Abstract. We consider the problem of incrementally solving a sequence of quantified Boolean formulae (QBF). Incremental solving aims at using information learned from one formula in the process of solving the next formulae in the sequence. Based on a general overview of the problem and related challenges, we present an approach to incremental QBF solving which is application-independent and hence applicable to QBF encodings of arbitrary problems. We implemented this approach in our incremental search-based QBF solver DepQBF and report on implementation details. Experimental results illustrate the potential benefits of incremental solving in QBF-based workflows.

1 Introduction

The success of SAT technology in practical applications is largely driven by incremental solving. SAT solvers based on conflict-driven clause learning (CDCL) [32] gather information about a formula in terms of learned clauses. When solving a sequence of closely related formulae, it is beneficial to keep clauses learned from one formula in the course of solving the next formulae in the sequence.

The logic of quantified Boolean formulae (QBF) extends propositional logic by universal and existential quantification of variables. QBF potentially allows for more succinct encodings of PSPACE-complete problems than SAT. Motivated by the success of incremental SAT solving, we consider the problem of incrementally solving a sequence of syntactically related QBFs in prenex conjunctive normal form (PCNF). Building on search-based QBF solving with clause and cube learning (QCDCL) [8,13,21,24,36], we present an approach to incremental QBF solving, which we implemented in our solver DepQBF.1

Different from many incremental SAT and QBF [27] solvers, DepQBF allows to add clauses to and delete clauses from the input PCNF in a stack-based way by push and pop operations. A related stack-based framework was implemented in the SAT solver PicoSAT [5]. A solver API with push and pop increases the usability from the perspective of a user. Moreover, we present an optimization based on this stack-based framework which reduces the size of the learned clauses.

1 DepQBF is free software: http://lonsing.github.io/depqbf/
Incremental QBF solving was introduced for QBF-based bounded model checking (BMC) of partial designs [26][27]. This approach, like ours, relies on selector variables and assumptions to support the deletion of clauses from the current input PCNF [11][20][23]. The quantifier prefixes of the incrementally solved PCNFs resulting from the BMC encodings are modified only at the left or right end. In contrast to that, we consider incremental solving of arbitrary sequences of PCNFs. For the soundness it is crucial to determine which of the learned clauses and cubes can be kept across different runs of an incremental QBF solver. We aim at a general presentation of incremental QBF solving and illustrate problems related to clause and cube learning. Our approach is application-independent and applicable to QBF encodings of arbitrary problems.

We report on experiments with constructed benchmarks. In addition to experiments with QBF-based conformant planning using DepQBF [12], our results illustrate the potential benefits of incremental QBF solving in application domains like synthesis [6][33], formal verification [4], testing [17][25][34], planning [9], and model enumeration [3], for example.

2 Preliminaries

We introduce terminology related to QBF and search-based QBF solving necessary to present a general view on incremental solving.

For a propositional variable \( x, l := x \) or \( l := \neg x \) is a literal, where \( v(l) = x \) denotes the variable of \( l \). A clause (cube) is a disjunction (conjunction) of literals. A constraint is a clause or a cube. The empty constraint \( \emptyset \) does not contain any literals. A clause (cube) \( C \) is tautological (contradictory) if \( x \in C \) and \( \neg x \in C \).

A propositional formula is in conjunctive (disjunctive) normal form if it consists of a conjunction (disjunction) of clauses (cubes), called CNF (DNF). For simplicity, we regard CNFs and DNFs as sets of clauses and cubes, respectively.

A quantified Boolean formula (QBF) \( \psi := Q \phi \) is in prenex CNF (PCNF) if it consists of a quantifier-free CNF \( \phi \) and a quantifier prefix \( Q \) with \( Q := Q_1 B_1 \ldots Q_n B_n \), where \( Q_i \in \{\forall, \exists\} \) are quantifiers and \( B_i \) are blocks (i.e. sets) of variables such that \( B_i \neq \emptyset \) and \( B_i \cap B_j = \emptyset \) for \( i \neq j \), and \( Q_i \neq Q_{i+1} \).

The blocks in the quantifier prefix are linearly ordered such that \( B_i < B_j \) if \( i < j \). The linear ordering is extended to variables and literals: \( x_i < x_j \) if \( x_i \in B_i \), \( x_j \in B_j \) and \( B_i < B_j \), and \( l < l' \) if \( v(l) < v(l') \) for literals \( l \) and \( l' \).

We consider only closed PCNFs, where every variable which occurs in the CNF is quantified in the prefix, and vice versa.

A variable \( x \in B_i \) is universal, written as \( q(x) = \forall \), if \( Q_i = \forall \) and existential, written as \( q(x) = \exists \), if \( Q_i = \exists \). A literal \( l \) is universal if \( q(v(l)) = \forall \) and existential if \( q(v(l)) = \exists \), written as \( q(l) := \forall \) and \( q(l) := \exists \), respectively.

An assignment is a mapping from variables to the truth values true and false. An assignment \( A \) is represented as a set of literals \( A := \{l_1, \ldots, l_k\} \) such that, for \( l_i \in A \), if \( v(l_i) \) is assigned to false (true) then \( l_i = \neg v(l_i) \) \( (l_i = v(l_i)) \).

A PCNF \( \psi \) under an assignment \( A \) is denoted by \( \psi[A] \) and is obtained from \( \psi \) as follows: for \( l_i \in A \), if \( l_i = v(l_i) \) \( (l_i = \neg v(l_i)) \) then all occurrences of \( v(l_i) \) in