Dual Proof Generation for Quantified Boolean Formulas with a BDD-Based Solver

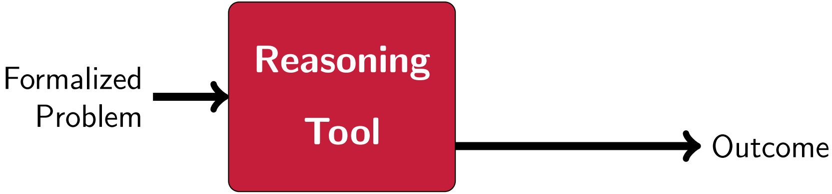
Randal E. Bryant and Marijn J. H. Heule

Carnegie Mellon University

CADE, 2021

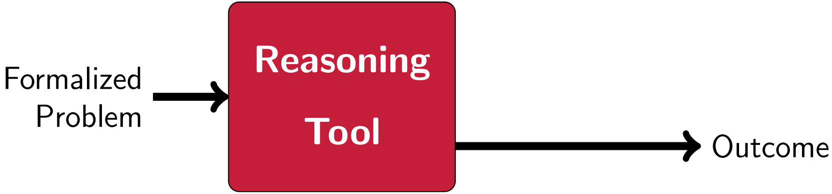
http://www.cs.cmu.edu/~bryant

Automated Reasoning Programs



Are The Results Trustworthy?

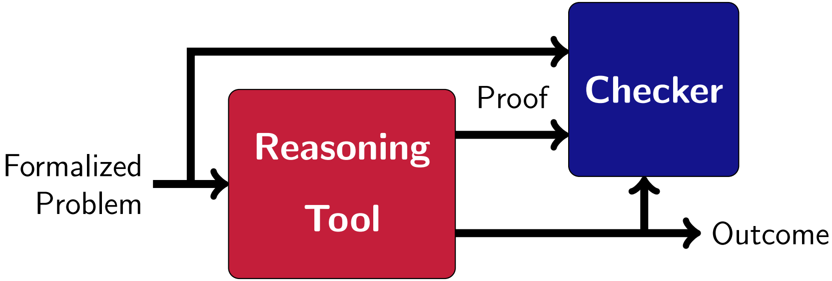
Automated Reasoning Programs



Are The Results Trustworthy?

- *No!*
- Complex software with many optimizations

Proof-Generating Automated Reasoning Programs

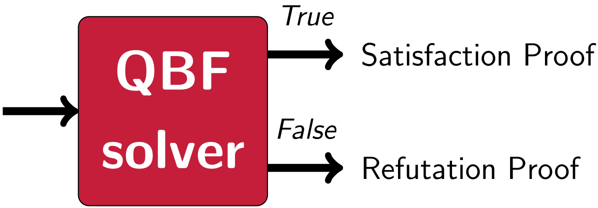


Checkable Proofs

- Step-by-step proof in some logical framework
- Independently validated by proof checker
- Checker should operate in low-degree polynomial time
  - Relative to tool

Proof-Generating QBF Solver

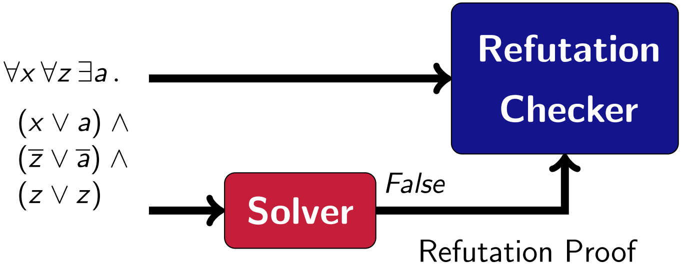
$$\exists a \forall x \exists b\,.$$
$$(a \lor b) \land$$
$$(\overline{x} \lor \overline{b}) \land$$
$$(a \lor x)$$



Quantified Formulas

- Assume fully quantified
  - no free variables
- No satisfying assignment
  - Formula is either true or false

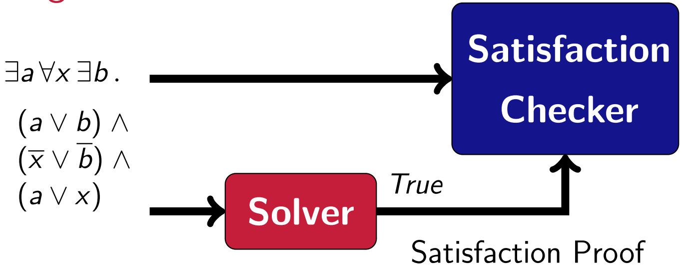
Handling False Formulas



Refutation Proof

- Similar to proofs of unsatisfiability by SAT checker
- Steps leading to empty clause
  - Add new clauses by resolution
  - Eliminate universal variables
- Implemented by *some*, but not all QBF solvers

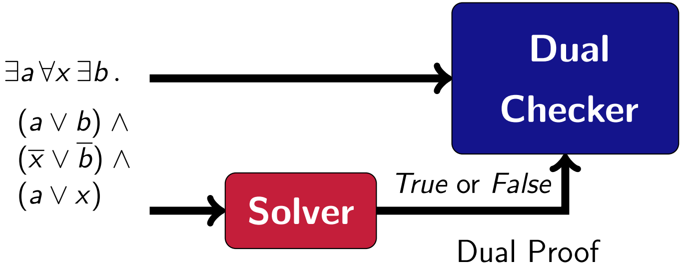
Handling True Formulas



Satisfaction Proof

- No approach in widespread use
- Some do not give low-degree polynomial checker
  - Prove negated formula false
  - Show that replacing existential variables by Skolem functions yields tautology
- Cube resolution
  - Separate proof system based on DNF
  - Derive empty cube, representing tautology

What We *Really* Want



Dual Proof

- Unified framework for satisfaction and refutation proofs
- QBF solver can generate proof as it operates
  - Before it determines whether formula is true or false
- Single checker can handle both cases
  - Single logical framework

QRAT Proof System

- Heule, Seidl, Biere 2014

Clausal Proof System

- Developed to check correctness of QBF preprocessors
- Start with input clauses
- Proof rules to add and delete clauses

Refutation Proof

- Add clauses until generate empty clause
- Logical contradiction

Satisfaction Proof

- Add and delete clauses until have empty set of clauses
- Logical tautology

QRAT Logical Basis

Proof Structure

- Input formula Φ_I
- Each clause addition or deletion step yields modified QBF

$$\Phi_I = \Phi_1, \Phi_2, \ldots, \Phi_t$$

Refutation Proof

- Each step must be *truth preserving*: $\Phi_i \rightarrow \Phi_{i+1}$

$$\Phi_I = \Phi_1 \rightarrow \Phi_2 \rightarrow \cdots \rightarrow \Phi_t = \bot$$

Satisfaction Proof

- Each step must be *falsehood preserving*: $\Phi_i \leftarrow \Phi_{i+1}$

$$\Phi_I = \Phi_1 \leftarrow \Phi_2 \leftarrow \cdots \leftarrow \Phi_t = \top$$

Dual Proof

- Each step must be *equivalence preserving*: $\Phi_i \leftrightarrow \Phi_{i+1}$

$$\Phi_I = \Phi_1 \leftrightarrow \Phi_2 \leftrightarrow \cdots \leftrightarrow \Phi_t \in \{\bot, \top\}$$

Symbolic Solving

$$\exists a \, \forall x \, \exists b \left[\begin{array}{l} (a \vee b) \wedge \\ (\overline{x} \vee \overline{b}) \wedge \\ (a \vee x) \end{array} \right]$$

$a \vee b \qquad \overline{x} \vee \overline{b} \qquad a \vee x$

Operations on Terms

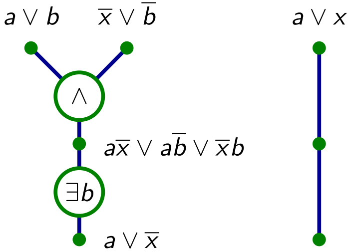
- Conjunction
- Quantification

Quantifiers

- Process from inner to outer
- Existential
  - Must have single term containing variable
- Universal
  - Can process terms separately

Symbolic Solving

$$\exists a \, \forall x \, \exists b \left[\begin{array}{l} (a \vee b) \wedge \\ (\overline{x} \vee \overline{b}) \wedge \\ (a \vee x) \end{array} \right]$$



Operations on Terms

- Conjunction
- Quantification

Quantifiers

- Process from inner to outer
- Existential
  - Must have single term containing variable
- Universal
  - Can process terms separately

Symbolic Solving

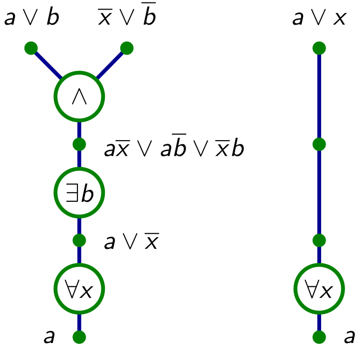
$$\exists a\, \forall x\, \exists b \left[\begin{array}{l} (a \lor b) \land \\ (\overline{x} \lor \overline{b}) \land \\ (a \lor x) \end{array} \right]$$

Operations on Terms

- Conjunction
- Quantification

Quantifiers

- Process from inner to outer
- Existential
  - Must have single term containing variable
- Universal
  - Can process terms separately



Symbolic Solving

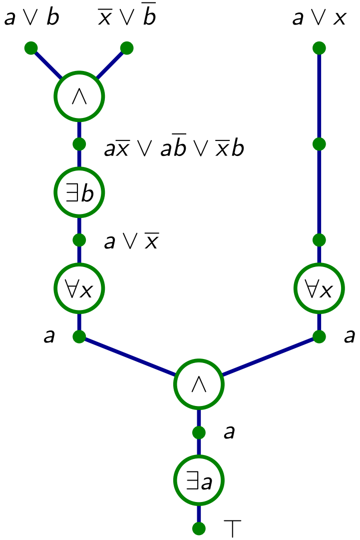
$$\exists a \, \forall x \, \exists b \left[\begin{array}{l} (a \vee b) \wedge \\ (\overline{x} \vee \overline{b}) \wedge \\ (a \vee x) \end{array} \right]$$

Operations on Terms

- Conjunction
- Quantification

Quantifiers

- Process from inner to outer
- Existential
  - Must have single term containing variable
- Universal
  - Can process terms separately



Proof Requirement

$$\exists a\, \forall x\, \exists b\, [(a \vee b) \wedge (\overline{x} \vee \overline{b}) \wedge (a \vee x)]$$

$$\Updownarrow$$

$$\exists a\, \forall x\, \exists b\, [(a\overline{x} \vee a\overline{b} \vee \overline{x}b) \wedge (a \vee x)]$$

$$\Updownarrow$$

$$\exists a\, \forall x\, [(a \vee \overline{x}) \wedge (a \vee x)]$$

$$\Updownarrow$$

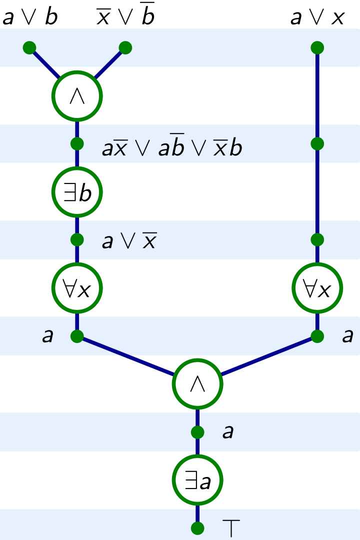
$$\exists a\, [a \wedge a]$$

$$\Updownarrow$$

$$\exists a\, [a]$$

$$\Updownarrow$$

$$\top$$



Reduced Ordered Binary Decision Diagrams (BDDs)

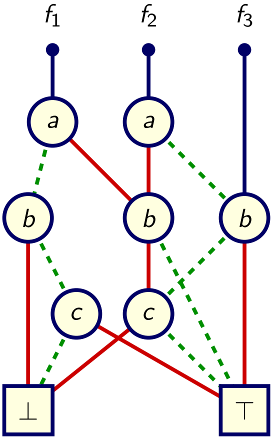
- Bryant 1986

Representation

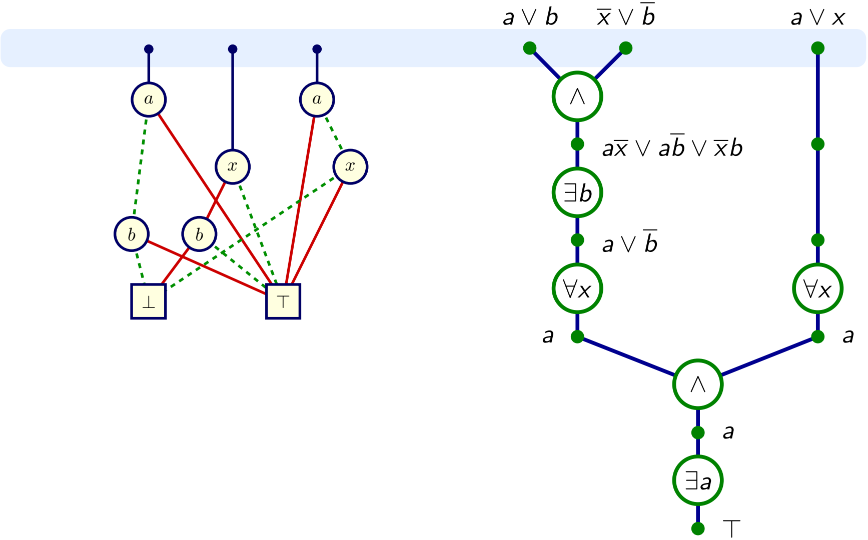
- Canonical representation of set of Boolean functions
- Compact for many useful cases

Algorithms

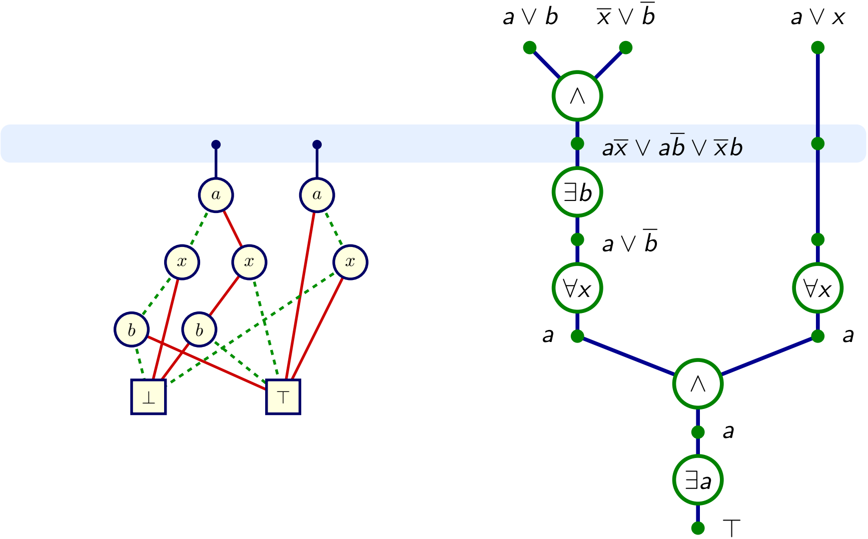
- Apply(f, g, op)
  - Boolean operation op
    - e.g., $\wedge$, $\vee$
  - Generates BDD representation of $f \; op \; g$
- Restrict(f, x, c)
  - $c \in \{0, 1\}$
  - BDD representation of $f|_{x=c}$
  - Used to implement quantification



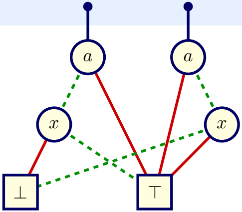
Solving with BDDs

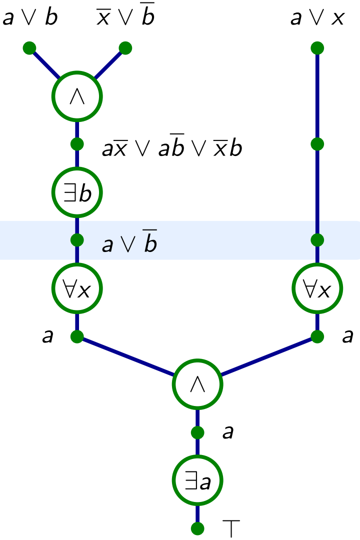


Solving with BDDs

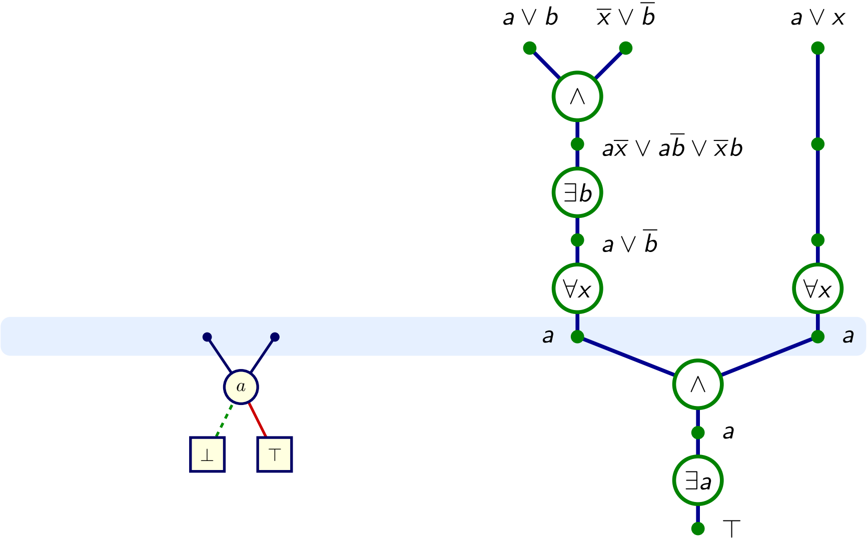


Solving with BDDs

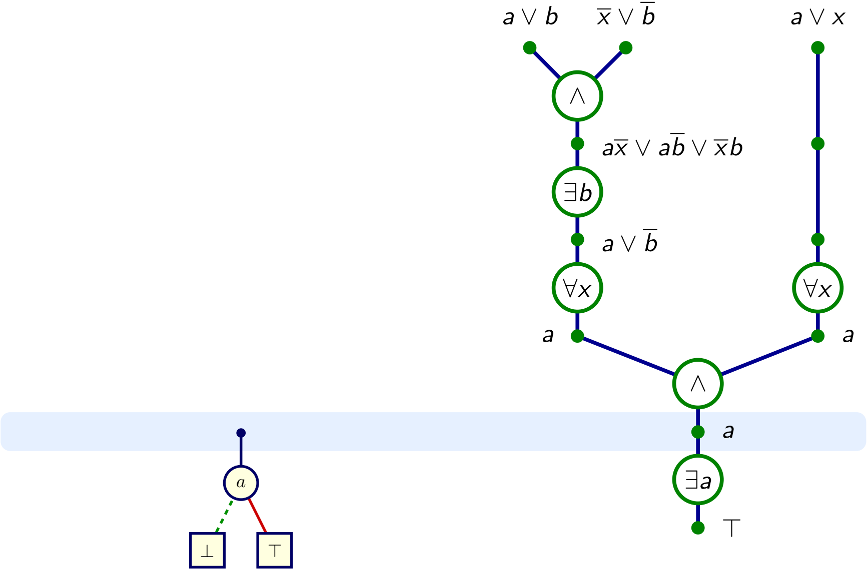




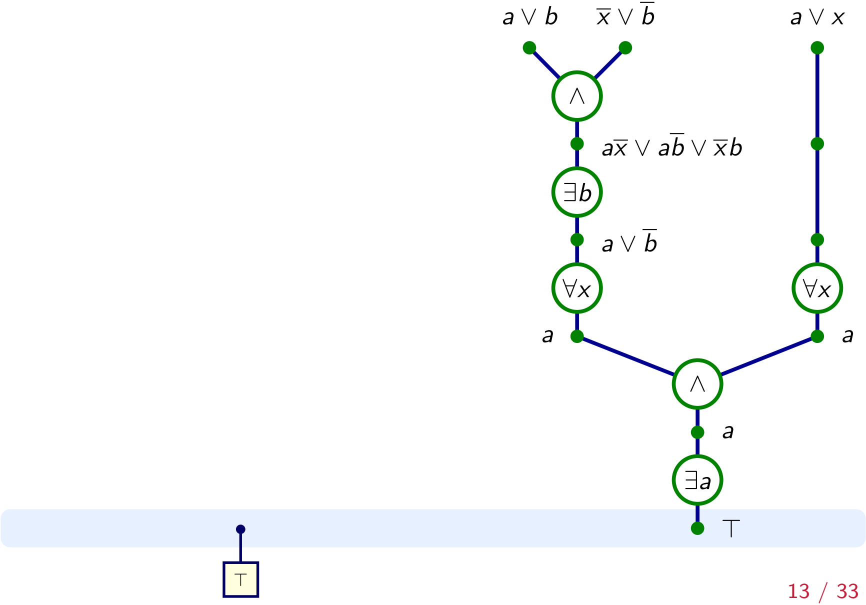
Solving with BDDs



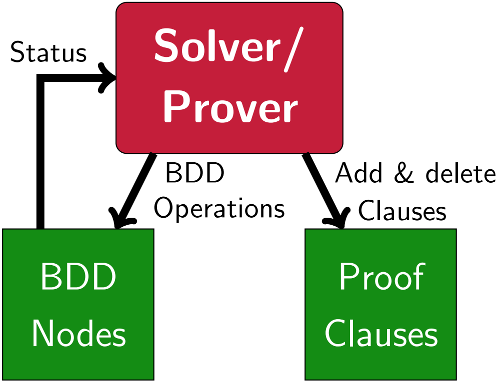
Solving with BDDs



Solving with BDDs



Solver/Prover Operation



- BDD-Based Solver
- Clause operations only to support proof generation

Proof Rules

- Clause addition rules that are truth preserving
- Clause deletion rules that are falsehood preserving
- All covered by QRAT proof system

Resolution:

- Add or remove clauses that are implied by other clauses

Extension:

- Introduce new variable and set of clauses
- Abbreviation for Boolean formula over existing variables

Universal Reduction:

- Remove redundant universal variables from clauses

Existential Elimination:

- Remove all clauses containing some existential variable
- Similar to Davis-Putnam reduction

Extended Resolution for QBF

- Jussila, Sinz, Biere, Kröning, Wintersteiger 2007

Add extension variable e **encoding** $e \leftrightarrow a \wedge x$

- θ is set of *defining clauses* for e
- e existential and after any other variable in θ

$$\Phi = \exists a \forall x \exists b \forall z \, \psi$$

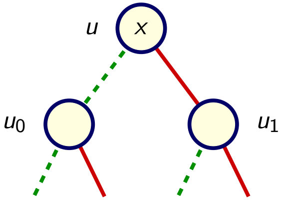
$$\theta \qquad e \vee \overline{a} \vee \overline{x} \qquad \overline{e} \vee a \qquad \overline{e} \vee x$$

$$\Phi' = \exists a \forall x \exists e \, \exists b \forall z \, (\psi \cup \theta)$$

- Subsequent clauses can use e to represent $a \wedge x$.

Clausal Representation of BDD

- Sinz & Biere 2006
- Create extension variable for each nonleaf node in BDD
  - Notation: Same symbol for node and its extension variable



- Defining clauses encode constraint $u \leftrightarrow \mathrm{ITE}(x, u_1, u_0)$

Clause name	Formula	Clausal form
$\mathrm{HD}(u)$	$x \rightarrow (u \rightarrow u_1)$	$\overline{x} \vee \overline{u} \vee u_1$
$\mathrm{LD}(u)$	$\overline{x} \rightarrow (u \rightarrow u_0)$	$x \vee \overline{u} \vee u_0$
$\mathrm{HU}(u)$	$x \rightarrow (u_1 \rightarrow u)$	$\overline{x} \vee \overline{u}_1 \vee u$
$\mathrm{LU}(u)$	$\overline{x} \rightarrow (u_0 \rightarrow u)$	$x \vee \overline{u}_0 \vee u$

Proof-Generating Versions of BDD Algorithms

- Bryant & Heule TACAS 2021
- Algorithms generate both BDD nodes and extended resolution proofs
- u, v, w: Both BDD nodes and extension variables
- Proofs capture underlying logic of algorithms

Operation	Truth Preserving	Falsehood Preserving
$w = \text{Apply}(u, v, \wedge)$	$u \wedge v \rightarrow w$	$w \rightarrow u$, $w \rightarrow v$
$w = \text{Apply}(u, v, \vee)$	$u \rightarrow w$, $v \rightarrow w$	$w \rightarrow (u \vee v)$
$w = \text{Restrict}(u, x, 1)$	$x \rightarrow (u \rightarrow w)$	$x \rightarrow (w \rightarrow u)$
$w = \text{Restrict}(u, x, 0)$	$\overline{x} \rightarrow (u \rightarrow w)$	$\overline{x} \rightarrow (w \rightarrow u)$

Overall Operation: Solver & Prover

Maintain set of active terms. Term u consists of:

- Root node u of BDD
- Unit clause with extension variable u.
- Set of defining clauses $\theta(u)$ for subgraph with root u.

Initially:

- Generate term for each input clause
- Add unit clause for term; delete input clause

Operations:

- Replace two terms u and v with conjunction $w = u \land v$.
- Replace term u with $w = \exists x\, u$ or $w = \forall x\, u$

Effect:

- Compute new BDD root w
- Add unit clause w
- Delete unit clauses for argument u (and possibly v)

Required Capability #1

Generate and validate BDD representations of clauses

Given:

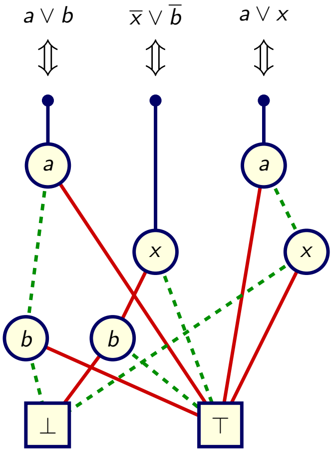
C: Input clause

u: Root node of its BDD representation

$\theta(u)$: Defining clauses for subgraph with root u

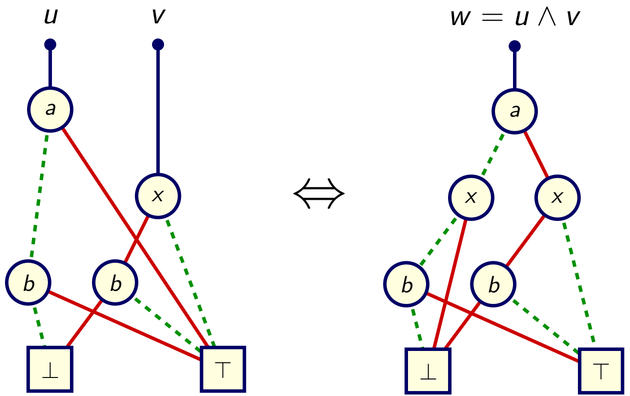
Generate Resolution Proofs:

- $C, \theta(u) \vdash u$
  - Assert unit clause for root
- $u, \theta(u) \vdash C$
  - Input clause can be deleted



Required Capability #2

Replace two BDDs by their conjunction

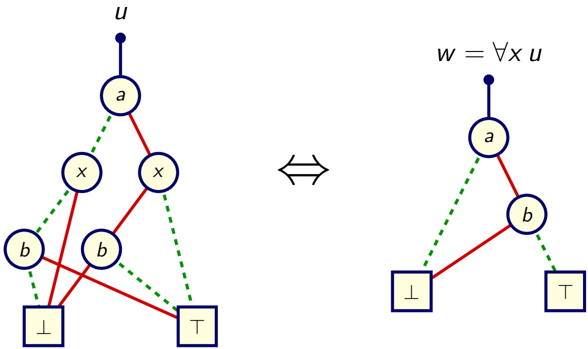


Proof Steps

- Have unit clauses u and v
- Combine with proof that $u \land v \rightarrow w$ to assert unit clause w
- Use proofs that $w \rightarrow u$ and $w \rightarrow v$ to justify deleting unit clauses u and v

Required Capability #3

Replace BDD by its universal quantification



Decompose

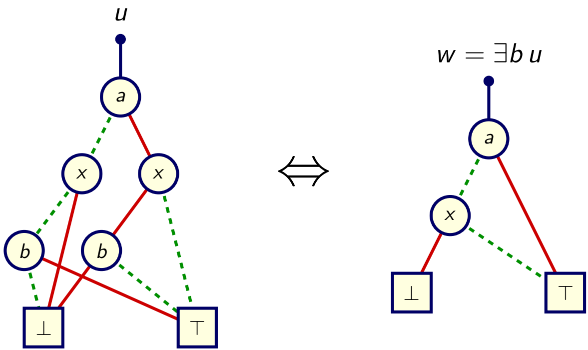
- $u_1 = \text{Restrict}(u, x, 1)$
- $u_0 = \text{Restrict}(u, x, 0)$
- $w = \text{Apply}(u_1, u_0, \wedge)$

Proofs:

- Assert unit clauses u_1 and u_0
  - Resolution, universal reduction
- Delete unit clause u
- Conjunct u_0 and u_1 to get w

Required Capability #4

Replace BDD by its existential quantification



Decompose

- $u_1 = \text{Restrict}(u, b, 1)$
- $u_0 = \text{Restrict}(u, b, 0)$
- $w = \text{Apply}(u_1, u_0, \vee)$

Proofs (Tricky!):

- Assert unit clause w
  - Resolution, extension, and existential elimination
- Delete unit clause u

Overall Proof Structure

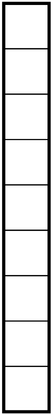
Representation at each step i

- Root nodes $T_i = \{u_1, u_2, \ldots, u_k\}$.
- Clauses for each root node u_j:
  - Unit clause u_j
  - Defining clauses $\theta(u_j)$

Possible outcomes

- Generate $u_j = \perp$
  - Creates empty clause
  - Refutation proof
- Reach step i with $T_i = \emptyset$.
  - Never add root node $\top$ to set.
  - Clause set empty
  - Satisfaction proof

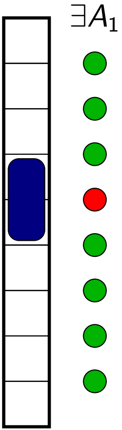
QBF Benchmark Problem: Linear Domino Game



Board with $1 \times N$ squares

- Players alternate placing dominos
- First player who can't place domino loses
- At most $\lfloor N/2 \rfloor$ moves
- B wins for:
  - $N \in \{0, 1, 15, 35\}$
  - $34i + c$
    - $i \geq 0$
    - $c \in \{5, 9, 21, 25, 29\}$
- **OEIS Sequence A215721**

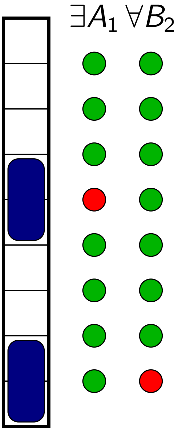
QBF Benchmark Problem: Linear Domino Game



Board with $1 \times N$ squares

- Players alternate placing dominos
- First player who can't place domino loses
- At most $\lfloor N/2 \rfloor$ moves
- B wins for:
  - $N \in \{0, 1, 15, 35\}$
  - $34i + c$
    - $i \geq 0$
    - $c \in \{5, 9, 21, 25, 29\}$
- **OEIS Sequence A215721**

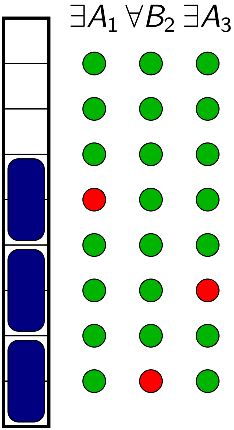
QBF Benchmark Problem: Linear Domino Game



Board with $1 \times N$ squares

- Players alternate placing dominos
- First player who can't place domino loses
- At most $\lfloor N/2 \rfloor$ moves
- B wins for:
  - $N \in \{0, 1, 15, 35\}$
  - $34i + c$
    - $i \geq 0$
    - $c \in \{5, 9, 21, 25, 29\}$
- **OEIS Sequence A215721**

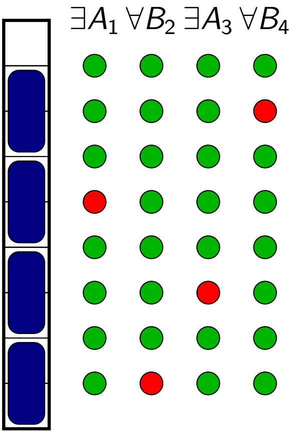
QBF Benchmark Problem: Linear Domino Game



Board with $1 \times N$ squares

- Players alternate placing dominos
- First player who can't place domino loses
- At most $\lfloor N/2 \rfloor$ moves
- B wins for:
  - $N \in \{0, 1, 15, 35\}$
  - $34i + c$
    - $i \geq 0$
    - $c \in \{5, 9, 21, 25, 29\}$
- **OEIS Sequence A215721**

QBF Benchmark Problem: Linear Domino Game

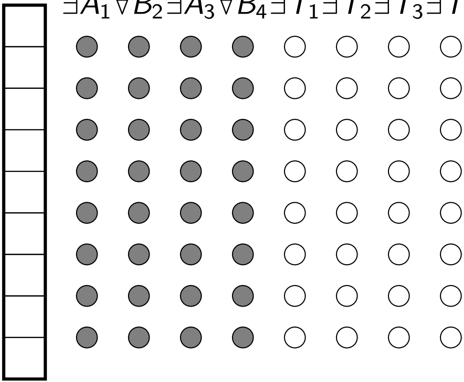


Board with $1 \times N$ squares

- Players alternate placing dominos
- First player who can't place domino loses
- At most $\lfloor N/2 \rfloor$ moves
- B wins for:
  - $N \in \{0, 1, 15, 35\}$
  - $34i + c$
    - $i \geq 0$
    - $c \in \{5, 9, 21, 25, 29\}$
- **OEIS Sequence A215721**

Linear Domino Game Encoding

$\exists A_1 \forall B_2 \exists A_3 \forall B_4 \exists T_1 \exists T_2 \exists T_3 \exists T_4$



Tseitin Variables

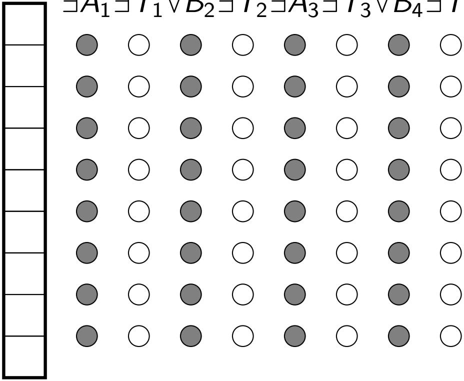
- Track state of board after each step
- Conventionally at innermost quantification level

Clauses (A as winner)

- Each move legal
- Move when possible
- Game consists of odd number of moves

Moving Tseitin Variables

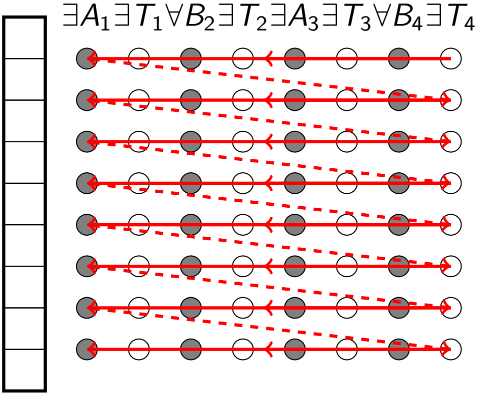
$\exists A_1 \exists T_1 \forall B_2 \exists T_2 \exists A_3 \exists T_3 \forall B_4 \exists T_4$



Moving Tseitin Variables

- Right after their defining input variables
- Avoids quantifying them out at beginning of symbolic evaluation

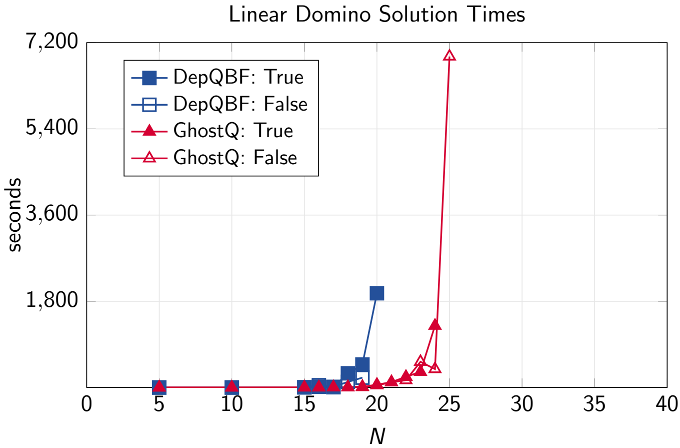
BDD Variable Ordering



Ordering

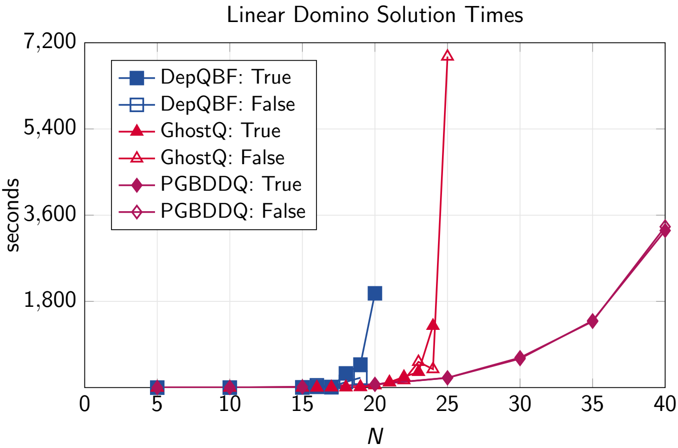
- Major ordering by position on board
- As variables quantified out, those encoding each position adjacent in ordering

Benchmarking Existing QBF Solvers



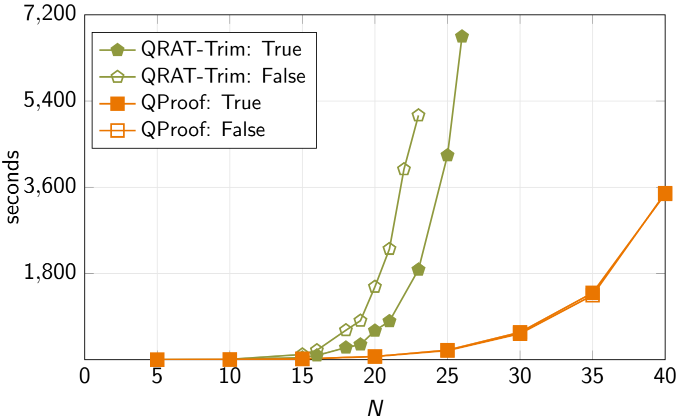
- Solve problems without proof generation
- Encode both true and false instances of problem
- Very fast until point at which times out (7200 seconds)

PGBDDQ with Dual-Proof Generation



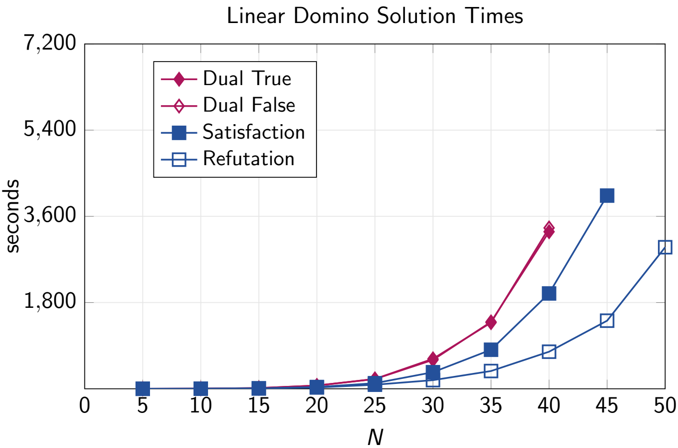
- Maintains polynomial scaling (Trend: $N^{4.8}$)
- Problem scales as $O(N^2)$ variables and $O(N^3)$ clauses

Dual Proof Checking Performance



- QRAT-Trim: Existing QRAT checker
  - Must search for justifications
- QProof: Checker for limited set of proof rules
  - Proof contains detailed justification of each step
  - Checking time ≈ solving time

Optimizing for Single Proof Type



- Additional optimizations when only generating satisfaction or refutation proof
- Checking times also decrease

Conclusions and Observations

Unified proof system for true and false formulas

- Start generating proof before have determined outcome
- Single proof checker

More automation required

- Identify and move Tseitin variables
- Determine elimination ordering within quantifier block
- Determine BDD variable ordering

Applications beyond QBF solving

- Equivalence-preserving transformation from one QBF to another
  - Provably correct