MANCHESTER 1824

The University of Manchester

Preprocessing for first-order clausification

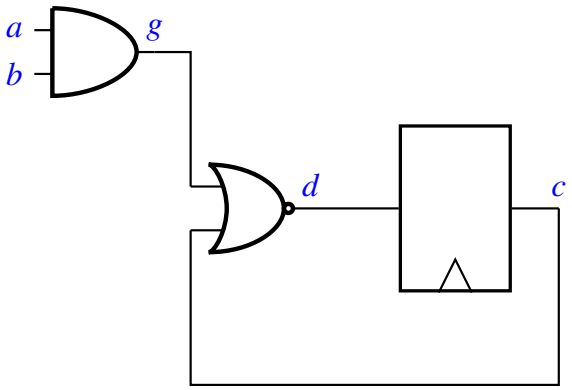
Krystof Hoder, Konstantin Korovin, Andrei Voronkov
Univeristy of Manchester

Zurab Khasidashvili
Intel Israel Development Center, Haifa

Outline

- EPR-based BMC
- EPR-preserving clausification
- Definition inlining
- Equivalence discovery using SAT sweeping
- Quantified AIGs (QAIGs) and QBDDs
- QAIGs + QBDDs
- Evaluation

Bounded model checking



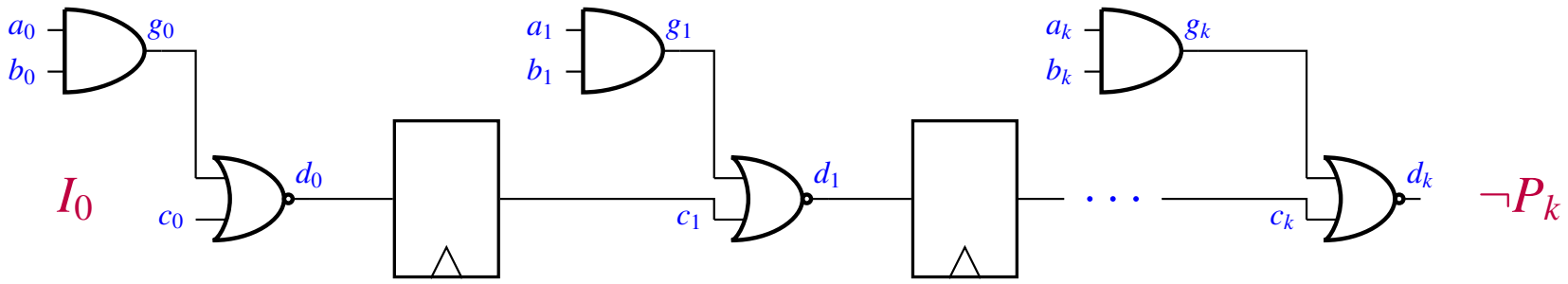
Symbolic representation:

$$
\begin{aligned}
I = \quad & (a_0 \leftrightarrow \neg c_0) \wedge (c_0 \rightarrow b_0) \\
& (g_0 \leftrightarrow a_0 \wedge b_0) \wedge (d_0 \leftrightarrow \neg g_0 \wedge \neg c_0)
\end{aligned}
$$

$$
\begin{aligned}
T = \quad & \\
& a' \leftrightarrow a && \wedge \\
& b' \leftrightarrow b && \wedge \\
& g' \leftrightarrow a' \wedge b' && \wedge \\
& c' \leftrightarrow d && \wedge \\
& d' \leftrightarrow \neg c' \wedge \neg g'
\end{aligned}
$$

$$P = \quad (d \leftrightarrow \neg g)$$

Bounded model checking (Unrolling)



The system is unsafe if and only if

$$I_0 \wedge T_{<1,2>} \wedge \ldots \wedge T_{<k-1,k>} \wedge \neg P_k$$

is satisfiable for some k.

Biere, Cimatti, Clarke, Zhu

First-order encoding of BMC

First-order encoding:

- $s_0, \ldots, s_k$ constants denoting states at unrolling bounds
- first-order formulas $I(S), P(S), T(S, S')$
- next state predicate $Next(S, S')$

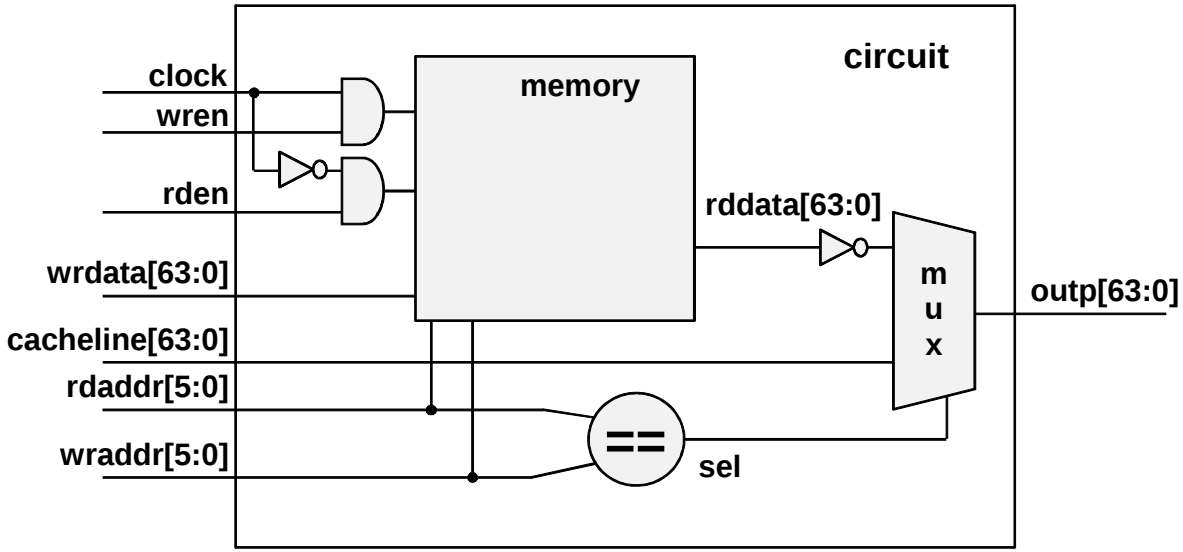
BMC can be encoded

$I(s_0); \neg P(s_k);$	initial and final states
$\forall S, S'(Next(S, S') \rightarrow T(S, S'));$	transition relation
$Next(s_0, s_1); Next(s_1, s_2); \ldots Next(s_{k-1}, s_k);$	next state relation

- FOL encoding provides succinct representation
- Reasoning can be done at higher level

Navarro-Perez, Voronkov

Word level



$$\forall \mathtt{S}, \mathtt{S'}\,(\mathit{Next}(\mathtt{S}, \mathtt{S'}) \rightarrow \qquad \text{// write is enabled}$$
$$\forall y(\mathtt{Assoc}_{\mathtt{wraddr}}(\mathtt{S'}, y) \rightarrow$$
$$\forall \mathtt{A}(\mathtt{clock}(\mathtt{S'}) \wedge \mathtt{wren}(\mathtt{S'}) \wedge \mathtt{A} = y \rightarrow$$
$$\forall \mathtt{B}(\mathtt{range}_{[0,63]}(\mathtt{B}) \rightarrow (\mathtt{mem}(\mathtt{S'}, \mathtt{A}, \mathtt{B}) \leftrightarrow \mathtt{wrdata}(\mathtt{S}, \mathtt{B}))))))).$$

Memories and bit-vectors are represented by first-order predicates: $\mathtt{mem}(\mathtt{S}, \mathtt{A}, \mathtt{B}), \mathtt{wrdata}(\mathtt{S}, \mathtt{B})$.

Emmer, Korovin, Khasidashvili, Sticksel, Voronkov;

relevant complexity resutls: Kovasznai, Frohlich, Biere

Effectively Propositional Fragment (EPR)

EPR: $\exists \bar{y} \forall \bar{x} F(\bar{x}, \bar{y})$, where F is function-free.

Skolemized: $\forall \bar{x} F(c, \bar{x})$

EPR

$$\forall \mathtt{S}, \mathtt{B}(\mathtt{mem}(\mathtt{S}, \mathtt{rdaddr}, \mathtt{B}) \leftrightarrow \mathtt{rddata}(\mathtt{S}, \mathtt{B}))$$

represents propositional:

$$\begin{array}{c}
(\mathtt{mem}(s_1, \mathtt{rdaddr}, i_1) \leftrightarrow \mathtt{rddata}(s_1, i_1)) \\
(\mathtt{mem}(s_1, \mathtt{rdaddr}, i_2) \leftrightarrow \mathtt{rddata}(s_1, i_2)) \\
(\mathtt{mem}(s_2, \mathtt{rdaddr}, i_3) \leftrightarrow \mathtt{rddata}(s_2, i_3)) \\
\ldots
\end{array}$$

Efficient solvers for EPR: iProver, Darwin, Equinox

CASC competition: EPR division

EPR-preserving Skolemization

Most formulas obtained from hardware designs are definitions.

Skolemization of definitions can introduce functions!

$$\mathtt{prop}(\mathtt{S}) \leftrightarrow \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{outp}(\mathtt{S},\mathtt{B}) \leftrightarrow \mathtt{cacheline}(\mathtt{S},\mathtt{B})))$$

EPR-preserving Skolemization

Most formulas obtained from hardware designs are definitions.

Skolemization of definitions can introduce functions!

$$\mathtt{prop}(\mathtt{S}) \leftrightarrows \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{outp}(\mathtt{S},\mathtt{B}) \leftrightarrow \mathtt{cacheline}(\mathtt{S},\mathtt{B})))$$

EPR-preserving Skolemization

Most formulas obtained from hardware designs are definitions.

Skolemization of definitions can introduce functions!

$$\mathtt{prop}(\mathtt{S}) \leftrightarrows \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{outp}(\mathtt{S},\mathtt{B}) \leftrightarrow \mathtt{cacheline}(\mathtt{S},\mathtt{B})))$$

$$\mathtt{prop}(\mathtt{S}) \leftarrow (\mathtt{range}_{[0:63]}(f(\mathtt{S})) \rightarrow (\mathtt{outp}(\mathtt{S},f(\mathtt{S})) \leftrightarrow \mathtt{cacheline}(\mathtt{S},f(\mathtt{S}))))$$

Note: only $\leftarrow$ is problematic.

EPR-preserving Skolemization

Most formulas obtained from hardware designs are definitions.

Skolemization of definitions can introduce functions!

$$\mathtt{prop}(\mathtt{S}) \leftrightarrows \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{outp}(\mathtt{S},\mathtt{B}) \leftrightarrow \mathtt{cacheline}(\mathtt{S},\mathtt{B})))$$

$$\mathtt{prop}(\mathtt{S}) \leftarrow (\mathtt{range}_{[0:63]}(f(\mathtt{S})) \rightarrow (\mathtt{outp}(\mathtt{S},f(\mathtt{S})) \leftrightarrow \mathtt{cacheline}(\mathtt{S},f(\mathtt{S}))))$$

Note: only $\leftarrow$ is problematic.

Remove: $\leftarrow$ if all occurrences of `prop` are positive

EPR-preserving Skolemization

Most formulas obtained from hardware designs are definitions.

Skolemization of definitions can introduce functions!

$$\mathtt{prop}(\mathtt{S}) \leftrightarrows \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{outp}(\mathtt{S},\mathtt{B}) \leftrightarrow \mathtt{cacheline}(\mathtt{S},\mathtt{B})))$$

$$\mathtt{prop}(\mathtt{S}) \leftarrow (\mathtt{range}_{[0:63]}(f(\mathtt{S})) \rightarrow (\mathtt{outp}(\mathtt{S},f(\mathtt{S})) \leftrightarrow \mathtt{cacheline}(\mathtt{S},f(\mathtt{S}))))$$

Note: only $\leftarrow$ is problematic.

Remove: $\leftarrow$ if all occurrences of `prop` are positive

Inline all negative occurrences of `prop`:

$$\neg\mathtt{prop}(s_k)$$

EPR-preserving Skolemization

Most formulas obtained from hardware designs are definitions.

Skolemization of definitions can introduce functions!

$$\mathtt{prop}(\mathtt{S}) \leftrightarrows \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{outp}(\mathtt{S},\mathtt{B}) \leftrightarrow \mathtt{cacheline}(\mathtt{S},\mathtt{B})))$$
$$\mathtt{prop}(\mathtt{S}) \leftarrow (\mathtt{range}_{[0:63]}(f(\mathtt{S})) \rightarrow (\mathtt{outp}(\mathtt{S},f(\mathtt{S})) \leftrightarrow \mathtt{cacheline}(\mathtt{S},f(\mathtt{S})))$$

Note: only $\leftarrow$ is problematic.

Remove: $\leftarrow$ if all occurrences of `prop` are positive

Inline all negative occurrences of `prop`:

$$\neg\mathtt{prop}(s_k)$$
$$\exists \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \wedge \neg(\mathtt{outp}(s_K,\mathtt{B}) \leftrightarrow \mathtt{cacheline}(s_k,\mathtt{B})))$$

EPR-preserving Skolemization

Most formulas obtained from hardware designs are definitions.

Skolemization of definitions can introduce functions!

$$\mathtt{prop}(\mathtt{S}) \leftrightarrows \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{outp}(\mathtt{S},\mathtt{B}) \leftrightarrow \mathtt{cacheline}(\mathtt{S},\mathtt{B})))$$
$$\mathtt{prop}(\mathtt{S}) \leftarrow (\mathtt{range}_{[0:63]}(f(\mathtt{S})) \rightarrow (\mathtt{outp}(\mathtt{S},f(\mathtt{S})) \leftrightarrow \mathtt{cacheline}(\mathtt{S},f(\mathtt{S})))$$

Note: only $\leftarrow$ is problematic.

Remove: $\leftarrow$ if all occurrences of `prop` are positive

Inline all negative occurrences of `prop`:

$$\neg\mathtt{prop}(s_k)$$
$$\exists \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \wedge \neg(\mathtt{outp}(s_K,\mathtt{B}) \leftrightarrow \mathtt{cacheline}(s_k,\mathtt{B})))$$
$$(\mathtt{range}_{[0:63]}(\mathrm{sk}) \wedge \neg(\mathtt{outp}(s_K,\mathrm{sk}) \leftrightarrow \mathtt{cacheline}(s_k,\mathrm{sk})))$$

After inlining:
1) obtain an EPR formula and
2) we can remove $\leftarrow$ from the definition.

Conditional Inlining

$$\forall \mathtt{S}(\mathtt{reach}(\mathtt{S}) \rightarrow \\ \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow \\ (\mathtt{bv}_1(\mathtt{S},\mathtt{B}) \leftrightarrow \mathtt{mem}_1(\mathtt{S},\mathtt{wraddr},\mathtt{B})))).$$

$$\forall \mathtt{S}(\mathtt{reach}(\mathtt{S}) \rightarrow \\ \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow \\ (\mathtt{bv}_1(\mathtt{S},\mathtt{B}) \vee (\neg\mathtt{mem}_2(\mathtt{S},\mathtt{rdaddr},\mathtt{B}) \wedge \mathtt{bv}_2(\mathtt{S},\mathtt{B}))))).$$

Conditional Inlining

$$\forall \mathtt{S}(\mathtt{reach}(\mathtt{S}) \rightarrow \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{bv}_1(\mathtt{S},\mathtt{B}) \leftrightarrow \mathtt{mem}_1(\mathtt{S},\mathtt{wraddr},\mathtt{B})))).$$

$$\forall \mathtt{S}(\mathtt{reach}(\mathtt{S}) \rightarrow \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{bv}_1(\mathtt{S},\mathtt{B}) \vee (\neg\mathtt{mem}_2(\mathtt{S},\mathtt{rdaddr},\mathtt{B}) \wedge \mathtt{bv}_2(\mathtt{S},\mathtt{B}))))).$$

$$\forall \mathtt{S}(\mathtt{reach}(\mathtt{S}) \rightarrow \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) (\mathtt{mem}_1(\mathtt{S},\mathtt{wraddr},\mathtt{B}) \vee (\neg\mathtt{mem}_2(\mathtt{S},\mathtt{rdaddr},\mathtt{B}) \wedge \mathtt{bv}_2(\mathtt{S},\mathtt{B}))))).$$

Conditional Inlining

$$\forall \mathtt{S}(\mathtt{reach}(\mathtt{S}) \rightarrow \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{bv}_1(\mathtt{S},\mathtt{B}) \leftrightarrow \mathtt{mem}_1(\mathtt{S},\mathtt{wraddr},\mathtt{B})))).$$

$$\forall \mathtt{S}(\mathtt{reach}(\mathtt{S}) \rightarrow \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{bv}_1(\mathtt{S},\mathtt{B}) \vee (\neg\mathtt{mem}_2(\mathtt{S},\mathtt{rdaddr},\mathtt{B}) \wedge \mathtt{bv}_2(\mathtt{S},\mathtt{B}))))).$$

$$\forall \mathtt{S}(\mathtt{reach}(\mathtt{S}) \rightarrow \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) (\mathtt{mem}_1(\mathtt{S},\mathtt{wraddr},\mathtt{B}) \vee (\neg\mathtt{mem}_2(\mathtt{S},\mathtt{rdaddr},\mathtt{B}) \wedge \mathtt{bv}_2(\mathtt{S},\mathtt{B}))))).$$

Conditional Inlining

$$\forall \mathtt{S}(\mathtt{reach}(\mathtt{S}) \rightarrow \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{bv}_1(\mathtt{S},\mathtt{B}) \leftrightarrow \mathtt{mem}_1(\mathtt{S},\mathtt{wraddr},\mathtt{B})))).$$

×

$$\forall \mathtt{S}(\mathtt{reach}(\mathtt{S}) \rightarrow \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{bv}_1(\mathtt{S},\mathtt{B}) \vee (\neg\mathtt{mem}_2(\mathtt{S},\mathtt{rdaddr},\mathtt{B}) \wedge \mathtt{bv}_2(\mathtt{S},\mathtt{B}))))).$$

×

$$\forall \mathtt{S}(\mathtt{reach}(\mathtt{S}) \rightarrow \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) (\mathtt{mem}_1(\mathtt{S},\mathtt{wraddr},\mathtt{B}) \vee (\neg\mathtt{mem}_2(\mathtt{S},\mathtt{rdaddr},\mathtt{B}) \wedge \mathtt{bv}_2(\mathtt{S},\mathtt{B}))))).$$

Conditional Inlining

$$\forall \mathtt{S}(\mathtt{reach}(\mathtt{S}) \rightarrow \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{bv}_1(\mathtt{S},\mathtt{B}) \leftrightarrow \mathtt{mem}_1(\mathtt{S},\mathtt{wraddr},\mathtt{B})))).$$

×

$$\forall \mathtt{S}(\mathtt{reach}(\mathtt{S}) \rightarrow \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) \rightarrow (\mathtt{bv}_1(\mathtt{S},\mathtt{B}) \vee (\neg\mathtt{mem}_2(\mathtt{S},\mathtt{rdaddr},\mathtt{B}) \wedge \mathtt{bv}_2(\mathtt{S},\mathtt{B}))))).$$

×

$$\forall \mathtt{S}(\mathtt{reach}(\mathtt{S}) \rightarrow \forall \mathtt{B}(\mathtt{range}_{[0:63]}(\mathtt{B}) (\mathtt{mem}_1(\mathtt{S},\mathtt{wraddr},\mathtt{B}) \vee (\neg\mathtt{mem}_2(\mathtt{S},\mathtt{rdaddr},\mathtt{B}) \wedge \mathtt{bv}_2(\mathtt{S},\mathtt{B}))))).$$

(Conditional) inlining:

- EPR - preserving Skolemization
- definition elimination
- non-growing inlining

Equivalence discovery with SAT sweeping

Given a first-order formula F which literals in F are equivalent?

SAT sweeping:

- clausify F
- abstract first-order literals to propositional variables
- use SAT-based techniques for equivalence checking

First-order F:

$$\begin{array}{rl} p(x) \vee \neg q(x,c) \vee \neg r(x) & \wedge \\ q(x,c) \vee r(x) & \wedge \\ q(x,c) \vee \neg r(x) & \wedge \\ \neg p(x) \vee \neg q(x,c) \vee r(x) & \end{array}$$

Propositional abstraction S:

$$\begin{array}{rl} v_{p(x)} \vee \neg v_{q(x,c)} \vee \neg v_{r(x)} & \wedge \\ v_{q(x,c)} \vee v_{r(x)} & \wedge \\ v_{q(x,c)} \vee \neg v_{r(x)} & \wedge \\ \neg v_{p(x)} \vee \neg v_{q(x,c)} \vee v_{r(x)} & \end{array}$$

If $S \models v_{p(x)} \leftrightarrow v_{r(x)}$ then $F \models \forall x\ (p(x) \leftrightarrow r(x))$.

If $S \cup \{v_{p(x)}, \neg v_{r(x)}\} \models \bot$ and $S \cup \{\neg v_{p(x)}, v_{r(x)}\} \models \bot$ then $S \models v_{p(x)} \leftrightarrow v_{r(x)}$ and therefore $F \models \forall x\ (p(x) \leftrightarrow r(x))$.

Eliminate equivalent literals

Extension: equivalences of subformulas

(simultaneous prop. equivalence checking: Khasidashvili, Nadel)

Equivalence discovery with SAT sweeping

Given a first-order formula F which literals in F are equivalent?

SAT sweeping:

- clausify F
- abstract first-order literals to propositional variables
- use SAT-based techniques for equivalence checking

First-order F:

$$\begin{array}{rl} p(x) \vee \neg q(x,c) \vee \neg r(x) & \wedge \\ q(x,c) \vee r(x) & \wedge \\ q(x,c) \vee \neg r(x) & \wedge \\ \neg p(x) \vee \neg q(x,c) \vee r(x) & \end{array}$$

Propositional abstraction S:

$$\begin{array}{rl} v_{p(x)} \vee \neg v_{q(x,c)} \vee \neg v_{r(x)} & \wedge \\ v_{q(x,c)} \vee v_{r(x)} & \wedge \\ v_{q(x,c)} \vee \neg v_{r(x)} & \wedge \\ \neg v_{p(x)} \vee \neg v_{q(x,c)} \vee v_{r(x)} & \end{array}$$

If $S \models v_{p(x)} \leftrightarrow v_{r(x)}$ then $F \models \forall x\ (p(x) \leftrightarrow r(x))$.

If $S \cup \{v_{p(x)}, \neg v_{r(x)}\} \models \bot$ and $S \cup \{\neg v_{p(x)}, v_{r(x)}\} \models \bot$ then $S \models v_{p(x)} \leftrightarrow v_{r(x)}$ and therefore $F \models \forall x\ (p(x) \leftrightarrow r(x))$.

Eliminate equivalent literals

Extension: equivalences of subformulas

(simultaneous prop. equivalence checking: Khasidashvili, Nadel)

Equivalence discovery with SAT sweeping

Given a first-order formula F which literals in F are equivalent?

SAT sweeping:

- clausify F
- abstract first-order literals to propositional variables
- use SAT-based techniques for equivalence checking

First-order F:

$$\begin{array}{rl} p(x) \vee \neg q(x,c) \vee \neg r(x) & \wedge \\ q(x,c) \vee r(x) & \wedge \\ q(x,c) \vee \neg r(x) & \wedge \\ \neg p(x) \vee \neg q(x,c) \vee r(x) & \end{array}$$

Propositional abstraction S:

$$\begin{array}{rl} v_{p(x)} \vee \neg v_{q(x,c)} \vee \neg v_{r(x)} & \wedge \\ v_{q(x,c)} \vee v_{r(x)} & \wedge \\ v_{q(x,c)} \vee \neg v_{r(x)} & \wedge \\ \neg v_{p(x)} \vee \neg v_{q(x,c)} \vee v_{r(x)} & \end{array}$$

If $S \models v_{p(x)} \leftrightarrow v_{r(x)}$ then $F \models \forall x\ (p(x) \leftrightarrow r(x))$.

If $S \cup \{v_{p(x)}, \neg v_{r(x)}\} \models \bot$ and $S \cup \{\neg v_{p(x)}, v_{r(x)}\} \models \bot$ then $S \models v_{p(x)} \leftrightarrow v_{r(x)}$ and therefore $F \models \forall x\ (p(x) \leftrightarrow r(x))$.

Eliminate equivalent literals

Extension: equivalences of subformulas

(simultaneous prop. equivalence checking: Khasidashvili, Nadel)

Equivalence discovery with SAT sweeping

Given a first-order formula F which literals in F are equivalent?

SAT sweeping:

- clausify F
- abstract first-order literals to propositional variables
- use SAT-based techniques for equivalence checking

First-order F:

$$\begin{array}{rl} p(x) \vee \neg q(x,c) \vee \neg r(x) & \wedge \\ q(x,c) \vee r(x) & \wedge \\ q(x,c) \vee \neg r(x) & \wedge \\ \neg p(x) \vee \neg q(x,c) \vee r(x) & \end{array}$$

Propositional abstraction S:

$$\begin{array}{rl} v_{p(x)} \vee \neg v_{q(x,c)} \vee \neg v_{r(x)} & \wedge \\ v_{q(x,c)} \vee v_{r(x)} & \wedge \\ v_{q(x,c)} \vee \neg v_{r(x)} & \wedge \\ \neg v_{p(x)} \vee \neg v_{q(x,c)} \vee v_{r(x)} & \end{array}$$

If $S \models v_{p(x)} \leftrightarrow v_{r(x)}$ then $F \models \forall x\ (p(x) \leftrightarrow r(x))$.

If $S \cup \{v_{p(x)}, \neg v_{r(x)}\} \models \bot$ and $S \cup \{\neg v_{p(x)}, v_{r(x)}\} \models \bot$ then $S \models v_{p(x)} \leftrightarrow v_{r(x)}$ and therefore $F \models \forall x\ (p(x) \leftrightarrow r(x))$.

Eliminate equivalent literals

Extension: equivalences of subformulas

(simultaneous prop. equivalence checking: Khasidashvili, Nadel)

Equivalence discovery with SAT sweeping

Given a first-order formula F which literals in F are equivalent?

SAT sweeping:

- clausify F
- abstract first-order literals to propositional variables
- use SAT-based techniques for equivalence checking

First-order F:

$$\begin{array}{rl} p(x) \vee \neg q(x,c) \vee \neg r(x) & \wedge \\ q(x,c) \vee r(x) & \wedge \\ q(x,c) \vee \neg r(x) & \wedge \\ \neg p(x) \vee \neg q(x,c) \vee r(x) & \end{array}$$

Propositional abstraction S:

$$\begin{array}{rl} v_{p(x)} \vee \neg v_{q(x,c)} \vee \neg v_{r(x)} & \wedge \\ v_{q(x,c)} \vee v_{r(x)} & \wedge \\ v_{q(x,c)} \vee \neg v_{r(x)} & \wedge \\ \neg v_{p(x)} \vee \neg v_{q(x,c)} \vee v_{r(x)} & \end{array}$$

If $S \models v_{p(x)} \leftrightarrow v_{r(x)}$ then $F \models \forall x\ (p(x) \leftrightarrow r(x))$.

If $S \cup \{v_{p(x)}, \neg v_{r(x)}\} \models \bot$ and $S \cup \{\neg v_{p(x)}, v_{r(x)}\} \models \bot$ then $S \models v_{p(x)} \leftrightarrow v_{r(x)}$ and therefore $F \models \forall x\ (p(x) \leftrightarrow r(x))$.

Eliminate equivalent literals

Extension: equivalences of subformulas

(simultaneous prop. equivalence checking: Khasidashvili, Nadel)

Equivalence discovery with SAT sweeping

Given a first-order formula F which literals in F are equivalent?

SAT sweeping:

- clausify F
- abstract first-order literals to propositional variables
- use SAT-based techniques for equivalence checking

First-order F:

$$\begin{array}{rl} p(x) \vee \neg q(x,c) \vee \neg r(x) & \wedge \\ q(x,c) \vee r(x) & \wedge \\ q(x,c) \vee \neg r(x) & \wedge \\ \neg p(x) \vee \neg q(x,c) \vee r(x) & \end{array}$$

Propositional abstraction S:

$$\begin{array}{rl} v_{p(x)} \vee \neg v_{q(x,c)} \vee \neg v_{r(x)} & \wedge \\ v_{q(x,c)} \vee v_{r(x)} & \wedge \\ v_{q(x,c)} \vee \neg v_{r(x)} & \wedge \\ \neg v_{p(x)} \vee \neg v_{q(x,c)} \vee v_{r(x)} & \end{array}$$

If $S \models v_{p(x)} \leftrightarrow v_{r(x)}$ then $F \models \forall x\ (p(x) \leftrightarrow r(x))$.

If $S \cup \{v_{p(x)}, \neg v_{r(x)}\} \models \bot$ and $S \cup \{\neg v_{p(x)}, v_{r(x)}\} \models \bot$ then $S \models v_{p(x)} \leftrightarrow v_{r(x)}$ and therefore $F \models \forall x\ (p(x) \leftrightarrow r(x))$.

Eliminate equivalent literals

Extension: equivalences of subformulas

(simultaneous prop. equivalence checking: Khasidashvili, Nadel)

Equivalence discovery with SAT sweeping

Given a first-order formula F which literals in F are equivalent?

SAT sweeping:

- clausify F
- abstract first-order literals to propositional variables
- use SAT-based techniques for equivalence checking

First-order F:

$$\begin{array}{rl} p(x) \vee \neg q(x,c) \vee \neg r(x) & \wedge \\ q(x,c) \vee r(x) & \wedge \\ q(x,c) \vee \neg r(x) & \wedge \\ \neg p(x) \vee \neg q(x,c) \vee r(x) & \end{array}$$

Propositional abstraction S:

$$\begin{array}{rl} v_{p(x)} \vee \neg v_{q(x,c)} \vee \neg v_{r(x)} & \wedge \\ v_{q(x,c)} \vee v_{r(x)} & \wedge \\ v_{q(x,c)} \vee \neg v_{r(x)} & \wedge \\ \neg v_{p(x)} \vee \neg v_{q(x,c)} \vee v_{r(x)} & \end{array}$$

If $S \models v_{p(x)} \leftrightarrow v_{r(x)}$ then $F \models \forall x\ (p(x) \leftrightarrow r(x))$.

If $S \cup \{v_{p(x)}, \neg v_{r(x)}\} \models \bot$ and $S \cup \{\neg v_{p(x)}, v_{r(x)}\} \models \bot$ then $S \models v_{p(x)} \leftrightarrow v_{r(x)}$ and therefore $F \models \forall x\ (p(x) \leftrightarrow r(x))$.

Eliminate equivalent literals

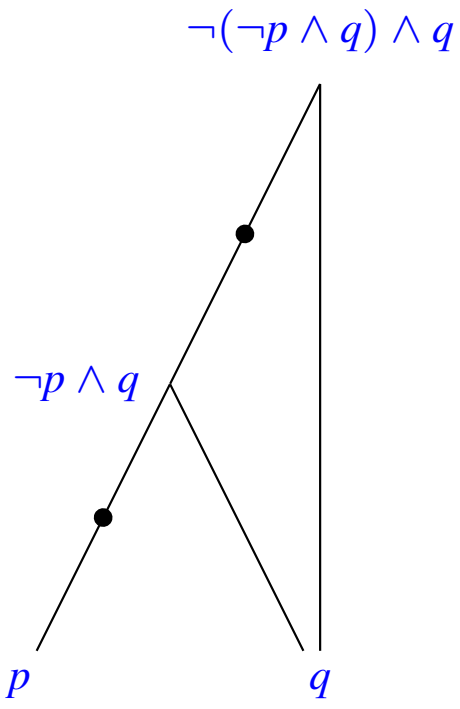
Extension: equivalences of subformulas

(simultaneous prop. equivalence checking: Khasidashvili, Nadel)

And-Inverter Graphs (AIG)

And-Inverter graphs for propositional formula representation.

Only two nodes: $\wedge$ and $\neg$



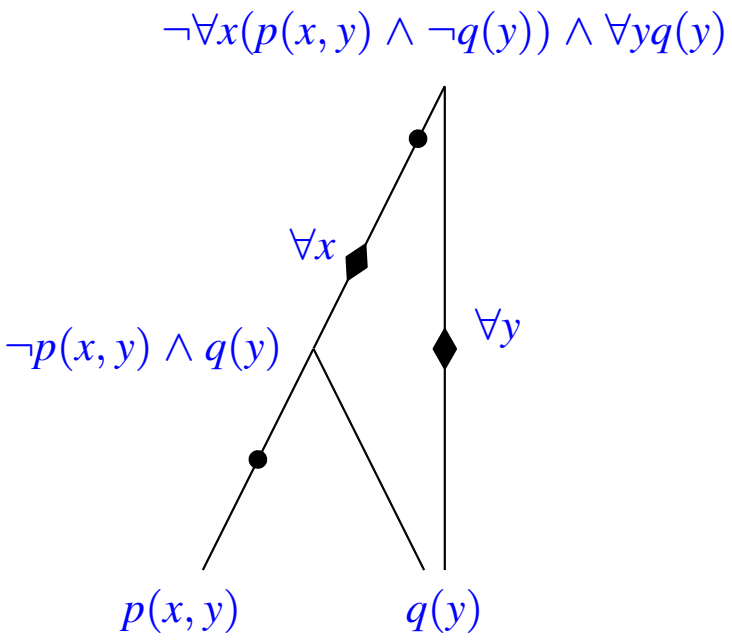
AIGs:

- compact representation
- sharing structurally isomorphic subformulas
- simplifications

Quantified AIGs

Quantified AIGs for first-order formula representation.

Three nodes: $\wedge$, $\neg$ and $\forall$



QAIGs:

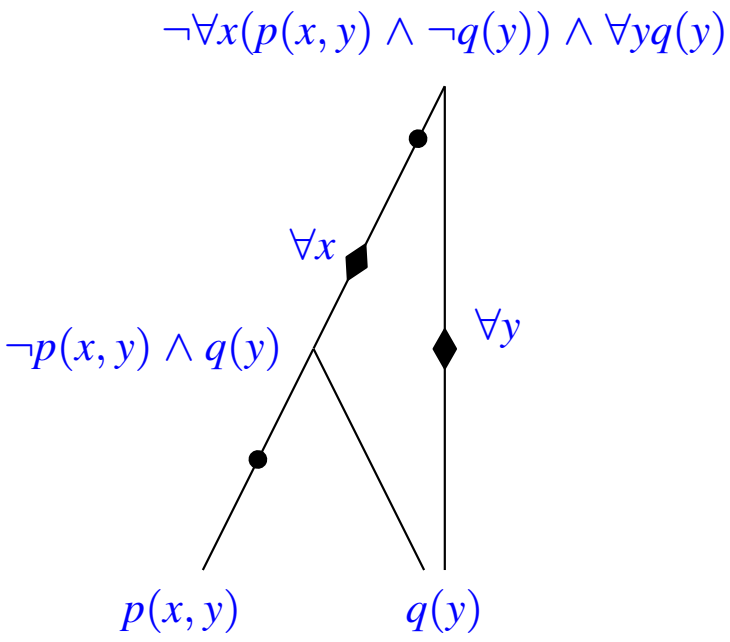
- compact representation of FO formulas
- definition inlining on QAIGs
- variable instantiations

Shortcomings: (Q)AIGs are not canonical even for simple propositional cases.

Quantified AIGs

Quantified AIGs for first-order formula representation.

Three nodes: $\wedge$, $\neg$ and $\forall$



QAIGs:

- compact representation of FO formulas
- definition inlining on QAIGs
- variable instantiations

Shortcomings: (Q)AIGs are not canonical even for simple propositional cases.

Quantified BDDs

Main idea: abstract quantified formulas by propositional variables.

$$\begin{array}{lcl}
\mathrm{atb}(atom(a)) & = & \mathrm{bdd}_{var}(atom(a)) \\
\mathrm{atb}((\forall x, q)) & = & \mathrm{bdd}_{var}((\forall x, \mathrm{atb}(q))) \\
\mathrm{atb}(neg(q)) & = & \mathrm{bdd}_{neg}(\mathrm{atb}(q)) \\
\mathrm{atb}(and(q_1, q_2)) & = & \mathrm{bdd}_{and}(\mathrm{atb}(q_1), \mathrm{atb}(q_2))
\end{array}$$

Example:

$$\forall x[(p(x,y) \vee \neg\exists x q(x)) \wedge (\exists x q(x) \vee p(x,y)) \wedge q(y)] \wedge \neg\forall x(p(x,y) \wedge q(y))$$

Quantified BDDs

Main idea: abstract quantified formulas by propositional variables.

$$\begin{array}{lcl} \mathrm{atb}(atom(a)) & = & \mathrm{bdd}_{var}(atom(a)) \\ \mathrm{atb}((\forall x, q)) & = & \mathrm{bdd}_{var}((\forall x, \mathrm{atb}(q))) \\ \mathrm{atb}(neg(q)) & = & \mathrm{bdd}_{neg}(\mathrm{atb}(q)) \\ \mathrm{atb}(and(q_1, q_2)) & = & \mathrm{bdd}_{and}(\mathrm{atb}(q_1), \mathrm{atb}(q_2)) \end{array}$$

Example:

$$\forall x[(p(x,y) \vee \neg\exists x q(x)) \wedge (\exists x q(x) \vee p(x,y)) \wedge q(y)] \wedge \neg\forall x(p(x,y) \wedge q(y))$$

level 0: $v_{p(x,y)}, v_{q(x)}, v_{q(y)}$

Quantified BDDs

Main idea: abstract quantified formulas by propositional variables.

$$\begin{array}{lcl} \mathrm{atb}(atom(a)) & = & \mathrm{bdd}_{var}(atom(a)) \\ \mathrm{atb}((\forall x, q)) & = & \mathrm{bdd}_{var}((\forall x, \mathrm{atb}(q))) \\ \mathrm{atb}(neg(q)) & = & \mathrm{bdd}_{neg}(\mathrm{atb}(q)) \\ \mathrm{atb}(and(q_1, q_2)) & = & \mathrm{bdd}_{and}(\mathrm{atb}(q_1), \mathrm{atb}(q_2)) \end{array}$$

Example:

$$\forall x[(p(x,y) \vee \neg\exists x q(x)) \wedge (\exists x q(x) \vee p(x,y)) \wedge q(y)] \wedge \neg\forall x(p(x,y) \wedge q(y))$$

level 0: $v_{p(x,y)}, v_{q(x)}, v_{q(y)}$

level 1: $v_{\exists x\langle q(x)\rangle}, v_{\forall x\langle p(x,y)\wedge q(y)\rangle}$

Quantified BDDs

Main idea: abstract quantified formulas by propositional variables.

$$\begin{array}{lcl} \mathrm{atb}(atom(a)) & = & \mathrm{bdd}_{var}(atom(a)) \\ \mathrm{atb}((\forall x, q)) & = & \mathrm{bdd}_{var}((\forall x, \mathrm{atb}(q))) \\ \mathrm{atb}(neg(q)) & = & \mathrm{bdd}_{neg}(\mathrm{atb}(q)) \\ \mathrm{atb}(and(q_1, q_2)) & = & \mathrm{bdd}_{and}(\mathrm{atb}(q_1), \mathrm{atb}(q_2)) \end{array}$$

Example:

$$\forall x[(p(x,y) \vee \neg\exists x q(x)) \wedge (\exists x q(x) \vee p(x,y)) \wedge q(y)] \wedge \neg\forall x(p(x,y) \wedge q(y))$$

level 0: $v_{p(x,y)}, v_{q(x)}, v_{q(y)}$
level 1: $v_{\exists x\langle q(x)\rangle}, v_{\forall x\langle p(x,y)\wedge q(y)\rangle}$

QBDD reduction

Quantified BDDs

Main idea: abstract quantified formulas by propositional variables.

$$\begin{array}{lcl} \mathrm{atb}(atom(a)) & = & \mathrm{bdd}_{var}(atom(a)) \\ \mathrm{atb}((\forall x, q)) & = & \mathrm{bdd}_{var}((\forall x, \mathrm{atb}(q))) \\ \mathrm{atb}(neg(q)) & = & \mathrm{bdd}_{neg}(\mathrm{atb}(q)) \\ \mathrm{atb}(and(q_1, q_2)) & = & \mathrm{bdd}_{and}(\mathrm{atb}(q_1), \mathrm{atb}(q_2)) \end{array}$$

Example:

$$\forall x(p(x,y) \wedge q(y)) \wedge \neg\forall x(p(x,y) \wedge q(y))$$

level 0: $v_{p(x,y)}, v_{q(x)}, v_{q(y)}$
level 1: $v_{\exists x\langle q(x)\rangle}, v_{\forall x\langle p(x,y)\wedge q(y)\rangle}$

QBDD reduction

Quantified BDDs

Main idea: abstract quantified formulas by propositional variables.

$$\begin{array}{lcl} \mathrm{atb}(atom(a)) & = & \mathrm{bdd}_{var}(atom(a)) \\ \mathrm{atb}((\forall x, q)) & = & \mathrm{bdd}_{var}((\forall x, \mathrm{atb}(q))) \\ \mathrm{atb}(neg(q)) & = & \mathrm{bdd}_{neg}(\mathrm{atb}(q)) \\ \mathrm{atb}(and(q_1, q_2)) & = & \mathrm{bdd}_{and}(\mathrm{atb}(q_1), \mathrm{atb}(q_2)) \end{array}$$

Example:

$$\forall x(p(x,y) \wedge q(y)) \wedge \neg\forall x(p(x,y) \wedge q(y))$$

level 0: $v_{p(x,y)}, v_{q(x)}, v_{q(y)}$

level 1: $v_{\exists x\langle q(x)\rangle}, v_{\forall x\langle p(x,y)\wedge q(y)\rangle}$

QBDD reduction

Quantified BDDs

Main idea: abstract quantified formulas by propositional variables.

$$
\begin{array}{lcl}
\mathrm{atb}(atom(a)) & = & \mathrm{bdd}_{var}(atom(a)) \\
\mathrm{atb}((\forall x, q)) & = & \mathrm{bdd}_{var}((\forall x, \mathrm{atb}(q))) \\
\mathrm{atb}(neg(q)) & = & \mathrm{bdd}_{neg}(\mathrm{atb}(q)) \\
\mathrm{atb}(and(q_1, q_2)) & = & \mathrm{bdd}_{and}(\mathrm{atb}(q_1), \mathrm{atb}(q_2))
\end{array}
$$

Example:

$$\bot$$

level 0: $v_{p(x,y)}, v_{q(x)}, v_{q(y)}$

level 1: $v_{\exists x\langle q(x)\rangle}, v_{\forall x\langle p(x,y)\wedge q(y)\rangle}$

QBDD reduction

Quantified BDDs

Main idea: abstract quantified formulas by propositional variables.

$$\begin{array}{lcl} \mathrm{atb}(atom(a)) & = & \mathrm{bdd}_{var}(atom(a)) \\ \mathrm{atb}((\forall x, q)) & = & \mathrm{bdd}_{var}((\forall x, \mathrm{atb}(q))) \\ \mathrm{atb}(neg(q)) & = & \mathrm{bdd}_{neg}(\mathrm{atb}(q)) \\ \mathrm{atb}(and(q_1, q_2)) & = & \mathrm{bdd}_{and}(\mathrm{atb}(q_1), \mathrm{atb}(q_2)) \end{array}$$

Example:

$$\forall x[(p(x,y) \vee \neg\exists x q(x)) \wedge (\exists x q(x) \vee p(x,y)) \wedge q(y)] \wedge \neg\forall x(p(x,y) \wedge q(y))$$

level 0: $v_{p(x,y)}, v_{q(x)}, v_{q(y)}$
level 1: $v_{\exists x\langle q(x)\rangle}, v_{\forall x\langle p(x,y)\wedge q(y)\rangle}$

QBDD reduction

Applying QBDD simplifies this formula to $\perp$.

Quantified BDDs

Main idea: abstract quantified formulas by propositional variables.

$$\begin{array}{lcl} \mathrm{atb}(atom(a)) & = & \mathrm{bdd}_{var}(atom(a)) \\ \mathrm{atb}((\forall x, q)) & = & \mathrm{bdd}_{var}((\forall x, \mathrm{atb}(q))) \\ \mathrm{atb}(neg(q)) & = & \mathrm{bdd}_{neg}(\mathrm{atb}(q)) \\ \mathrm{atb}(and(q_1, q_2)) & = & \mathrm{bdd}_{and}(\mathrm{atb}(q_1), \mathrm{atb}(q_2)) \end{array}$$

Example:

$$\forall x[(p(x,y) \vee \neg\exists x q(x)) \wedge (\exists x q(x) \vee p(x,y)) \wedge q(y)] \wedge \neg\forall x(p(x,y) \wedge q(y))$$

level 0: $v_{p(x,y)}, v_{q(x)}, v_{q(y)}$
level 1: $v_{\exists x\langle q(x)\rangle}, v_{\forall x\langle p(x,y)\wedge q(y)\rangle}$

QBDD reduction

Applying QBDD simplifies this formula to $\perp$.

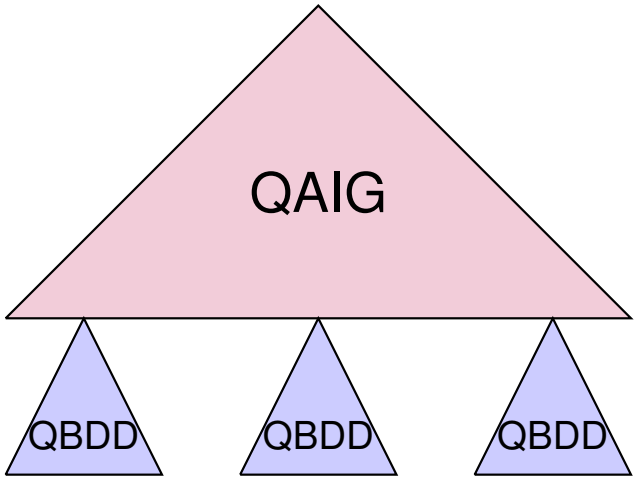
QBDDs: Canonical wrt. Boolean structure.

Shortcomings: BDDs can be exponential in size.

QAIGs+QBDDs

Combination QAIGs+QBDDs.

For each node in QAIG we try to build QBDD.
If QBDD is larger than a threshold then replace QBDD with QAIG.

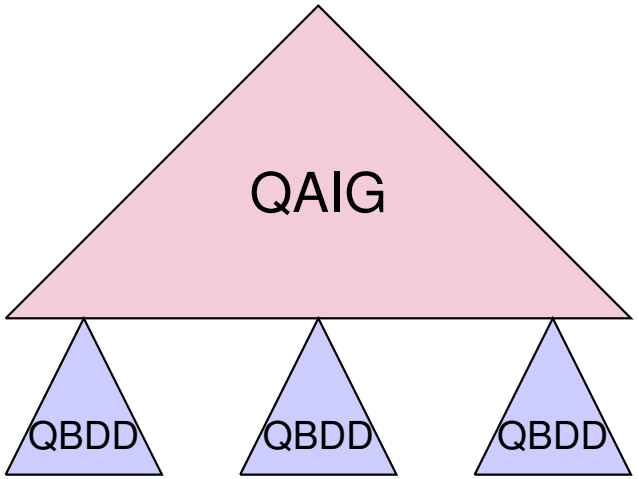


(related in prop. case: Andersen, Hulgaard; Bjesse, Boralv; Kuehlmann, Krohm)

QAIGs+QBDDs

Combination QAIGs+QBDDs.

For each node in QAIG we try to build QBDD.
If QBDD is larger than a threshold then replace QBDD with QAIG.



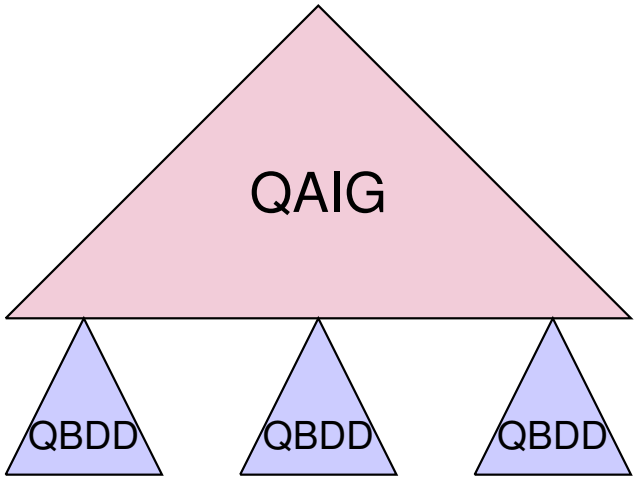
Combined structure: balance between size and Boolean canonicity.

(related in prop. case: Andersen, Hulgaard; Bjesse, Boralv; Kuehlmann, Krohm)

QAIGs+QBDDs

Combination QAIGs+QBDDs.

For each node in QAIG we try to build QBDD.
If QBDD is larger than a threshold then replace QBDD with QAIG.



Combined structure: balance between size and Boolean canonicity.

QAIGs+QBDDs:

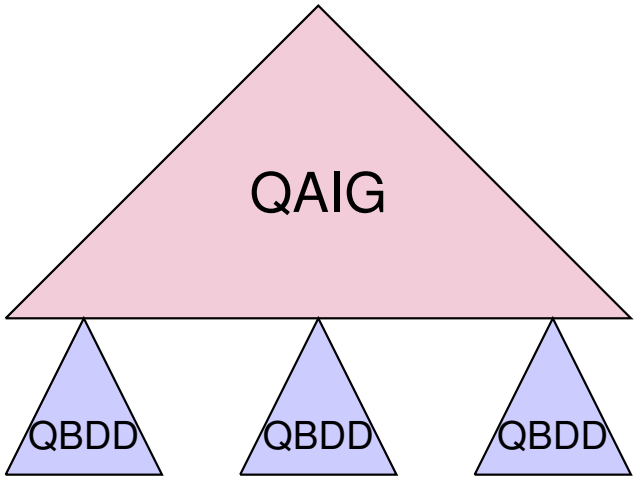
- subformula simplification
- discover new equivalences
- definition elimination

(related in prop. case: Andersen, Hulgaard; Bjesse, Boralv; Kuehlmann, Krohm)

QAIGs+QBDDs

Combination QAIGs+QBDDs.

For each node in QAIG we try to build QBDD.
If QBDD is larger than a threshold then replace QBDD with QAIG.



Combined structure: balance between size and Boolean canonicity.

QAIGs+QBDDs:

- subformula simplification
- discover new equivalences
- definition elimination

(related in prop. case: Andersen, Hulgaard; Bjesse, Boralv; Kuehlmann, Krohm)

Optimization: When
$| \text{AIG} | < | QBDD |$

- keep QBDD for canonicity;
- use AIG as the normal form.

Evaluation (EPR-BMC)

Intel industrial hardware benchmarks:

Design block	FOF size	Bound		CNF size	
		Base	Prep	Base	Prep
BPB2	913	7	9	1977	2921
DCC1	1093	4	4	3209	1615
DCC2	431	7	10	861	370
DCI1	4678	0	1	15899	9852
PMS1	574	5	7	1295	1016
ROB2	713	5	7	1717	1157
SCD1	736	8	9	1908	1328
SCD2	267	8	15	755	524
TOTAL	9404	44	62	27621	18783

Table : BMC1 results on industrial benchmarks.

iProver was used for the EPR solving.

Evaluation TPTP/SMT

ACR – AIGs conditional rewriting
ACR(ERI) – AIGs conditional rewriting restricted to EPR-restoring

		Base+ACR	Prep	Prep+ACR	Prep+ACR(ERI)
$\geq 2x$	faster	100	10	74	122
$> 1x$	faster	4890	1527	4847	4941
$\geq 2x$	slower	36	33	36	36

Table : QA_UF SMT problems

Z3 was used for SMT solving.

	Prep only	Base only
iProver	482	83
Vampire	276	76

Table : TPTP first-order problems

Evaluation TPTP/SMT

ACR – AIGs conditional rewriting
ACR(ERI) – AIGs conditional rewriting restricted to EPR-restoring

		Base+ACR	Prep	Prep+ACR	Prep+ACR(ERI)
$\geq 2x$	faster	100	10	74	122
$> 1x$	faster	4890	1527	4847	4941
$\geq 2x$	slower	36	33	36	36

Table : QA_UF SMT problems

Z3 was used for SMT solving.

	Prep only	Base only
iProver	482	83
Vampire	276	76

Table : TPTP first-order problems

Summary

First-order preprocessing techniques:

- Definition inlining
- Equivalence discovery using SAT sweeping
- Quantified AIGs
- Quantified BDDs
- QAIGs + QBDDs