Quantified Boolean Formulae

SAT - for a Boolean formula, determine if it is satisfiable
Quantified Boolean Formulae

SAT - for a Boolean formula, determine if it is satisfiable

QBF - for a Quantified Boolean formula, determine if it is true
Quantified Boolean Formulae

SAT - for a Boolean formula, determine if it is satisfiable

QBF - for a Quantified Boolean formula, determine if it is true

Example:
Quantified Boolean Formulae

SAT - for a Boolean formula, determine if it is satisfiable

QBF - for a Quantified Boolean formula, determine if it is true

Example:
\( \forall x \exists y. (x \leftrightarrow y) \)
SAT - for a Boolean formula, determine if it is satisfiable

QBF - for a Quantified Boolean formula, determine if it is true

Example:
$\forall x \exists y. (x \leftrightarrow y)$
$\forall x. (x \leftrightarrow 0) \lor (x \leftrightarrow 1)$
Quantified Boolean Formulae

SAT - for a Boolean formula, determine if it is satisfiable

QBF - for a Quantified Boolean formula, determine if it is true

Example:
\[ \forall x \exists y. (x \leftrightarrow y) \]
\[ \forall x. (x \leftrightarrow 0) \lor (x \leftrightarrow 1) \]
\[ ((0 \leftrightarrow 0) \lor (0 \leftrightarrow 1)) \land ((1 \leftrightarrow 0) \lor (1 \leftrightarrow 1)) \]
\[ 1 \]

QBF is the paradigmatic PSPACE-complete problem
Applications of Quantified Boolean Formulae

- Model checking
- Circuit synthesis
- Non-monotonic reasoning
- Conformant planning
- ...

Janota and Silva
Machine Learning of strategies for efficiently solving QBF with abstraction refinement
We consider prenex form: Quantifier-prefix. Matrix

Example: $\forall x_1x_2 \exists y_1y_2. (\neg x_1 \lor y_1) \land (x_2 \lor \neg y_2)$
Games and strategies

- We consider **prenex form**: *Quantifier-prefix*. Matrix

  **Example** $\forall x_1 x_2 \exists y_1 y_2. (\neg x_1 \lor y_1) \land (x_2 \lor \neg y_2)$

- A QBF represents a two-player game between $\forall$ and $\exists$. 
Games and strategies

- We consider **prenex form**: *Quantifier-prefix*. Matrix
  - **Example** $\forall x_1 x_2 \exists y_1 y_2. (\neg x_1 \lor y_1) \land (x_2 \lor \neg y_2)$
- A QBF represents a two-player game between $\forall$ and $\exists$.
- $\forall$ wins the game if the matrix becomes false.
Games and strategies

- We consider prenex form: Quantifier-prefix. Matrix
  Example $\forall x_1 x_2 \exists y_1 y_2. (\neg x_1 \lor y_1) \land (x_2 \lor \neg y_2)$
- A QBF represents a two-player game between $\forall$ and $\exists$.
  $\forall$ wins the game if the matrix becomes false.
  $\exists$ wins the game if the matrix becomes true.
Games and strategies

- We consider **prenex form**: Quantifier-prefix. Matrix

  **Example** \( \forall x_1 x_2 \exists y_1 y_2. (\neg x_1 \lor y_1) \land (x_2 \lor \neg y_2) \)

- A QBF represents a two-player game between \( \forall \) and \( \exists \).
- \( \forall \) wins the game if the matrix becomes false.
- \( \exists \) wins the game if the matrix becomes true.
- A QBF is false iff there exists a winning strategy for \( \forall \).
Games and strategies

- We consider prenex form: Quantifier-prefix. Matrix Example: \( \forall x_1 x_2 \exists y_1 y_2 . (\neg x_1 \lor y_1) \land (x_2 \lor \neg y_2) \)
- A QBF represents a two-player game between \( \forall \) and \( \exists \).
- \( \forall \) wins the game if the matrix becomes false.
- \( \exists \) wins the game if the matrix becomes true.
- A QBF is false iff there exists a winning strategy for \( \forall \).
- A QBF is true iff there exists a winning strategy for \( \exists \).
We consider prenex form: Quantifier-prefix. Matrix Example
\[ \forall x_1 x_2 \exists y_1 y_2. (\neg x_1 \lor y_1) \land (x_2 \lor \neg y_2) \]
A QBF represents a two-player game between \( \forall \) and \( \exists \).
\( \forall \) wins the game if the matrix becomes false.
\( \exists \) wins the game if the matrix becomes true.
A QBF is false iff there exists a winning strategy for \( \forall \).
A QBF is true iff there exists a winning strategy for \( \exists \).
Example
\[ \forall u \exists e. u \leftrightarrow e \]
Solving QBF: DPLL versus expansion based solvers
Solving by Expansion

\[ \exists x_1 \ldots x_m \forall y_1 \ldots y_m. \phi \]

Equivalent to:

\[ (\exists x_1 \ldots x_m) y_1, \ldots, y_{n-1}, y_n \land \phi[0], \ldots, y_n \land \phi[0], \ldots, y_{n-1}, y_n \land \phi[1], \ldots, y_{n-1}, y_n \land \phi[1], \ldots, y_{n-1}, y_n \land \ldots \land \phi[1], \ldots, y_{n-1}, y_n \]
Solving by Expansion

\[ \exists x_1 \ldots x_m \forall y_1 \ldots y_m. \phi \]

Equivalent to:

\[
(\exists x_1 \ldots x_m)
\wedge \phi[0, \ldots, 0, 0]
\wedge \phi[0, \ldots, 0, 1]
\wedge \phi[0, \ldots, 1, 0]
\wedge \phi[0, \ldots, 1, 1]
\wedge \ldots
\wedge \phi[1, \ldots, 1, 1]
\]
Solving by Expansion

\[ \exists x_1 \ldots x_m \forall y_1 \ldots y_m. \phi \]

Equivalent to:

\[(\exists x_1 \ldots x_m) \land \phi[0, \ldots, 0, 0] \land \phi[0, \ldots, 0, 1] \land \phi[0, \ldots, 1, 0] \land \phi[0, \ldots, 1, 1] \land \ldots \land \phi[1, \ldots, 1, 1]\]

- Now only one type of quantifier: \(\exists x_1 \ldots x_m\)
Solving by Expansion

\[ \exists x_1 \ldots x_m \forall y_1 \ldots y_m. \phi \]

Equivalent to:

\[
(\exists x_1 \ldots x_m) \land \
\begin{align*}
\land & \phi[0, \ldots, 0, 0] \\
\land & \phi[0, \ldots, 0, 1] \\
\land & \phi[0, \ldots, 1, 0] \\
\land & \phi[0, \ldots, 1, 1] \\
\land & \ldots \\
\land & \phi[1, \ldots, 1, 1]
\end{align*}
\]

- Now only one type of quantifier: \( \exists x_1 \ldots x_m \)
- We can call a SAT solver!
Solving by Expansion

\[ \exists x_1 \ldots x_m \forall y_1 \ldots y_m. \phi \]

Equivalent to:

\[ (\exists x_1 \ldots x_m) \wedge \phi[0, \ldots, 0, 0] \wedge \phi[0, \ldots, 0, 1] \wedge \phi[0, \ldots, 1, 0] \wedge \phi[0, \ldots, 1, 1] \wedge \ldots \wedge \phi[1, \ldots, 1, 1] \]

\[ \begin{cases} 2^n \\ \end{cases} \]

- Now only one type of quantifier: \( \exists x_1 \ldots x_m \)
- We can call a SAT solver!
Expanding everything = exponential blow-up
Solving by Expansion (contd.)

- Expanding everything = exponential blow-up
- Do we need to expand everything?
Expanding everything = exponential blow-up
Do we need to expand everything?
Example:
$$\exists x_1 x_2 \forall y_1 y_2. (x_1 \land x_2 \land y_1 \land y_2)$$
Expanding everything = exponential blow-up
Do we need to expand everything?

Example:
\[ \exists x_1 x_2 \forall y_1 y_2. (x_1 \land x_2 \land y_1 \land y_2) \]

... sufficient to expand \( y_1 = y_2 = 0 \)
Solving by Expansion (contd.)

- Expanding everything = exponential blow-up
- Do we need to expand everything?
- **Example:**
  \[ \exists x_1 x_2 \forall y_1 y_2. (x_1 \land x_2 \land y_1 \land y_2) \]

- ... sufficient to expand \( y_1 = y_2 = 0 \)
- How to come up with the right expansions?
CEGAR paradigm: careful expansion

- Expand the formula **gradually**, to avoid exponential blow-up.
- ... means gradually strengthening **abstraction** of the formula.
CEGAR paradigm: careful expansion

- Expand the formula **gradually**, to avoid exponential blow-up.
- ... means gradually strengthening **abstraction** of the formula.

---

**Refine** = expand more

```
refine
→ Is there a solution for the abstraction? yes ↓ no → return false
   yes ↓ no → return true
```

CEGAR loop
CEGAR paradigm: careful expansion

- Expand the formula **gradually**, to avoid exponential blow-up.
- ... means gradually strengthening **abstraction** of the formula.

```
refine = expand more

Is there a solution for the abstraction?
  yes
  Is there a counterexample to the solution?
    no  return false
  no  return true

CEGAR loop

AReQS: CEGAR-based solver for 2-QBF [J. and Silva SAT’11]
```
CEGAR paradigm: careful expansion

- Expand the formula **gradually**, to avoid exponential blow-up.
- ... means gradually strengthening **abstraction** of the formula.

```
refine → Is there a solution for the abstraction? 
          yes ↓ 
          no → return false

refine yes Is there a counterexample to the solution? no → return true
```

CEGAR loop

refine = expand more

**AReQS**: CEGAR-based solver for 2-QBF [J. and Silva SAT’11]
**RAReQS**: generalises AReQS through recursion [J. et al. SAT’12]
Issue with Expansion

Example

\[ \exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1\ldots100} (x_i \neq y_i) \]
Issue with Expansion

**Example**

\[ \exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1 \ldots 100} (x_i \neq y_i) \]

**Move**

Counter-move
Issue with Expansion

**Example**

\[ \exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100} \cdot \bigvee_{i=1\ldots100} (x_i \neq y_i) \]

**Move**

000 \ldots 0001

**Counter-move**
Issue with Expansion

**Example**

\[ \exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1 \ldots 100} (x_i \neq y_i) \]

Move  Counter-move
000 \ldots 0001  000 \ldots 0001

Expansion necessarily exponential
Issue with Expansion

Example

\[ \exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1 \ldots 100} (x_i \neq y_i) \]

Move | Counter-move
---|---
000 \ldots 0001 | 000 \ldots 0001
000 \ldots 0010 | 000 \ldots 0010
Issue with Expansion

Example

$$\exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100} \cdot \bigvee_{i=1\ldots100} (x_i \neq y_i)$$

Move | Counter-move
---|---
000...0001 | 000...0001
000...0010 | 000...0010
Issue with Expansion

**Example**

\[ \exists x_1 \ldots x_{100} \land y_1 \ldots y_{100}. \bigvee_{i=1 \ldots 100} (x_i \neq y_i) \]

<table>
<thead>
<tr>
<th>Move</th>
<th>Counter-move</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 \ldots 0001</td>
<td>000 \ldots 0001</td>
</tr>
<tr>
<td>000 \ldots 0010</td>
<td>000 \ldots 0010</td>
</tr>
<tr>
<td>000 \ldots 0011</td>
<td></td>
</tr>
</tbody>
</table>
### Issue with Expansion

#### Example

$$\exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1\ldots100} (x_i \neq y_i)$$

<table>
<thead>
<tr>
<th>Move</th>
<th>Counter-move</th>
</tr>
</thead>
<tbody>
<tr>
<td>000...0001</td>
<td>000...0001</td>
</tr>
<tr>
<td>000...0010</td>
<td>000...0010</td>
</tr>
<tr>
<td>000...0011</td>
<td>000...0011</td>
</tr>
</tbody>
</table>

Move Counter-move
000...0001 000...0001
000...0010 000...0010
000...0011 000...0011
Issue with Expansion

**Example**

\[ \exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1\ldots100} (x_i \neq y_i) \]

<table>
<thead>
<tr>
<th>Move</th>
<th>Counter-move</th>
</tr>
</thead>
<tbody>
<tr>
<td>000...0001</td>
<td>000...0001</td>
</tr>
<tr>
<td>000...0010</td>
<td>000...0010</td>
</tr>
<tr>
<td>000...0011</td>
<td>000...0011</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

Expansion necessarily exponential
Issue with Expansion

Example

\[ \exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100} \cdot \bigvee_{i=1\ldots100} (x_i \neq y_i) \]

Move | Counter-move
--- | ---
000 \ldots 0001 | 000 \ldots 0001
000 \ldots 0010 | 000 \ldots 0010
000 \ldots 0011 | 000 \ldots 0011
\vdots

Expansion necessarily exponential
Expansion by Strategies

Example

\[ \exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1\ldots100} (x_i \neq y_i) \]

has exponential expansion by constants 0/1
Example

\[ \exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1 \ldots 100} (x_i \neq y_i) \]

- has exponential expansion by constants 0/1
- **BUT**: \( \forall \) has a short **winning strategy**
Expansion by Strategies

Example

$$\exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1 \ldots 100} (x_i \neq y_i)$$

- has exponential expansion by constants 0/1
- **BUT**: $\forall$ has a short *winning strategy*
- $\forall$-player wins by playing $y_i \triangleq x_i$.

QUESTION: So, how do we obtain good strategies?
Expansion by Strategies

Example

\[ \exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100} \cdot \bigvee_{i=1\ldots100} (x_i \neq y_i) \]

- has exponential expansion by constants 0/1
- **BUT:** \( \forall \) has a short **winning strategy**
- \( \forall \)-player wins by playing \( y_i \triangleq x_i \).
- Plug it in gives UNSAT:

\[ \exists x_1 \ldots x_{100}. \bigvee_{i=1\ldots100} (x_i \neq x_i) \]
Expansion by Strategies

Example

\[ \exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1 \ldots 100} (x_i \neq y_i) \]

- has exponential expansion by constants 0/1
- **BUT**: \( \forall \) has a short winning strategy
- \( \forall \)-player wins by playing \( y_i \triangleq x_i \).
- Plug it in gives UNSAT:

\[ \exists x_1 \ldots x_{100}. \bigvee_{i=1 \ldots 100} (x_i \neq x_i) \]

- \( y_i \triangleq x_i \) is winning strategy for \( \forall \)
Expansion by Strategies

Example

\[ \exists x_1 \ldots x_{100} \forall y_1 \ldots y_{100}. \bigvee_{i=1\ldots100} (x_i \neq y_i) \]

- has exponential expansion by constants 0/1
- **BUT:** \( \forall \) has a short **winning strategy**
- \( \forall \)-player wins by playing \( y_i \triangleq x_i \).
- Plug it in gives UNSAT:

\[ \exists x_1 \ldots x_{100}. \bigvee_{i=1\ldots100} (x_i \neq x_i) \]

- \( y_i \triangleq x_i \) is **winning strategy** for \( \forall \)
- **QUESTION:** So, how do we obtain good strategies?
QUESTION: So, how do we obtain good strategies?
QUESTION: So, how do we obtain good strategies?
Use machine learning
  - repeatedly during the execution of the solver
  - on previous moves and counter-moves
QUESTION: So, how do we obtain good strategies?
Use machine learning
  ▶ repeatedly during the execution of the solver
  ▶ on previous moves and counter-moves
periodically refine abstraction with the learned strategy
- **QUESTION**: So, how do we obtain good strategies?
- Use machine learning
  - **repeatedly during** the execution of the solver
  - **on** previous moves and counter-moves
- **periodically** refine abstraction with the learned strategy

**Example:**

<table>
<thead>
<tr>
<th>Move</th>
<th>Counter-move</th>
</tr>
</thead>
<tbody>
<tr>
<td>000...001</td>
<td>000...001</td>
</tr>
<tr>
<td>100...1000</td>
<td>000...1000</td>
</tr>
<tr>
<td>010...0011</td>
<td>010...0011</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
QUESTION: So, how do we obtain good strategies?

- Use machine learning
  - repeatedly during the execution of the solver
  - on previous moves and counter-moves

- periodically refine abstraction with the learned strategy

Example:
Move | Counter-move
---|---
000...0001 | 000...0001
100...1000 | 000...1000
010...0011 | 010...0011

\[ y_i \triangleq x_i \] is learnt from \( \ll 2^n \) expansions

[\text{J. AAAI’18}]
Learning occurs during the solver’s execution, therefore we have a tight time constraint.
Learning occurs during the solver’s execution, therefore we have a tight time constraint.

We need to learn boolean formulae that can be passed onto the solver.
Requirements on ML

- Learning occurs during the solver’s execution, therefore we have a tight time constraint
- We need to learn boolean formulae that can be passed onto the solver
- Our samples are small (especially for ML standards), but the number of variables can be quite large.
Alternative to Decision Trees: Decision Lists
- $k$-decision list . . . each rule at most $k$ literals
- $k$-decision list are PAC-learnable [Rivest ’87]
- $k$-decision list . . . each rule at most $k$ literals
- $k$-decision list are PAC-learnable [Rivest ’87]

Decision Lists and Rivest ($k = 2$)
Learning Algorithms

Greedy3

Janota and Silva Machine Learning of strategies for efficiently solving QBF with abstraction refinement 16 / 27
Grove
Learning Algorithms

Simple

No learning
simple-i16
simple-i32
simple-i64
simple-i128

CPU time (s)
instances
Learning Algorithms

CN2

CPU time (s)

instances

No learning
cn2m2-i16
cn2m2-i32
cn2m2-i64
cn2m2-i128

Janota and Silva
Machine Learning of strategies for efficiently solving QBF with abstraction refinement
Learning Algorithms

CN2

CPU time (s)

instances

No learning
cn2m3-i16
cn2m3-i32
cn2m3-i64
cn2m3-i128

Janota and Silva

Machine Learning of strategies for efficiently solving QBF with abstraction refinement
Learning Algorithms

CN2

CPU time (s)
instances
No learning
cn2m4-i16
cn2m4-i32
cn2m4-i64
cn2m4-i128

Janota and Silva
Machine Learning of strategies for efficiently solving QBF with abstraction refinement
270 instances, consisting of QBF encodings for a number of basic building blocks of circuits
Families of Formulae

Toy

270 instances, consisting of QBF encodings for a number of basic building blocks of circuits
126 instances of encodings for generalized buffer specification
Families of Formulae

126 instances of encodings for generalized buffer specification
Families of Formulae

48 instances encoding specifications of a driver for a hard disk controller
Families of Formulae

48 instances encoding specifications of a driver for a hard disk controller
Families of Formulae

522 QBFs encoding the specifications for circuits that perform a single matrix multiplication, or repeated multiplication with a subset of controllable inputs.
Families of Formulae

522 QBFs encoding the specifications for circuits that perform a single matrix multiplication, or repeated multiplication with a subset of controllable inputs.
Conclusions

- Machine learning of strategies during the solving of QBF with counter-example guided abstraction refinement is feasible and enables improvements in the solver’s performance.
Conclusions and Future Work

Conclusions

- Machine learning of strategies during the solving of QBF with counter-example guided abstraction refinement is feasible and enables improvements in the solver’s performance.
- Learning only takes a small fraction of the total solving time, so the crucial point is the quality of the strategies learned.
Conclusions

- Machine learning of strategies during the solving of QBF with counter-example guided abstraction refinement is feasible and enables improvements in the solver’s performance.
- Learning only takes a small fraction of the total solving time, so the crucial point is the quality of the strategies learned.
- Using beam search to select literals is feasible. More complex learning algorithms might be suitable candidates to improve QFUN.
Conclusions

- Machine learning of strategies during the solving of QBF with counter-example guided abstraction refinement is feasible and enables improvements in the solver’s performance.

- Learning only takes a small fraction of the total solving time, so the crucial point is the quality of the strategies learned.

- Using beam search to select literals is feasible. More complex learning algorithms might be suitable candidates to improve QFUN.

- For some families of QBF, QFUN with learning is particularly useful, namely the families toy, genbuf, driver and cycle-sched.
Future Work

- Learn strategies for multiple variables at once.
Conclusions and Future Work

Future Work

- Learn strategies for multiple variables at once.
- Implement a look-ahead algorithm
Future Work

- Learn strategies for multiple variables at once.
- Implement a look-ahead algorithm
- Dynamic learning intervals
Conclusions and Future Work

Future Work

- Learn strategies for multiple variables at once.
- Implement a look-ahead algorithm
- Dynamic learning intervals
- Incremental learning

Janota and Silva
Machine Learning of strategies for efficiently solving QBF with abstraction refinement
Conclusions and Future Work

Future Work

- Learn strategies for multiple variables at once.
- Implement a look-ahead algorithm
- Dynamic learning intervals
- Incremental learning
- Improve the analysis of families of QBFs.
Conclusions and Future Work

Future Work

- Learn strategies for multiple variables at once.
- Implement a look-ahead algorithm
- Dynamic learning intervals
- Incremental learning
- Improve the analysis of families of QBFs.
- Parallelism / solver portfolio