Solving QBF with Counterexample Guided Refinement

Mikoláš Janota[1] William Klieber[2]
Joao Marques-Silva[1,3] Edmund Clarke[2]

[1] INESC-ID/IST, Lisbon, Portugal
[2] Carnegie Mellon University, Pittsburgh, PA, USA
[3] CASL/CSI, University College Dublin, Ireland

QBF

- an extension of SAT with quantifiers
- PSPACE-complete
- formal verification
- planning

QBF

- an extension of SAT with quantifiers
- PSPACE-complete
- formal verification
- planning

Example

$$\forall y_1 y_2 \exists x_1 x_2.\ (y_1 \leftrightarrow x_1) \wedge (y_2 \leftrightarrow x_2)$$

QBF

- an extension of SAT with quantifiers
- PSPACE-complete
- formal verification
- planning

Example

$$\forall y_1 y_2 \exists x_1 x_2.\ (y_1 \leftrightarrow x_1) \land (y_2 \leftrightarrow x_2)$$

- we consider prenex form with maximal blocks of variables

$$Q X_1\ \bar{Q} Y_1\ Q X_2\ \bar{Q} Y_2 \ldots .\ \phi$$

$$\textit{where } Q \in \{\exists, \forall\}$$
$$\bar{\exists} = \forall, \bar{\forall} = \exists$$

A QBF as a Game

- it is useful to think about a QBF as a game between the universal and existential player
- universal player wins when the matrix becomes false
- existential player wins when the matrix becomes true

A QBF as a Game

- it is useful to think about a QBF as a game between the universal and existential player
- universal player wins when the matrix becomes false
- existential player wins when the matrix becomes true
- a QBF is true if and only if the *"exist player can always win"*

A QBF as a Game

- it is useful to think about a QBF as a game between the universal and existential player
- universal player wins when the matrix becomes false
- existential player wins when the matrix becomes true
- a QBF is true if and only if the *"exist player can always win"*

Example

$$\forall y_1 y_2 \exists x_1 x_2 .\ (y_1 \leftrightarrow x_1) \wedge (y_2 \leftrightarrow x_2)$$

- $\exists$ always wins by playing $x_1 = y_1$, $x_2 = y_2$

Semantics with Winning Move

winning move, base case $QX.\phi$, for ϕ propositional

- for $Q = \exists$, an assignment that makes ϕ true (model of ϕ)
- for $Q = \forall$, an assignment that makes ϕ false

Semantics with Winning Move

winning move, base case $QX.\phi$, for ϕ propositional

- for $Q = \exists$, an assignment that makes ϕ true (model of ϕ)
- for $Q = \forall$, an assignment that makes ϕ false

winning move, general case $QX.\ \Phi$, for Φ QBF

- an assignment τ s.t. there is no winning move for $\bar{Q}$ for $\Phi[\tau]$

Semantics with Winning Move

winning move, base case $QX.\phi$, for ϕ propositional

- for $Q = \exists$, an assignment that makes ϕ true (model of ϕ)
- for $Q = \forall$, an assignment that makes ϕ false

winning move, general case $QX.\ \Phi$, for Φ QBF

- an assignment τ s.t. there is no winning move for $\bar{Q}$ for $\Phi[\tau]$

countermove, for $QX.\ \Phi$, for Φ QBF

- an assignment μ is a countermove to the assignment τ if μ is a winning move for $\bar{Q}$ for $\Phi[\tau]$

Winning Move Semantics

QBF semantics

- $\exists X.\Phi$ is true if and only if there is a winning move for $\exists$
- $\forall X.\Phi$ is false if and only if there is a winning move for $\forall$

Winning Move Semantics

QBF semantics

- $\exists X.\Phi$ is true if and only if there is a winning move for $\exists$
- $\forall X.\Phi$ is false if and only if there is a winning move for $\forall$

Example

$$\forall y \exists x.\ x \wedge (y \vee \bar{x})$$

- $\{\bar{y}\}$ is a winning move for $\forall$, formula is false
- $\{y\}$ is not a winning move and $\{x\}$ is a countermove

Computing a Winning Move—Base Case

`Solve` ($\exists X.\ \phi$), where ϕ is a propositional
output : a winning move for $\exists$ if there is one; NULL otherwise
return `SAT`(ϕ)

Computing a Winning Move—Base Case

$\texttt{Solve}$ ($\exists X.\ \phi$), where ϕ is a propositional
output : a winning move for $\exists$ if there is one; NULL otherwise
return $\texttt{SAT}(\phi)$

$\texttt{Solve}$ ($\forall X.\ \phi$), where ϕ is a propositional
output : a winning move for $\forall$ if there is one; NULL otherwise
return $\texttt{SAT}(\neg\phi)$

Naive Algorithm for a Winning Move

```
Function Solve (QX. Φ)
Λ ← {true, false}^X                  // consider all assignments
while true do
    if Λ = ∅ then
        return NULL                  // all assignments used up
    τ ← pick(Λ)                      // pick a candidate solution
    μ ← Solve(Φ[τ])                  // find a countermove
    if μ = NULL then
        return τ                     // winning move
    Λ ← Λ ∖ {τ}                      // remove bad candidate
end
```

Removing More Than One Candidate at a Time

Observation

- The naive algorithm does not avail of the countermove

Removing More Than One Candidate at a Time

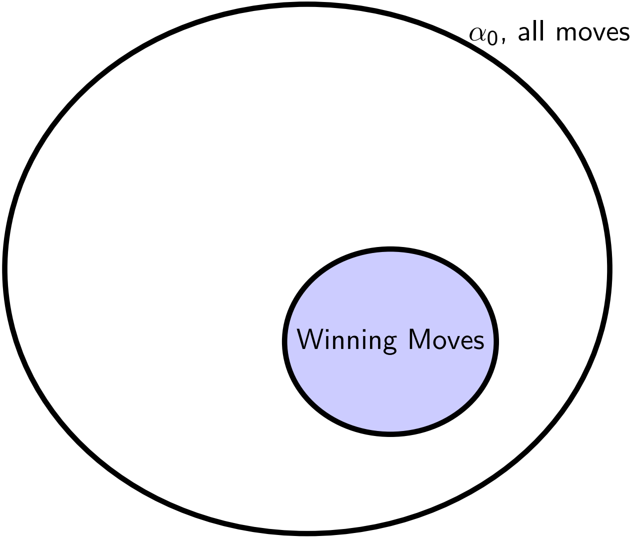
Observation

- The naive algorithm does not avail of the countermove

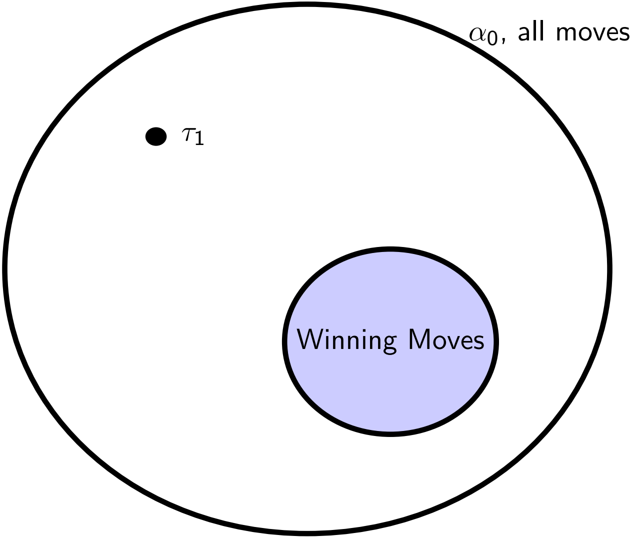
How?

- represent the set of considered candidates as the set of winning moves of a (simpler) QBF (abstraction)
- each time a countermove is found, strengthen the abstraction so that the same countermove cannot be used in the future (refinement)

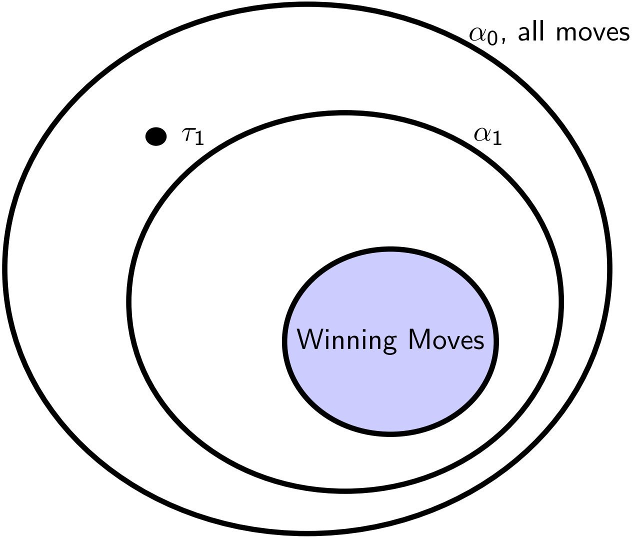
CounterExample Guided Abstraction Refinement (CEGAR)



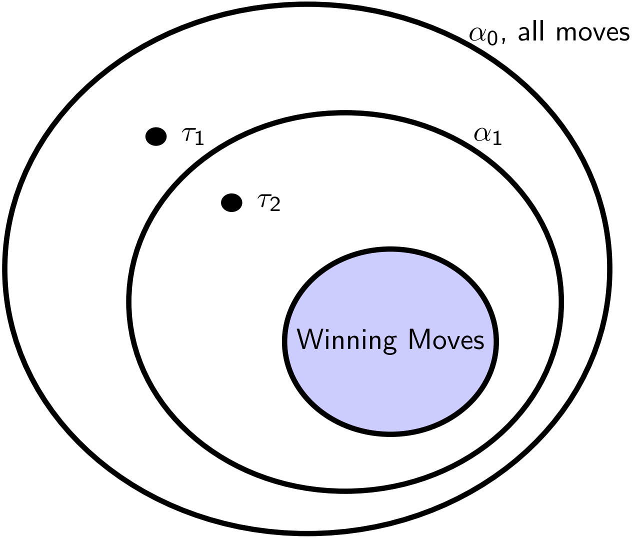
CounterExample Guided Abstraction Refinement (CEGAR)



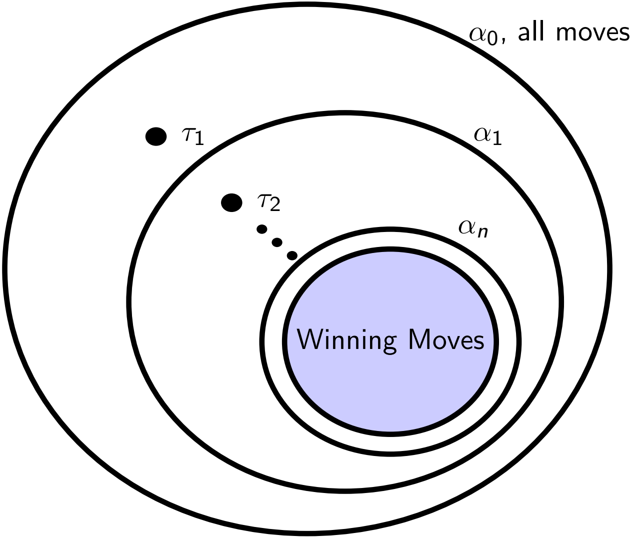
CounterExample Guided Abstraction Refinement (CEGAR)



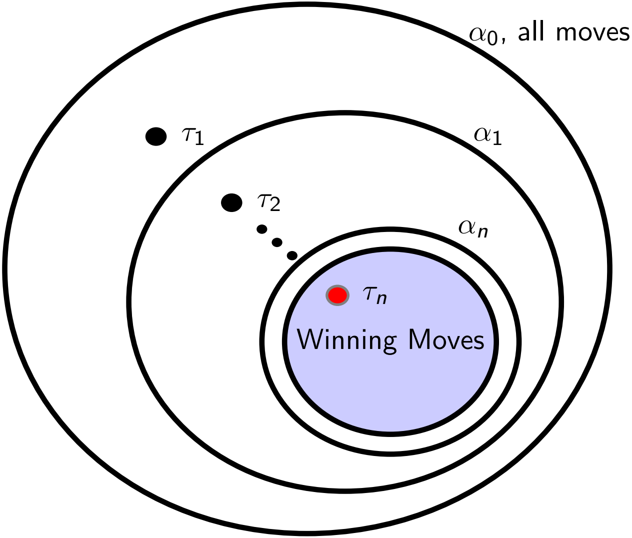
CounterExample Guided Abstraction Refinement (CEGAR)



CounterExample Guided Abstraction Refinement (CEGAR)



CounterExample Guided Abstraction Refinement (CEGAR)



Refinement

for a bad candidate τ

- $\bar{Q}$ wins by μ, i.e. $QX\bar{Q}Y.\ \Phi[\tau][\mu]$ is losing for Q

Refinement

for a bad candidate τ

- $\bar{Q}$ wins by μ, i.e. $QX\bar{Q}Y.\ \Phi[\tau][\mu]$ is losing for Q

for next candidates....

- make sure that next candidate τ is a winning move for $QX.\ \Phi[\mu]$
- for such τ, μ cannot be a countermove

Refinement

for a bad candidate τ

- $\bar{Q}$ wins by μ, i.e. $QX\bar{Q}Y.\ \Phi[\tau][\mu]$ is losing for Q

for next candidates....

- make sure that next candidate τ is a winning move for $QX.\ \Phi[\mu]$
- for such τ, μ cannot be a countermove

for a set of countermoves $\omega = \{\mu_1, \ldots, \mu_n\}$

- $\bigwedge_{\mu \in \omega} \Phi[\mu]$, $Q = \exists$
- $\bigvee_{\mu \in \omega} \Phi[\mu]$, $Q = \forall$

Refinement—Example

- $\forall y \exists x.\ x \wedge (y \vee \bar{x})$
- candidate: $\{y\}$, countermove: $\{x\}$
- abstraction: $\forall y.\ y$
  (with the single winning move $\{\bar{y}\}$)

Refinement—Example

- $\forall y \exists x.\ x \wedge (y \vee \bar{x})$
- candidate: $\{y\}$, countermove: $\{x\}$
- abstraction: $\forall y.\ y$
  (with the single winning move $\{\bar{y}\}$)
- $\forall y_1 y_2 \exists x.\ (y_1 \vee \bar{x}) \wedge (y_2 \vee x)$

Refinement—Example

- $\forall y \exists x.\ x \wedge (y \vee \bar{x})$
- candidate: $\{y\}$, countermove: $\{x\}$
- abstraction: $\forall y.\ y$
  (with the single winning move $\{\bar{y}\}$)

- $\forall y_1 y_2 \exists x.\ (y_1 \vee \bar{x}) \wedge (y_2 \vee x)$
- candidate: $\{y_1, \bar{y}_2\}$, countermove: $\{x\}$ $(\Phi[x] = y_1)$

Refinement—Example

- $\forall y \exists x.\ x \wedge (y \vee \bar{x})$
- candidate: $\{y\}$, countermove: $\{x\}$
- abstraction: $\forall y.\ y$
  (with the single winning move $\{\bar{y}\}$)

- $\forall y_1 y_2 \exists x.\ (y_1 \vee \bar{x}) \wedge (y_2 \vee x)$
- candidate: $\{y_1, \bar{y}_2\}$, countermove: $\{x\}$ $(\Phi[x] = y_1)$
- candidate: $\{\bar{y}_1, y_2\}$, countermove: $\{\bar{x}\}$ $(\Phi[\bar{x}] = y_2)$

Refinement—Example

- $\forall y \exists x.\ x \wedge (y \vee \bar{x})$
- candidate: $\{y\}$, countermove: $\{x\}$
- abstraction: $\forall y.\ y$
  (with the single winning move $\{\bar{y}\}$)

- $\forall y_1 y_2 \exists x.\ (y_1 \vee \bar{x}) \wedge (y_2 \vee x)$
- candidate: $\{y_1, \bar{y}_2\}$, countermove: $\{x\}$ $(\Phi[x] = y_1)$
- candidate: $\{\bar{y}_1, y_2\}$, countermove: $\{\bar{x}\}$ $(\Phi[\bar{x}] = y_2)$
- abstraction: $\forall y_1 y_2.\ y_1 \vee y_2$
  (with the single winning move $\{\bar{y}_1, \bar{y}_2\}$)

Abstraction-Based Algorithm for a Winning Move

```
Function Solve (QX. Φ)
begin
    if Φ has no quant then
        return (Q = ∃) ? SAT(φ) : SAT(¬φ)
    ω ← ∅
    while true do
        α ← (Q = ∃) ? ⋀_{μ∈ω} Φ[μ] : ⋁_{μ∈ω} Φ[μ]   // abstraction
        τ' ← Solve(Prenex(QX. α))                   // find a candidate
        if τ' = NULL then return NULL               // no winning move
        τ ← {l | l ∈ τ' ∧ var(l) ∈ X}               // filter a move for X
        μ ← Solve(Φ[τ])                             // find a countermove
        if μ = NULL then return τ                   // winning move
        ω ← ω ∪ {μ}                                 // refine
    end
end
```

Results

- RAReQS implementation of the above using minisat2.2

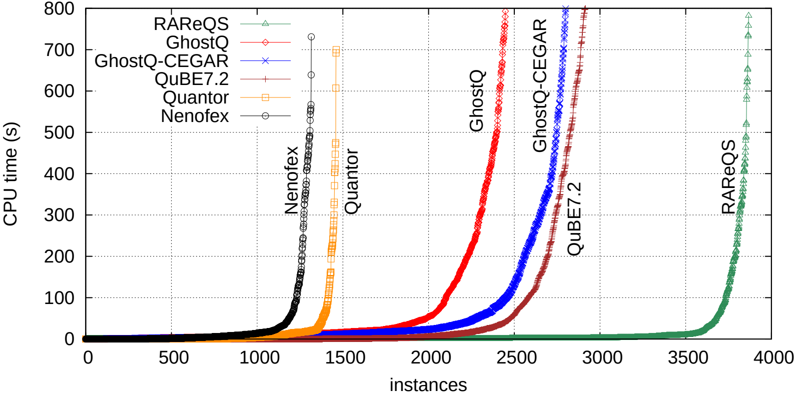
Results

- RAReQS implementation of the above using minisat2.2
- GhostQ-CEGAR integration into an existing DPLL solver

Results

- RAReQS implementation of the above using minisat2.2
- GhostQ-CEGAR integration into an existing DPLL solver

Results for planning and Formal Verification families



Conclusions and Future Work

- a novel CEGAR-based technique for QBF solving RAReQS

Conclusions and Future Work

- a novel CEGAR-based technique for QBF solving RAReQS
- in some sense RAReQS is close to expansion-based solvers (e.g. Quantor, Nenofex) but the expansion is driven by counterexamples

Conclusions and Future Work

- a novel CEGAR-based technique for QBF solving RAReQS
- in some sense RAReQS is close to expansion-based solvers (e.g. Quantor, Nenofex) but the expansion is driven by counterexamples
- step-by-step expansion enables RAReQS to avoid inherent memory blowup of expansion solvers

Conclusions and Future Work

- a novel CEGAR-based technique for QBF solving RAReQS
- in some sense RAReQS is close to expansion-based solvers (e.g. Quantor, Nenofex) but the expansion is driven by counterexamples
- step-by-step expansion enables RAReQS to avoid inherent memory blowup of expansion solvers
- enables solving a large number of practical instances not solved by state-of-the-art solvers
  (220 instances that only RAReQS solved)

Conclusions and Future Work

- a novel CEGAR-based technique for QBF solving RAReQS
- in some sense RAReQS is close to expansion-based solvers (e.g. Quantor, Nenofex) but the expansion is driven by counterexamples
- step-by-step expansion enables RAReQS to avoid inherent memory blowup of expansion solvers
- enables solving a large number of practical instances not solved by state-of-the-art solvers
  (220 instances that only RAReQS solved)
- in the future we plan to further develop the integration between DPLL and CEGAR

Conclusions and Future Work

- a novel CEGAR-based technique for QBF solving RAReQS
- in some sense RAReQS is close to expansion-based solvers (e.g. Quantor, Nenofex) but the expansion is driven by counterexamples
- step-by-step expansion enables RAReQS to avoid inherent memory blowup of expansion solvers
- enables solving a large number of practical instances not solved by state-of-the-art solvers
  (220 instances that only RAReQS solved)
- in the future we plan to further develop the integration between DPLL and CEGAR
- in RAReQS we plan to investigate how to integrate techniques used in other solvers (e.g. dependency detection)

Questions?

Results

Total instances solved (out of 4669):

RAReQS	GhostQ	GhostQ-C	Qube	Quantor	Nenofex
3868	2449	2801	2916	1462	1317