RAReQS: Recursive Abstraction Refinement QBF Solver

Mikoláš Janota

INESC-ID, Lisbon, Portugal

April 25, 2012

1 Overview

RAReQS is an abstraction based solver based on the ideas presented in [8] and is a follow-up of the 2QBF solver AReQS2 [7]. It uses `minisat2.2` [6] as the underlying SAT solver.

2 Algorithm

RAReQS enables solving QBF in the *closed prenex CNF* form, i.e. the input is in the form $Q_1 z_1 \ldots Q_n z_n.\phi$ where $Q_i \in \{\forall, \exists\}$, z_i are distinct variables, and ϕ is a CNF using only the variables z_i.

Internally a formula is represented as a *multi-game*, which intuitively corresponds to solving multiple QBF formulas at the same time.

Definition 1 (multi-game)**.** *A* multi-game *is denoted by* $QX.\{\Phi_1, \ldots, \Phi_n\}$ *where each* Φ_i *is a prenex QBF starting with* $\bar{Q}$ *or has no quantifiers. The free variables of each* Φ_i *must be in* X *and all* Φ_i *have the same number of quantifier blocks. We refer to the formulas* Φ_i *as* subgames *and* QX *as the* top-level prefix*.*

A winning move *for a multi-game is an assignment to the variables* X *such that it is a winning move for each of the formulas* $QX.\ \Phi_i$*.*

Algorithm 1 is used to solve multi-games. Algorithm 1 is a recursive algorithm that starts at the outermost quantifier. At each level, the algorithm constructs a multi-game that serves as an *abstraction* for the winning moves of the current level. This abstraction is gradually refined by recursive calls. This follows the well-known paradigm of counterexample guided abstraction refinement (CEGAR) [4].

2.1 Implementation Details

RAReQS is implemented in `C++`, supporting the QDIMACS format, with the underlying SAT solver `minisat 2.2` [6]. The implementation of Algorithm 1 has several distinctive features. In Algorithm 1 an abstraction computed within a sub-call is forgotten once the call returns. This may lead to repetition of work and hence the solver supports maintaining these abstractions and strengthening

Algorithm 1: Recursive CEGAR algorithm for multi-games

```
1  Function RAReQS (QX. {Φ_1, ..., Φ_n})
   output : a winning move for Q if there is one; NULL otherwise
2  begin
3      if Φ_i have no quantifiers then
4          return Q = ∃ ? SAT(⋀_i Φ_i) : SAT(¬(⋁_i Φ))
5      end
6      α ← QX. {}
7      while true do
8          τ' ← RAReQS(α)                    // find a candidate solution
9          if τ' = NULL then return NULL
10         τ ← {l | l ∈ τ' ∧ var(l) ∈ X}     // filter a move for X
11         for i ← 1 to n do μ_i ← RAReQS(Φ_i[τ])   // find a
           counterexample
12         if μ_i = NULL for all i ∈ {1..n} then return τ
13         let l ∈ {1..n} be s.t. μ_l ≠ NULL
14         α ← Refine(α, Φ_l, μ_l)                  // refine
15     end
16 end
```

them gradually, similarly to the way SAT solvers provide *incremental* interface. This incremental approach, however, tends to lead to unwieldy memory consumption and therefore, it is used only when the given multigame's subgames have 2 or fewer quantification blocks.

If an assignment τ is a candidate for a winning move that turns out *not* to be a winning move, the refinement guarantees that τ is not a solution to the abstraction in the future iterations of the CEGAR loop. This knowledge enables us to make the subcall for solving the abstraction more efficient by explicitly disabling τ as a winning move for the abstraction. We refer to this technique as *blocking* and it is similar to the refinement used in certain SMT solvers [5, 1].

Throughout its course, the algorithm may produce a large number of new formulas, either by substitution or refinement. Since these formulas tend to be simpler than the given one, they can be further simplified by standard QBF *preprocessing* techniques. The implementation uses unit propagation and *monotone* (pure) literal rule [3]. These simplifications introduce the complication that in a multi-game $QX.\{\Phi_1, \ldots, \Phi_n\}$ the individual subgames might not necessarily have the same number of quantifier levels. In such case, all games with no quantifiers are immediately put into the abstraction before the loop starts.

We submit to the competition one version with in one version without `bloqqer` as a preprocessor [2].

References

[1] Clark W. Barrett, David L. Dill, and Aaron Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In *CAV*, 2002.

[2] Armin Biere. Bloqqer preprocessor. http://fmv.jku.at/bloqqer/bloqqer-031-7a176af-110509.tar.gz.

[3] M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate Quantified Boolean Formulae. In *National Conference on Artificial Intelligence*, 1998.

[4] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM*, 50(5), 2003.

[5] Leonardo Mendonça de Moura, Harald Rueß, and Maria Sorea. Lazy theorem proving for bounded model checking over infinite domains. In *CADE*, 2002.

[6] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *SAT*, 2003.

[7] Mikolás Janota and João P. Marques Silva. Abstraction-based algorithm for 2QBF. In Karem A. Sakallah and Laurent Simon, editors, *SAT*, volume 6695 of *Lecture Notes in Computer Science*, pages 230–244. Springer, 2011.

[8] Mikoláš Janota, William Klieber, Joao Marques-Silva, and Edmund Clarke. Solving QBF with counterexample guided refinement. In *SAT, to appear*, 2012. Preprint available from the authors.