Quantified Boolean Formula Solver

Solving QBF by converting to EPR

David Green
BSc(Hons) Computer Science and Mathematics
Project report for third year project under the supervision of Konstantin Korovin done in the School of Computer Science at the University of Manchester

May, 2016

Abstract

This paper details the tool QBFTOEPR that converts quantified boolean formulas (QBFs) to effectively propositional logic (EPR) through a process of skolemization. This opens new techniques for solving QBFs using automated theorem provers for first order logic such as iProver [2] over traditional techniques for solving QBFs such as the Davis-Putnam-Logemann-Loveland (DPLL) algorithm. Other techniques are discussed for improving the efficiency of conversion and reducing the complexity of the EPR result. In particular, an implementation of dependency schemes is detailed and the concept of anti-prenexing is outlined. The tool is evaluated against the traditional QBF solver DepQBF [3] to compare the efficiency of converting and solving to solving the QBF directly. It is also evaluated against another tool called qbf2epr [4] that also converts QBFs to EPR to compare it against a different implentation of the same conversion.

cite dpll algorithm

Contents

Chapter 1

Background

First we must introduce the terminology and concepts of boolean logic including propositional logic, first order logic (including EPR) and quantified propositional logic. After the terminology has been introduced the complexity classes of the satisfiability problem of each logic will be discussed. Finally the concept of automated reasoning of both first order logic and QBF will be detailed.

1.1 Boolean Logic and Satisfiability

Boolean logic and satisfiability (SAT) have found wide applications in industry and research because propositional satisfiability is the prototypical problem for NP-Completeness so any other NP-Hard problem can be embedded in propositional satisfiability.

this needs rewriting, sat stuff is next section

1.1.1 Propositional Logic

A propositional variable p can take one of two values; either *true* or $false$. A variable can be negated using the *negation* ($\neg$) symbol which reverses its value. If p was *true* then $\neg p$ is $false$ and vice versa. We will call a variable or its negation a *literal* and denote the positive literal by l and the negative literal by $\bar{l}$. Boolean formulas are constructed from propositional variables built using the logical connectives *disjunction* ($\vee$), *conjunction* ($\wedge$) and *implies* ($\rightarrow$). A typical boolean formula might look like $(x \wedge y) \rightarrow z$.

The satisfiability of a boolean formula is a decision problem that asks if an assignment of truth values to propositional variables can make the boolean formula true. In the previous example of $(x \wedge y) \rightarrow z$ we can see that it is satisfiable as the assignment $x := true; y := true; z := true$ makes it true.

We require a more standard form of boolean formula that is easier to describe as an input format. For this we will use conjunctive normal form (CNF). CNF is a conjunction of clauses and a clause is a disjunction of literals. For example, a clause might be $(p \vee \neg q)$ and a full formula in CNF might look like $(p \vee r) \wedge (\neg r \vee q) \wedge (q)$. Using CNF allows us to more easily work with boolean formulas algorithmically.

1.1.2 Quantified Boolean Formulas

QBF extends propositional logic with the *universal* ($\forall$) and *existential* ($\exists$) quantifiers. The statement $\forall x \exists y (x \vee y)$ states that for every assignment of x there is at least one assignment of y such that the formula $(x \vee y)$ is true. We can see that this is true; if $x := true$ then the formula is true but if $x := false$ then the assignment $y := false$ doesn't work but that $y := true$ does make the formula satisfiable. Therefore, for any assignment of x there exists an assignment of y such that the formula is true.

In the most general case quantifiers can appear anywhere in a QBF. Again we need a more standard form of QBF that we can deal with algorthmically. This form is called *prenex* CNF (however we may refer to it by just CNF assuming that the formula is prenexed). Any QBF is logically equivalent to a CNF formula and the process for transforming the QBF into CNF is called *prenexing*. This process uses rewriting rules to move all the quantifiers in the formula to the leftmost part of the formula resulting in a *quantifier prefix*. For example, $(\neg(\exists x A) \wedge B$ is equivalent to $\forall x(\neg A \wedge B)$. Because all QBFs are equivalent to some QBF in CNF we shall assume that any QBF we are dealing with is already in CNF.

1.1.3 First Order Logic

First order logic uses propositions that take variables or functions as arguments to form its formulas. These variables range over a specified problem domain such as the natural numbers. For example in thedomain of the natural numbers the formula $\forall n \exists m P(n, m)$ where $P(n, m) = m > n$ is true; for any natural number n there is a number m that is larger than n. This differs from our previous definition of QBF in that QBF deals with only variables in a two valued domain (i.e. boolean) and does not have propositions.

Our notions of prenexed CNF also extend to first order logic.

As with propositional logic and QBF we require a way to write our formulas that is convenient to work with. In this case we will use EPR, formally known as the *Bernays-Schönfinkel class* of formulas. A formula is in EPR form if when it is written in CNF it has the quantifier prefix $\exists * \forall *$ and contains no functions. This format will be useful because we can solve these problems using first order logic theorem provers.

1.2 Complexity of Satisfiability

Boolean Logics are significant in complexity theory as they are standard embeddings for other problems in their complexity classes.

1.2.1 SAT is NP complete

1.2.2 QBF is PSPACE complete

1.2.3 EPR is NEXPTIME complete

1.3 Automated Reasoning

Chapter 2

Converting QBF to EPR

acronym-ise header

Detailed conversion goes here

2.1 Raising QBF to First Order Logic

2.2 Removing Existential Quantifiers by Skolemization

2.3 Removing Function Symbols Introduced by Skolemization

2.4 Dependency Schemes

Chapter 3

Technical Details

probably give it a better name

3.1 Language Choice

3.2 Input and Output Formats

3.2.1 QDIMACS

3.2.2 TPTP

3.3 Data Structures

3.4 Algorithms

3.4.1 Skolemization

3.4.2 Removing Functions

3.4.3 Dependency Scheme Construction

3.5 Testing

Chapter 4

Future work

Future work goes here

4.1 Dependency Scheme Optimisations

4.2 Anti-prenexing

Chapter 5

Evaluation

evaluation goes here

5.1 Comparison Against Direct QBF Solvers

5.2 Comparison Against EPR Converters

Chapter 6

Project plan

project plan goes here

Chapter 7

Conclusion

conclusion goes here

Chapter 8

Bibliography

[1] University of Manchester logo from Wikipedia by source, fair use
https://en.wikipedia.org/w/index.php?curid=43485475
2014

[2] Korovin, Konstantin. "iProver-an instantiation-based theorem prover for first-order logic (system description)." Automated Reasoning. Springer Berlin Heidelberg, 2008. 292-298.

[3] Lonsing, Florian, and Armin Biere. "DepQBF: A dependency-aware QBF solver." Journal on Satisfiability, Boolean Modeling and Computation 7 (2010): 71-76.

[4] Seidl, Martina, Florian Lonsing and Armin Biere. "qbf2epr: A Tool for Generating EPR Formulas from QBF." PAAR@ IJCAR. 2012.

Acronyms