Nenofex and RAReQS perform considerably worse in terms of solved planning instances and solved QBFs (columns TS-QBFs and US-QBFs) if Bloqqer is applied prior to solving (section at the bottom of Table 4). The application of Bloqqer is beneficial for DepQBF only (line DepQBF-B), which solves more planning instances (177) and more QBFs than DepQBF without Bloqqer (166). The number of instances solved by

Table 4 Related to Table 1 (same column labels) and Fig. 2. Performance of the planning workflow with nonincremental QBF solving by DepQBF, Nenofex and RAReQS. The latter two are based on variable expansion. Lines are sorted by numbers of total solved planning instances. All solvers were tested with (-B) and without preprocessing by Bloqqer

	288 Planning Instances (Dungeon Benchmark: *v0* and *v1*)					
	Time	Solved	Plan found	No plan	TS-QBFs	US-QBFs
No Bloqqer:						
RAReQS:	60,566	234	213	21	9124	3484
Nenofex:	73,268	220	205	15	9002	3620
DepQBF:	113,020	166	163	3	2146	373
Bloqqer:						
RAReQS-B:	71,864	219	213	6	7295	2554
Nenofex-B:	78,848	209	203	6	7241	2669
DepQBF-B:	103,465	177	171	6	4967	1173

the nonincremental variant of DepQBF as shown in Tables 1 and 4 differs by one. That is due to different calling principles of the solvers. For the experiments shown in Table 1, the formulas to be solved are directly added to a solver via its API. In contrast to that, for Table 4 the formulas are first written to hard disk and then read by the solvers. Although Bloqqer is beneficial for DepQBF (DepQBF-B), the incremental variants of incDepQBF including QBCE (section on top of Table 1) solve more planning instances and more QBFs (columns TS-QBFs) than DepQBF with Bloqqer (except for incDepQBF-pre).

It would be interesting to apply the preprocessing techniques implemented in Bloqqer in an *incremental* way similar to our implementation of incremental QBCE (cf. Example 8). For example, incremental applications of variable expansion may considerably improve the performance of the planning workflow and combine the strengths of expansion-based solvers such as Nenofex and RAReQS and incremental solvers such as incDepQBF.

5.3 Comparison to heuristic approaches

Although our focus is on a comparison of nonincremental and incremental QBF solving, we report on additional experiments with the heuristic planning tools ConformantFF [21] and T0 [42]. In contrast to our implemented QBF-based approach to conformant planning, heuristic tools do not guarantee to find a plan with the optimal (i.e., shortest) length. In practical settings, plans with optimal length are often desirable. Moreover, the QBF-based approach allows to verify the nonexistence of a plan with respect to a given upper bound on the plan length. Given these differences between our QBF-based approach and heuristic approaches, a comparison by run times and numbers of solved instances only is not appropriate.

Related to Table 1, ConformantFF solved 169 planning instances, where it found a plan for 144 instances and for 25 instances concluded that no plan exists (with a length shorter than our considered upper bound of 200). Considering the 124 instances where both incDepQBF and ConformantFF found a plan, for 42 instances the optimal plan found by incDepQBF was strictly shorter than the plan found by ConformantFF. On these 124 instances, the average (median) length of the plan found by incDepQBF was 2.06 (1), compared to an average (median) length of 3.45 (1) by ConformantFF.

Due to technical problems, we were not able to run the experiments with T0[12] on the same system as the experiments with (inc)DepQBF and ConformantFF. Hence the results by T0 reported in the following are actually incomparable to Table 1. However, we include them here to allow for a basic comparison of the plan lengths.

Using the same time and memory limits as for incDepQBF and ConformantFF, T0 solved 206 planning instances, where it found a plan for 203 instances and concluded that no plan exists (with a length shorter than the upper bound of 200) for three instances. Given the 156 instances where both incDepQBF and T0 found a plan, for 56 instances the optimal plan found by incDepQBF was strictly shorter than the plan found by T0. On the 156 instances, the average (median) length of the plan found by incDepQBF was 2.25 (1), compared to an average (median) length of 3.08 (2) by T0.

From the 13 instances solved by incDepQBF for which no plan exists (Table 1), none was solved using T0 and 12 were solved using ConformantFF.

Our experiments confirm that the QBF-based approach to conformant planning finds optimal plans in contrast to the plans found by the heuristic approaches implemented in

[12] Experiments with T0 were run on AMD Opteron 6176 SE, 2.3 GHz, 64-bit Linux.

ConformantFF and T0. Moreover, as (inc)DepQBF and other search-based QBF solvers rely on Q-resolution [27] as a proof system underlying QCDCL, these solvers allow to independently explain and verify the nonexistence of a plan (of a particular length) for an instance P of conformant planning. To this end, the unsatisfiability of the QBF encodings arising from the planning workflow when applied to P can be verified based on Q-resolution proofs and Herbrand function countermodels [41, 49]. This is an appealing property of the QBF-based approach. In practical applications, it may be interesting to have an explanation of the nonexistence of a plan in addition to the mere answer that no plan exists.

The exact QBF-based approach to conformant planning can be combined with heuristic approaches in a portfolio-style system, for example, to benefit from their individual strengths. Thereby, the two approaches are applied in parallel and independently from each other. This way, modern multi-core hardware can naturally be exploited.

6 Conclusion

We presented a case study of incremental QBF solving based on a workflow to incrementally encode planning problems with uncertainty in the initial state into sequences of QBFs. Thereby, we focused on the general-purpose QBF solver DepQBF. The incremental approach avoids some redundancy when encoding and solving the QBFs. First, parts of the QBF encodings for shorter plan lengths can be reused in the encodings for longer plan lengths. Second, the incremental QBF solver benefits from information that was learned from previously solved QBFs. Compared to heuristic approaches, the QBF-based approach has the advantage that it always finds the shortest plan and that it allows to verify the nonexistence of a plan of a certain length by Q-resolution proofs.

To be able to compare our approach with other QBF solvers we have integrated the expansion-based QBF solvers Nenofex and RAReQS as well as the QBF preprocessor Bloqqer into our planning tool. Further, we have implemented pre- and inprocessing techniques based on QBCE in DepQBF. In our experiments, we also considered a novel dynamic application of QBCE in QCDCL-based QBF solvers [32].

We provided the first empirical study of incremental QBF solving in the context of planning. Furthermore, for the first time we reported on experiments with incremental QBCE as an inprocessing technique. Our experimental results show that in the planning workflow incremental QBF solving with incremental QBCE as implemented in our solver DepQBF outperforms incremental QBF solving without QBCE, which in turn outperforms nonincremental QBF solving using DepQBF. These performance results manifest in terms of solved planning instances and statistics like the number of backtracks, assignments, and run time of the workflow.

The expansion-based solvers Nenofex and RAReQS result in the best overall performance of the workflow. These solvers are nonincremental. From a proof complexity point of view, variable expansion is different from Q-resolution, which is applied in QCDCL. On certain classes of QBFs, variable expansion allows for proofs which are exponentially more compact than any Q-resolution proof, and vice versa [4, 24]. Hence solvers based on variable expansion and Q-resolution have individual strengths depending on the QBFs to be solved. Despite the good performance of expansion-based solvers on the planning benchmarks, in general it is necessary to further improve QCDCL-based solving as an approach which is orthogonal to variable expansion with respect to proof complexity. For example, as shown by our results using the planning instances, full preprocessing using Bloqqer is beneficial for nonincremental solving by DepQBF but harmful for Nenofex and RAReQS.

To improve QCDCL-based solving, an incremental variant of variable expansion could be combined with incremental solving as a pre- or inprocessing technique.

The results of our empirical study motivate the use of incremental QBF solving in applications other than planning. In our planning tool, we successively create formulas which encode the existence of a plan. This is similar to QBF encodings of reachability problems like BMC [39] where the existence of a path from an initial state to a goal state is encoded. Therefore, the basic workflow as implemented in our planning tool can also be applied to solve reachability problems like BMC.

We implemented the Java interface DepQBF4J to integrate the solver DepQBF in our planning tool. This interface is extensible and can be combined with arbitrary Java applications. From the perspective of the user, our implementation of incremental solving by DepQBF including QBCE is a black box, which facilitates the integration of DepQBF in other applications. In DepQBF, QBCE is carried out entirely inside the solver. This is in contrast to incremental preprocessing based on don't touch variables [39, 46].

Our experiments revealed that keeping learned information in incremental QBF solving might be harmful if the heuristics of the solver are negatively influenced. Our observations merit a closer look on these heuristics when used in incremental solving. In general, the integration of additional QBF preprocessing techniques into an incremental QBF solver is highly desirable since this could further improve the performance of QBF-based workflows.

Acknowledgments Open access funding provided by TU Wien. Supported by the Austrian Science Fund (FWF) under grants S11409-N23, P25518-N23, and Y698 and the German Research Foundation (DFG) under grant ER 738/2-1.

References

1. Audemard, G., Lagniez, J.M., Simon, L.: Improving Glucose for incremental SAT solving with assumptions: Application to MUS extraction. In: Proc. SAT 2013, LNCS, vol. 7962, pp. 309–317. Springer (2013)
2. Balabanov, V., Jiang, J.H.R.: Unified QBF certification and its applications. Formal Methods Syst. Des. **41**(1), 45–65 (2012)
3. Baral, C., Kreinovich, V., Trejo, R.: Computational complexity of planning and approximate planning in the presence of incompleteness. Artif. Intell. **122**(1-2), 241–267 (2000)
4. Beyersdorff, O., Chew, L., Janota, M.: Proof complexity of resolution-based QBF calculi. In: Proc. STACS 2015, LIPIcs, vol. 30, pp. 76–89. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2015)
5. Biere, A.: Resolve and expand. In: Proc. SAT 2004, LNCS, vol. 3542, pp. 59–70. Springer (2004)
6. Biere, A., Lonsing, F., Seidl, M.: Blocked clause elimination for QBF. In: Proc. CADE 2011, LNCS, vol. 6803, pp. 101–115. Springer (2011)
7. Blum, A., Furst, M.L.: Fast planning through planning graph analysis. Artif. Intell. **90**(1-2), 281–300 (1997)
8. Bubeck, U., Kleine Büning, H.: Bounded universal expansion for preprocessing QBF. In: Proc. SAT 2007, LNCS, vol. 4501, pp. 244–257. Springer (2007)
9. Cadoli, M., Schaerf, M., Giovanardi, A., Giovanardi, M.: An algorithm to evaluate quantified Boolean formulae and its experimental evaluation. J. Autom. Reas. **28**(2), 101–142 (2002)
10. Cashmore, M., Fox, M., Giunchiglia, E.: Planning as quantified Boolean formula. In: Proc. ECAI, FAIA, vol. 242, pp. 217–222. IOS Press (2012)
11. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962)

12. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electron. Notes Theor. Comput. Sci. **89**(4), 543–560 (2003)
13. Egly, U., Kronegger, M., Lonsing, F., Pfandler, A.: Conformant planning as a case study of incremental QBF solving. In: Proc. AISC 2014, LNCS, pp. 120–131. Springer (2014)
14. Giunchiglia, E., Marin, P.: Narizzano, M.: sQueezeBF: An effective preprocessor for QBFs based on equivalence reasoning. In: Proc. SAT 2010, LNCS, vol. 6175, pp. 85–98. Springer (2010)
15. Giunchiglia, E., Narizzano, M., Tacchella, A.: Clause/term resolution and learning in the evaluation of quantified Boolean formulas. J. Artif. Intell. Res. **26**, 371–416 (2006)
16. Goultiaeva, A., Van Gelder, A., Bacchus, F.: A uniform approach for generating proofs and strategies for both true and false QBF formulas. In: Proc. IJCAI 2011, pp. 546–553. AAAI Press (2011)
17. Heule, M., Järvisalo, M., Lonsing, F., Seidl, M., Biere, A.: Clause elimination for SAT and QSAT. J. Artif. Intell. Res. **53**, 127–168 (2015)
18. Heule, M., Seidl, M., Biere, A.: Efficient extraction of skolem functions from QRAT proofs. In: Proc. FMCAD 2014, pp. 107–114. IEEE (2014)
19. Heule, M., Seidl, M., Biere, A.: A unified proof system for QBF preprocessing. In: Proc. IJCAR 2014, LNCS, vol. 8562, pp. 91–106. Springer (2014)
20. Heyman, T., Smith, D., Mahajan, Y., Leong, L.: Abu-Haimed, H.: Dominant controllability check using QBF-solver and netlist optimizer. In: Proc. SAT 2014, LNCS, vol. 8561, pp. 227–242. Springer (2014)
21. Hoffmann, J., Brafman, R.I.: Conformant planning via heuristic forward search: A new approach. Artif. Intell **170**(6–7), 507–541 (2006)
22. Janota, M., Grigore, R.: Marques-Silva, J.: On QBF proofs and preprocessing. In: Proc. LPAR 2013, LNCS, vol. 8312, pp. 473–489. Springer (2013)
23. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.M.: Solving QBF with counterexample guided refinement. In: Proc. SAT 2012, LNCS, vol. 7317, pp. 114–128. Springer (2012)
24. Janota, M., Marques-Silva, J.: Expansion-based QBF solving versus Q-resolution. Theor. Comput. Sci. **577**, 25–42 (2015)
25. Järvisalo, M., Biere, A.: Reconstructing solutions after blocked clause elimination. In: Proc. SAT 2010, LNCS, vol. 6175, pp. 340–345. Springer (2010)
26. Järvisalo, M., Heule, M., Biere, A.: Inprocessing rules. In: Proc. IJCAR 2012, LNCS, vol. 7364, pp. 355–370. Springer (2012)
27. Kleine Büning, H., Karpinski, M., Flögel, A.: Resolution for quantified Boolean formulas. Inform. Comput. **117**(1), 12–18 (1995)
28. Kronegger, M., Pfandler, A., Pichler, R.: Conformant planning as a benchmark for QBF-solvers. In: Proc. QBF 2013, pp. 1–5. http://fmv.jku.at/qbf2013/reportQBFWS13.pdf (2013)
29. Kullmann, O.: On a generalization of extended resolution. Discrete Appl. Math. **96–97**, 149–176 (1999)
30. Lagniez, J.M., Biere, A.: Factoring out assumptions to speed up MUS extraction. In: Proc. SAT 2013, LNCS, vol. 7962, pp. 276–292. Springer (2013)
31. Letz, R.: Lemma and model caching in decision procedures for quantified Boolean formulas. In: Proc. TABLEAUX 2002, LNCS, vol. 2381, pp. 160–175. Springer (2002)
32. Lonsing, F., Bacchus, F., Biere, A., Egly, U., Seidl, M.: Enhancing search-based QBF solving by dynamic blocked clause elimination. In: Proc. LPAR 2015, LNCS, vol. 9450, pp. 418–433. Springer (2015)
33. Lonsing, F., Biere, A.: Nenofex: Expanding NNF for QBF solving. In: Proc. SAT 2008, LNCS, vol. 4996, pp. 196–210. Springer (2008)
34. Lonsing, F., Egly, U.: Incremental QBF solving. In: Proc. CP 2014, LNCS, vol. 8656, pp. 514–530. Springer (2014)
35. Lonsing, F., Egly, U.: Incremental QBF solving by DepQBF. In: Proc. ICMS 2014, LNCS, vol. 8592, pp. 307–314. Springer (2014)
36. Lonsing, F., Egly, U., Van Gelder, A.: Efficient clause learning for quantified Boolean formulas via QBF pseudo unit propagation. In: Proc. SAT 2013, LNCS, vol. 7962, pp. 100–115. Springer (2013)
37. Marin, P., Miller, C., Becker, B.: Incremental QBF preprocessing for partial design verification - (poster presentation). In: Proc. SAT 2012, LNCS, vol. 7317, pp. 473–474. Springer (2012)
38. Marin, P., Miller, C., Lewis, M.D.T., Becker, B.: Verification of partial designs using incremental QBF solving. In: Proc. DATE 2012, pp. 623–628. IEEE (2012)
39. Miller, C., Marin, P., Becker, B.: Verification of partial designs using incremental QBF. AI Commun. **28**(2), 283–307 (2015)
40. Nadel, A., Ryvchin, V., Strichman, O.: Ultimately incremental SAT. In: Proc. SAT 2014, LNCS, vol. 8561, pp. 206–218. Springer (2014)
41. Niemetz, A., Preiner, M., Lonsing, F., Seidl, M., Biere, A.: Resolution-based certificate extraction for QBF - (tool presentation). In: Proc. SAT 2012, LNCS, vol. 7317, pp. 430–435. Springer (2012)
42. Palacios, H., Geffner, H.: Compiling uncertainty away in conformant planning problems with bounded width. J. Artif. Intell. Res. **35**, 623–675 (2009)

43. Rintanen, J.: Asymptotically optimal encodings of conformant planning in QBF. In: Proc. AAAI 2007, pp. 1045–1050. AAAI Press (2007)
44. Samer, M., Szeider, S.: Backdoor sets of quantified Boolean formulas. J. Autom. Reas. **42**(1), 77–97 (2009)
45. Samulowitz, H., Davies, J., Bacchus, F.: Preprocessing QBF. In: Proc. CP 2006, LNCS, vol. 4204, pp. 514–529. Springer (2006)
46. Seidl, M., Könighofer, R.: Partial witnesses from preprocessed quantified Boolean formulas. In: Proc. DATE 2014, pp. 1–6. IEEE (2014)
47. Smith, D.E., Weld, D.S.: Conformant graphplan. In: Proc. AAAI/IAAI 1998, pp. 889–896. AAAI Press / The MIT Press (1998)
48. Van Gelder, A., Wood, S.B., Lonsing, F.: Extended failed-literal preprocessing for quantified Boolean formulas. In: Proc. SAT 2012, LNCS, vol. 7317, pp. 86–99. Springer (2012)
49. Yu, Y., Malik, S.: Validating the result of a quantified Boolean formula (QBF) solver: theory and practice. In: Proc. ASP-DAC 2005, pp. 1047–1051. ACM Press (2005)
50. Zhang, L., Malik, S.: Towards a symmetric treatment of satisfaction and conflicts in quantified Boolean formula evaluation. In: Proc. CP 2002, LNCS, vol. 2470, pp. 200–215. Springer (2002)