Certified DQBF Solving by Definition Extraction

Franz-Xaver Reichl Friedrich Slivovsky Stefan Szeider

International Conference on Theory and Applications of Satisfiability Testing

ALGORITHMS AND COMPLEXITY GROUP

Contents

Motivation

- DQBF allows succincter problem encodings than QBF or propositional logic:
  - Partial Equivalence Checking
  - Synthesis

Motivation

- DQBF allows succincter problem encodings than QBF or propositional logic:
  - Partial Equivalence Checking
  - Synthesis
- Yes/No answers do not always suffice.
  - Certificates increase confidence in results.
  - Applications may require a witness for the truth of a DQBF.

Dependency Quantified Boolean Formulae (DQBF)

Example

$\forall u_1, u_2\, \exists e_1(u_1), e_2(u_2).\ (u_1 \vee \neg e_1) \wedge (\neg u_1 \vee e_1) \wedge (u_2 \vee e_2)$

- *Prenex Conjunctive Normal Form* (PCNF)
- *Prefix*: $\forall u_1, u_2\, \exists e_1(u_1), e_2(u_2)$
- *Matrix*: $(u_1 \vee e_1) \wedge (u_2 \vee \neg e_2)$
- *Model*: $f_{e_1}(u) := u$, $f_{e_2}(u) := \neg u \rightsquigarrow$ the formula is *true*.

Contents

Propositional Definability

Definition (Propositional Definitions)

Let φ and ψ be propositional formulae and $v \in var(\varphi)$. ψ is a *definition* for v if for each model σ of φ we have: $\sigma(v) = \psi[\sigma]$

Propositional Definability

Definition (Propositional Definitions)

Let φ and ψ be propositional formulae and $v \in var(\varphi)$. ψ is a *definition* for v if for each model σ of φ we have: $\sigma(v) = \psi[\sigma]$

Example

Consider the formula $(u \vee \neg e) \wedge (\neg u \vee e)$. The variable e is defined by the formula u.

Propositional Definability

Definition (Propositional Definitions)

Let φ and ψ be propositional formulae and $v \in var(\varphi)$. ψ is a *definition* for v if for each model σ of φ we have: $\sigma(v) = \psi[\sigma]$

Example

Consider the formula $(u \vee \neg e) \wedge (\neg u \vee e)$. The variable e is defined by the formula u.

Lemma

Definitions can be computed by means of interpolants.

Using Definitions to Decide DQBF

- $\Phi := \forall U\, \exists e_1(D_1), \ldots, e_m(D_m).\ \varphi$ where each e_i has a definition ψ_i by D_i.
- $\neg\varphi \wedge \bigwedge_i (e_i \Leftrightarrow \psi_i)$ satisfiable iff Φ is false

Example

- $\forall u\, \exists e(u).\ (u \vee \neg e) \wedge (\neg u \vee e)$
- Definition for e: u
- $\neg((u \vee \neg e) \wedge (\neg u \vee e)) \wedge (e \Leftrightarrow u)$ is unsatisfiable.

Using Definitions to Decide DQBF

- An existential variable e may not be uniquely defined by an assignment σ to its dependencies.

Using Definitions to Decide DQBF

- An existential variable e may not be uniquely defined by an assignment σ to its dependencies.
- For each such e and σ introduce an *arbiter variable* a and *arbiter clauses*:
  - $a \vee \neg\sigma \vee \neg e$
  - $\neg a \vee \neg\sigma \vee e$

Using Definitions to Decide DQBF

- An existential variable e may not be uniquely defined by an assignment σ to its dependencies.
- For each such e and σ introduce an *arbiter variable* a and *arbiter clauses*:
  - $a \vee \neg\sigma \vee \neg e$
  - $\neg a \vee \neg\sigma \vee e$
- Given an assignment for a, e is uniquely determined by σ.

The Two-Phase Algorithm

procedure $\textsc{SolveByDefinitionExtraction}(\Phi)$
$\triangleright\ \Phi = \forall u_1, \ldots, u_n \exists e_1(D_1), \ldots, e_m(D_m).\ \varphi$
$A \leftarrow \emptyset, \varphi_A \leftarrow \emptyset$
for $i = 1, \ldots, m$ **do**
  while e_i is undefined **do**
    $\textsc{AddArbiter}(\Phi, \varphi_A, A)$
$Def \leftarrow \textsc{computeDefinitions}(\varphi, \varphi_A)$
$usedAssignments \leftarrow \emptyset, \tau \leftarrow \bigwedge_{a \in A} a$
loop
  if $\neg\varphi \wedge Def \wedge \tau$ is unsatisfiable **then**
    return TRUE
  $\sigma \leftarrow \textsc{getModel}(\neg\varphi \wedge Def \wedge \tau)$
  $\textsc{insert}(usedAssignments, \neg\textsc{getCore}(\varphi \wedge \varphi_A, \sigma)|_A)$
  if $usedAssignments$ is satisfiable **then**
    $\tau \leftarrow \textsc{getModel}(usedAssignments)$
  else
    return FALSE

Correctness

Lemma

Let Φ be a DQBF. Φ is true if, and only if, for each $e \in E$ there is a formula ψ_e with $var(\psi_e) \subseteq D(e)$ such that $\neg\varphi \wedge \bigwedge_{e \in E}(e \leftrightarrow \psi_e)$ is unsatisfiable.

- If the algorithm returns true $\neg\varphi \wedge \bigwedge_{e \in E}(e \leftrightarrow \psi_e[\tau])$ is unsatisfiable.
- For an existential variable e we can extract a model function from $\psi_e[\tau]$.

Completeness

∀Exp+Res

- Propositional resolution
- Instantiation

$$\frac{}{\{\ell^{\sigma}|_{D(var(\ell))} \mid \ell \in C, var(\ell) \in E\}}$$

- σ total assignment for U
- σ falsifies each universal literal in C

Completeness

$\forall$Exp+Res

- Propositional resolution
- Instantiation

$$\frac{}{\{\ell^{\sigma|_{D(var(\ell))}} \mid \ell \in C, var(\ell) \in E\}}$$

- σ total assignment for U
- σ falsifies each universal literal in C

- An arbiter variable a, introduced for σ and e can be associated to e^{σ}.
- If an arbiter assignment τ fails a subset of associated literals to $\neg\tau$ can be derived.
- If the algorithm returns false then there is a $\forall$Exp+Res proof.

Contents

Motivation

- Running time of the Two-Phase Algorithm is mainly determined by the number of assignments where an existential variable is not uniquely determined.
- Even for simple formulae the algorithm can get stuck.

Motivation

- Running time of the Two-Phase Algorithm is mainly determined by the number of assignments where an existential variable is not uniquely determined.
- Even for simple formulae the algorithm can get stuck.

$$\forall u_1, \ldots, u_n \, \exists e(u_1, \ldots, u_n).\ u_1 \vee \ldots \vee u_n \vee \neg e$$

- $2^n - 1$ arbiter variables need to be introduced.

Idea of the Algorithm

- Based on *Counter-Example Guided Inductive Synthesis* (CEGIS)
- Iteratively build a model.
- Find conflicts in matrix with respect to the current model.
- Use conflicts to refine the model.

The CEGIS-Algorithm

procedure $\textsc{SolveByDefinitionExtractionCEGIS}(\Phi)$
    $\triangleright\ \Phi = \forall u_1, \ldots, u_n\, \exists e_1(D_1), \ldots, e_m(D_m).\ \varphi$
    $A \leftarrow \emptyset, \varphi_A \leftarrow \emptyset, \tau \leftarrow \emptyset$
    $\mathit{usedAssignments} \leftarrow \emptyset$
    loop
        $\mathit{Def} \leftarrow \textsc{FindDefinitions}(\varphi, \varphi_A)$
        $\text{valid}, \sigma_\forall, \sigma_\exists \leftarrow \textsc{CheckArbiterAssignment}(\varphi, \mathit{Def}, \varphi_A, \tau)$
        if valid **then**
            return $\textsc{true}$
        $\textsc{AddArbiters}(A, \varphi_A, \sigma_\forall, \sigma_\exists, \tau)$
        $\hat{\tau} \leftarrow \textsc{getCore}(\varphi \wedge \varphi_A, \sigma_\forall \wedge \sigma_\exists \wedge \tau)|_A$
        $\textsc{insert}(\mathit{usedAssignments}, \neg\hat{\tau})$
        if *usedAssignments* is satisfiable **then**
            $\tau \leftarrow \textsc{getModel}(\mathit{usedAssignments})$
        else
            return $\textsc{false}$

Contents

Experimental Setup

- Pedant
  - Implementation of an improved version of the CEGIS Algorithm
  - Implemented in C++
  - Available at: `https://github.com/perebor/pedant-solver`

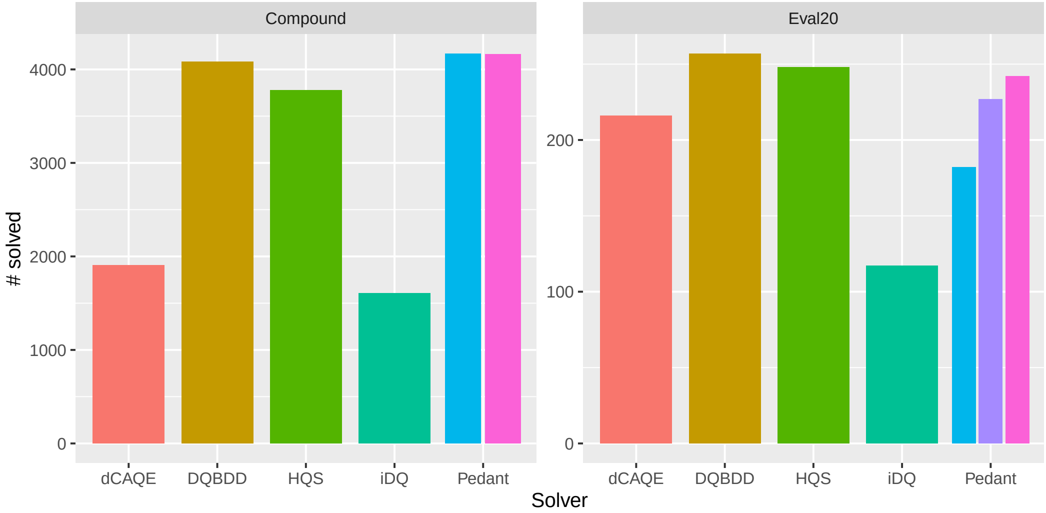
Experimental Setup

- Pedant
  - Implementation of an improved version of the CEGIS Algorithm
  - Implemented in C++
  - Available at: `https://github.com/perebor/pedant-solver`
- Benchmarks
  - Compound: `http://abs.informatik.uni-freiburg.de/src/projectfiles/21/DQBFBenchmarks.zip`
  - Eval20: `http://www.qbflib.org/QBFEVAL_20_DATASET.zip`

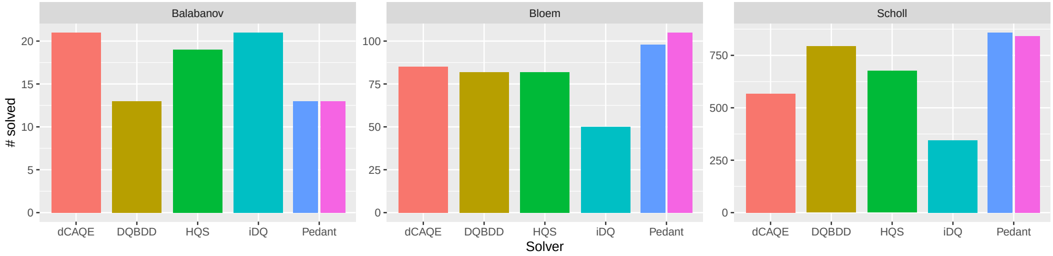
Experimental Setup

- Pedant
  - Implementation of an improved version of the CEGIS Algorithm
  - Implemented in C++
  - Available at: `https://github.com/perebor/pedant-solver`
- Benchmarks
  - Compound: `http://abs.informatik.uni-freiburg.de/src/projectfiles/21/DQBFBenchmarks.zip`
  - Eval20: `http://www.qbflib.org/QBFEVAL_20_DATASET.zip`
- Test Configuration
  - Timeout of 30 minutes
  - Memory limit of 8 GB

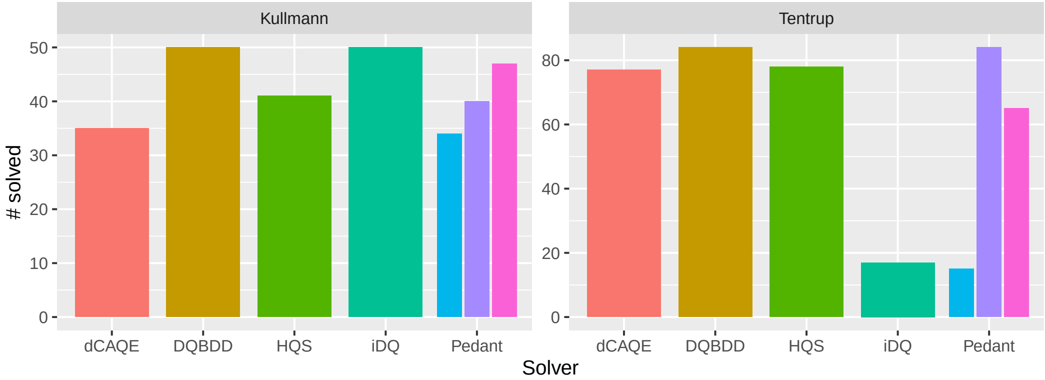
Experiments – Overview



Experiments – Compound



Experiments – Eval20



Conclusions

- Summary
  - We presented two decision procedures for DQBF based on definition extraction.
  - We proved their correctness and completeness.
  - We implemented and evaluated the CEGIS based algorithm.
  - We showed that our solver performs very well compared to state of the art solvers.

Conclusions

- Summary
  - We presented two decision procedures for DQBF based on definition extraction.
  - We proved their correctness and completeness.
  - We implemented and evaluated the CEGIS based algorithm.
  - We showed that our solver performs very well compared to state of the art solvers.
- Outlook
  - Arbiter variables for partial assignments.
  - Certificates for false results.