Blocked Clauses in First-Order Logic

Benjamin Kiesl[1], **Martin Suda**[1], Martina Seidl[2],
Hans Tompits[1], and Armin Biere[2]

[1] TU Wien, Vienna, Austria
[2] Johannes Kepler University, Linz, Austria

Motivation

- Many automated reasoning systems use preprocessing techniques to speed up the solving process.
  - Before the actual solving starts, these techniques simplify a formula without affecting its satisfiability.

Motivation

- Many automated reasoning systems use preprocessing techniques to speed up the solving process.
  - Before the actual solving starts, these techniques simplify a formula without affecting its satisfiability.
- Most solvers operate on formulas in conjunctive normal form (CNF).

➥ CNF preprocessing is important.

Motivation

- Many automated reasoning systems use preprocessing techniques to speed up the solving process.
  - Before the actual solving starts, these techniques simplify a formula without affecting its satisfiability.
- Most solvers operate on formulas in conjunctive normal form (CNF).

➥ CNF preprocessing is important.

- Successful CNF-preprocessing techniques in SAT and QSAT solving are based on the notion of blocked clauses (Kullmann, 1999):

Motivation

- Many automated reasoning systems use preprocessing techniques to speed up the solving process.
  - Before the actual solving starts, these techniques simplify a formula without affecting its satisfiability.
- Most solvers operate on formulas in conjunctive normal form (CNF).

➡ CNF preprocessing is important.

- Successful CNF-preprocessing techniques in SAT and QSAT solving are based on the notion of blocked clauses (Kullmann, 1999):
  - Blocked-clause elimination speeds up SAT and (D)QBF solving.
  - The winner of the SATRace 2015, abcdSAT, uses blocked-clause decomposition as core technology.
  - Addition of short blocked clauses can also improve performance.

Motivation

- Many automated reasoning systems use preprocessing techniques to speed up the solving process.
  - Before the actual solving starts, these techniques simplify a formula without affecting its satisfiability.
- Most solvers operate on formulas in conjunctive normal form (CNF).

➥ CNF preprocessing is important.

- Successful CNF-preprocessing techniques in SAT and QSAT solving are based on the notion of blocked clauses (Kullmann, 1999):
  - Blocked-clause elimination speeds up SAT and (D)QBF solving.
  - The winner of the SATRace 2015, abcdSAT, uses blocked-clause decomposition as core technology.
  - Addition of short blocked clauses can also improve performance.

➥ We lift the notion of a blocked clause to first-order logic.

Main Contributions

- We lift the notion of a blocked clause to first-order logic (FOL)

Main Contributions

- We lift the notion of a blocked clause to first-order logic (FOL)
- We introduce a version of blocked clauses for FOL with equality.

Main Contributions

- We lift the notion of a blocked clause to first-order logic (FOL)
- We introduce a version of blocked clauses for FOL with equality.
- We give a polynomial time algorithm for deciding whether a clause is blocked.

Main Contributions

- We lift the notion of a blocked clause to first-order logic (FOL)
- We introduce a version of blocked clauses for FOL with equality.
- We give a polynomial time algorithm for deciding whether a clause is blocked.
- We implement blocked-clause elimination as a preprocessing technique for first-order theorem provers.

Main Contributions

- We lift the notion of a blocked clause to first-order logic (FOL)
- We introduce a version of blocked clauses for FOL with equality.
- We give a polynomial time algorithm for deciding whether a clause is blocked.
- We implement blocked-clause elimination as a preprocessing technique for first-order theorem provers.
  - We evaluate the effectiveness for various provers on the TPTP benchmark library.

Outline

1. Background:
   - Overview on preprocessing techniques.
   - Blocked clauses in propositional logic.
2. Blocked clauses in first-order logic without equality
3. Blocked clauses in first-order logic with equality (*equality-blocked clauses*)
4. Complexity of detecting blocked clauses.
5. Evaluation results for first-order blocked-clause elimination.

Preprocessing for Automated Provers

- Preprocessing methods simplify formulas.
- Given a CNF formula, they often remove or add redundant clauses.

Preprocessing for Automated Provers

- Preprocessing methods simplify formulas.
- Given a CNF formula, they often remove or add redundant clauses.
- We call a clause redundant w.r.t. a formula if its addition or removal maintians satisfiability equivalence.

Preprocessing for Automated Provers

- Preprocessing methods simplify formulas.
- Given a CNF formula, they often remove or add redundant clauses.
- We call a clause redundant w.r.t. a formula if its addition or removal maintians satisfiability equivalence.
- Examples:
  - Tautological clauses are redundant in every formula.

Preprocessing for Automated Provers

- Preprocessing methods simplify formulas.
- Given a CNF formula, they often remove or add redundant clauses.
- We call a clause redundant w.r.t. a formula if its addition or removal maintians satisfiability equivalence.
- Examples:
  - Tautological clauses are redundant in every formula.
  - Clauses containing a pure literal are redundant

Preprocessing for Automated Provers

- Preprocessing methods simplify formulas.
- Given a CNF formula, they often remove or add redundant clauses.
- We call a clause redundant w.r.t. a formula if its addition or removal maintians satisfiability equivalence.
- Examples:
  - Tautological clauses are redundant in every formula.
  - Clauses containing a pure literal are redundant
  - As we will see, blocked clauses are redundant too.

Blocked Clauses in Propositional Logic

- Intuitively, a clause C is blocked if all resolvents of C upon one of its literals are tautologies.

Blocked Clauses in Propositional Logic

- Intuitively, a clause C is blocked if all resolvents of C upon one of its literals are tautologies.

Definition (Kullmann, 1999)

A clause $C \vee b$ is blocked by the literal b in a formula F if, for every clause $D \vee \bar{b} \in F$, the resolvent $C \vee D$ is a tautology.

Blocked Clauses in Propositional Logic

- Intuitively, a clause C is blocked if all resolvents of C upon one of its literals are tautologies.

Definition (Kullmann, 1999)

A clause $C \vee b$ is blocked by the literal b in a formula F if,
for every clause $D \vee \bar{b} \in F$, the resolvent $C \vee D$ is a tautology.

- Example:

C: $x \vee y \vee b$

$\neg z \vee x \vee y$

$\neg b \vee \neg y$

$\neg b \vee \neg x$

$\neg a \vee z \vee b$

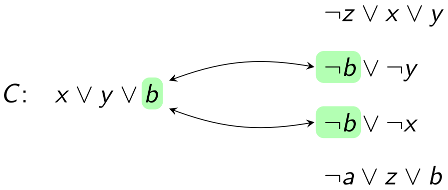
Blocked Clauses in Propositional Logic

- Intuitively, a clause C is blocked if all resolvents of C upon one of its literals are tautologies.

Definition (Kullmann, 1999)

A clause $C \vee b$ is blocked by the literal b in a formula F if,
for every clause $D \vee \bar{b} \in F$, the resolvent $C \vee D$ is a tautology.

- Example:



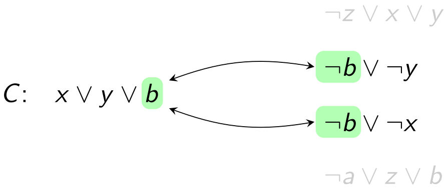
Blocked Clauses in Propositional Logic

- Intuitively, a clause C is blocked if all resolvents of C upon one of its literals are tautologies.

Definition (Kullmann, 1999)

A clause $C \vee b$ is blocked by the literal b in a formula F if,
for every clause $D \vee \bar{b} \in F$, the resolvent $C \vee D$ is a tautology.

- Example:



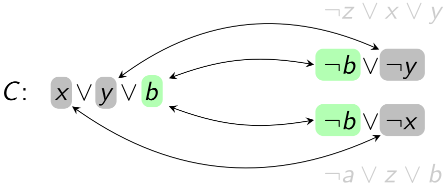
Blocked Clauses in Propositional Logic

- Intuitively, a clause C is blocked if all resolvents of C upon one of its literals are tautologies.

Definition (Kullmann, 1999)

A clause $C \vee b$ is blocked by the literal b in a formula F if, for every clause $D \vee \bar{b} \in F$, the resolvent $C \vee D$ is a tautology.

- Example:



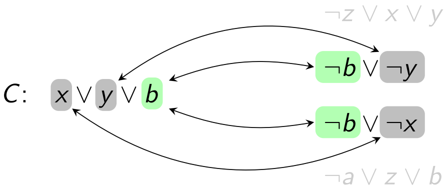
Blocked Clauses in Propositional Logic

- Intuitively, a clause C is blocked if all resolvents of C upon one of its literals are tautologies.

Definition (Kullmann, 1999)

A clause $C \vee b$ is blocked by the literal b in a formula F if,
for every clause $D \vee \bar{b} \in F$, the resolvent $C \vee D$ is a tautology.

- Example:



- Blocked clauses are redundant, so they can be safely removed/added.

Why Blocked Clauses are Redundant

- To show: $F \setminus \{C\}$ is satisfiable $\Leftrightarrow$ $F \cup \{C\}$ is satisfiable.

Why Blocked Clauses are Redundant

- To show: $F \setminus \{C\}$ is satisfiable $\Leftrightarrow$ $F \cup \{C\}$ is satisfiable.
- $\Leftarrow$ is trivial.

Why Blocked Clauses are Redundant

- To show: $F \setminus \{C\}$ is satisfiable $\Leftrightarrow$ $F \cup \{C\}$ is satisfiable.
- $\Leftarrow$ is trivial.
- $\Rightarrow$ (idea): "Repair" satisfying assignments of $F \setminus \{C\}$.

Why Blocked Clauses are Redundant

- To show: $F \setminus \{C\}$ is satisfiable $\Leftrightarrow$ $F \cup \{C\}$ is satisfiable.
- $\Leftarrow$ is trivial.
- $\Rightarrow$ (idea): "Repair" satisfying assignments of $F \setminus \{C\}$.
  - Suppose C is blocked by b and there exists a satisfying assignment of $F \setminus \{C\}$ that falsifies C.

$\neg z \vee x \vee y$

$\neg b \vee \neg y$

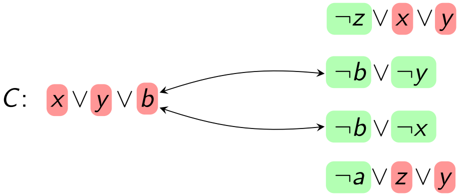
$C\colon \quad x \vee y \vee b$

$\neg b \vee \neg x$

$\neg a \vee z \vee y$

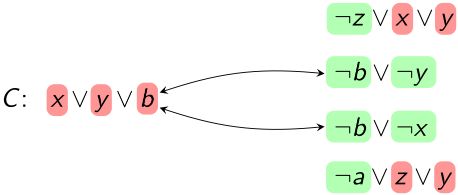
Why Blocked Clauses are Redundant

- To show: $F \setminus \{C\}$ is satisfiable $\Leftrightarrow$ $F \cup \{C\}$ is satisfiable.
- $\Leftarrow$ is trivial.
- $\Rightarrow$ (idea): "Repair" satisfying assignments of $F \setminus \{C\}$.
  - Suppose C is blocked by b and there exists a satisfying assignment of $F \setminus \{C\}$ that falsifies C.



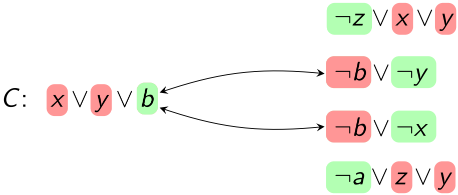
Why Blocked Clauses are Redundant

- To show: $F \setminus \{C\}$ is satisfiable $\Leftrightarrow F \cup \{C\}$ is satisfiable.
- $\Leftarrow$ is trivial.
- $\Rightarrow$ (idea): "Repair" satisfying assignments of $F \setminus \{C\}$.
  - Suppose C is blocked by b and there exists a satisfying assignment of $F \setminus \{C\}$ that falsifies C.
  - Make C true by "flipping" the truth value of b.



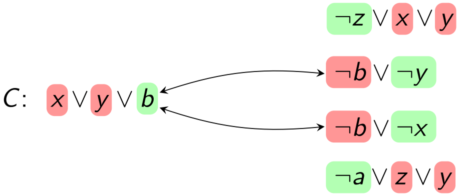
Why Blocked Clauses are Redundant

- To show: $F \setminus \{C\}$ is satisfiable $\Leftrightarrow F \cup \{C\}$ is satisfiable.
- $\Leftarrow$ is trivial.
- $\Rightarrow$ (idea): "Repair" satisfying assignments of $F \setminus \{C\}$.
  - Suppose C is blocked by b and there exists a satisfying assignment of $F \setminus \{C\}$ that falsifies C.
  - Make C true by "flipping" the truth value of b.



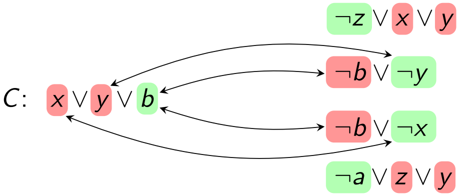
Why Blocked Clauses are Redundant

- To show: $F \setminus \{C\}$ is satisfiable $\Leftrightarrow F \cup \{C\}$ is satisfiable.
- $\Leftarrow$ is trivial.
- $\Rightarrow$ (idea): "Repair" satisfying assignments of $F \setminus \{C\}$.
  - Suppose C is blocked by b and there exists a satisfying assignment of $F \setminus \{C\}$ that falsifies C.
  - Make C true by "flipping" the truth value of b.
  - Clauses containing $\neg b$ stay true.



Why Blocked Clauses are Redundant

- To show: $F \setminus \{C\}$ is satisfiable $\Leftrightarrow$ $F \cup \{C\}$ is satisfiable.
- $\Leftarrow$ is trivial.
- $\Rightarrow$ (idea): "Repair" satisfying assignments of $F \setminus \{C\}$.
  - Suppose C is blocked by b and there exists a satisfying assignment of $F \setminus \{C\}$ that falsifies C.
  - Make C true by "flipping" the truth value of b.
  - Clauses containing $\neg b$ stay true.



Blocked Clauses in First-Order Logic (Without Equality)

- What does "all resolvents upon a literal are tautologies" mean in first-order logic?

Blocked Clauses in First-Order Logic (Without Equality)

- What does "all resolvents upon a literal are tautologies" mean in first-order logic?

Example ("binary resolvents" do not guarantee redundancy)

- Let $C = P(x, y) \vee P(y, x)$ and $F = \{\neg P(u, v) \vee \neg P(v, u)\}$

Blocked Clauses in First-Order Logic (Without Equality)

- What does "all resolvents upon a literal are tautologies" mean in first-order logic?

Example ("binary resolvents" do not guarantee redundancy)

- Let $C = P(x, y) \vee P(y, x)$ and $F = \{\neg P(u, v) \vee \neg P(v, u)\}$
- There are two binary resolvents of C upon $P(x, y)$:
  1. $P(v, u) \vee \neg P(v, u)$ via resolution with $\neg P(u, v)$

Blocked Clauses in First-Order Logic (Without Equality)

- What does "all resolvents upon a literal are tautologies" mean in first-order logic?

Example ("binary resolvents" do not guarantee redundancy)

- Let $C = P(x, y) \lor P(y, x)$ and $F = \{\neg P(u, v) \lor \neg P(v, u)\}$
- There are two binary resolvents of C upon $P(x, y)$:
  1. $P(v, u) \lor \neg P(v, u)$ via resolution with $\neg P(u, v)$
  2. $P(u, v) \lor \neg P(u, v)$ via resolution with $\neg P(v, u)$

Blocked Clauses in First-Order Logic (Without Equality)

- What does "all resolvents upon a literal are tautologies" mean in first-order logic?

Example ("binary resolvents" do not guarantee redundancy)

- Let $C = P(x, y) \vee P(y, x)$ and $F = \{\neg P(u, v) \vee \neg P(v, u)\}$
- There are two binary resolvents of C upon $P(x, y)$:
  1. $P(v, u) \vee \neg P(v, u)$ via resolution with $\neg P(u, v)$
  2. $P(u, v) \vee \neg P(u, v)$ via resolution with $\neg P(v, u)$
- Both resolvents are tautologies but C is not redundant w.r.t. F.
  - F is clearly satisfiable.
  - $F \cup \{C\}$ is unsatisfiable:

Blocked Clauses in First-Order Logic (Without Equality)

- What does "all resolvents upon a literal are tautologies" mean in first-order logic?

Example ("binary resolvents" do not guarantee redundancy)

- Let $C = P(x, y) \lor P(y, x)$ and $F = \{\neg P(u, v) \lor \neg P(v, u)\}$
- There are two binary resolvents of C upon $P(x, y)$:
  1. $P(v, u) \lor \neg P(v, u)$ via resolution with $\neg P(u, v)$
  2. $P(u, v) \lor \neg P(u, v)$ via resolution with $\neg P(v, u)$
- Both resolvents are tautologies but C is not redundant w.r.t. F.
  - F is clearly satisfiable.
  - $F \cup \{C\}$ is unsatisfiable: $P(c, c) \lor P(c, c)$ and $\neg P(c, c) \lor \neg P(c, c)$ are inconsistent ground instances of clauses in $F \cup \{C\}$.

Blocked Clauses in First-Order Logic (Without Equality)

- What does "all resolvents upon a literal are tautologies" mean in first-order logic?

Example ("binary resolvents" do not guarantee redundancy)

- Let $C = P(x, y) \vee P(y, x)$ and $F = \{\neg P(u, v) \vee \neg P(v, u)\}$
- There are two binary resolvents of C upon $P(x, y)$:
  1. $P(v, u) \vee \neg P(v, u)$ via resolution with $\neg P(u, v)$
  2. $P(u, v) \vee \neg P(u, v)$ via resolution with $\neg P(v, u)$
- Both resolvents are tautologies but C is not redundant w.r.t. F.
  - F is clearly satisfiable.
  - $F \cup \{C\}$ is unsatisfiable: $P(c, c) \vee P(c, c)$ and $\neg P(c, c) \vee \neg P(c, c)$ are inconsistent ground instances of clauses in $F \cup \{C\}$.

➥ The problem is factoring: $\neg P(u, v)$ and $\neg P(v, u)$ unify.

Blocked Clauses in First-Order Logic (Without Equality)

- Better approach: Take care of factoring.

Definition (L-resolvent)

Let $C = L \lor C$ and $D = N_1 \lor \cdots \lor N_m \lor D$ be clauses where $L, \bar{N}_1, \ldots, \bar{N}_m$ are unifiable by an mgu σ.

Blocked Clauses in First-Order Logic (Without Equality)

- Better approach: Take care of factoring.

Definition (L-resolvent)

Let $C = L \vee C$ and $D = N_1 \vee \cdots \vee N_m \vee D$ be clauses where $L, \bar{N}_1, \ldots, \bar{N}_m$ are unifiable by an mgu σ.
Then, $C\sigma \vee D\sigma$ is called L-resolvent of C and D.

Blocked Clauses in First-Order Logic (Without Equality)

- Better approach: Take care of factoring.

Definition (L-resolvent)

Let $C = L \vee C$ and $D = N_1 \vee \cdots \vee N_m \vee D$ be clauses where $L, \bar{N}_1, \ldots, \bar{N}_m$ are unifiable by an mgu σ.
Then, $C\sigma \vee D\sigma$ is called *L-resolvent* of C and D.

- Consider again $C = P(x, y) \vee P(y, x)$ and $F = \{\neg P(u, v) \vee \neg P(v, u)\}$

Blocked Clauses in First-Order Logic (Without Equality)

- Better approach: Take care of factoring.

Definition (L-resolvent)

Let $C = L \vee C$ and $D = N_1 \vee \cdots \vee N_m \vee D$ be clauses where $L, \bar{N}_1, \ldots, \bar{N}_m$ are unifiable by an mgu σ.
Then, $C\sigma \vee D\sigma$ is called L-resolvent of C and D.

- Consider again $C = P(x, y) \vee P(y, x)$ and $F = \{\neg P(u, v) \vee \neg P(v, u)\}$
- For $L = P(x, y)$, there are now three L-resolvents of C:
  1. $P(v, u) \vee \neg P(v, u)$ via resolution with $\neg P(u, v)$ (as before).

Blocked Clauses in First-Order Logic (Without Equality)

- Better approach: Take care of factoring.

Definition (L-resolvent)

Let $C = L \lor C$ and $D = N_1 \lor \cdots \lor N_m \lor D$ be clauses where $L, \bar{N}_1, \ldots, \bar{N}_m$ are unifiable by an mgu σ.
Then, $C\sigma \lor D\sigma$ is called L-resolvent of C and D.

- Consider again $C = P(x, y) \lor P(y, x)$ and $F = \{\neg P(u, v) \lor \neg P(v, u)\}$
- For $L = P(x, y)$, there are now three L-resolvents of C:
  1. $P(v, u) \lor \neg P(v, u)$ via resolution with $\neg P(u, v)$ (as before).
  2. $P(u, v) \lor \neg P(u, v)$ via resolution with $\neg P(v, u)$ (as before).

Blocked Clauses in First-Order Logic (Without Equality)

- Better approach: Take care of factoring.

Definition (L-resolvent)

Let $C = L \vee C$ and $D = N_1 \vee \cdots \vee N_m \vee D$ be clauses where $L, \bar{N}_1, \ldots, \bar{N}_m$ are unifiable by an mgu σ.
Then, $C\sigma \vee D\sigma$ is called L-resolvent of C and D.

- Consider again $C = P(x, y) \vee P(y, x)$ and $F = \{\neg P(u, v) \vee \neg P(v, u)\}$
- For $L = P(x, y)$, there are now three L-resolvents of C:
  1. $P(v, u) \vee \neg P(v, u)$ via resolution with $\neg P(u, v)$ (as before).
  2. $P(u, v) \vee \neg P(u, v)$ via resolution with $\neg P(v, u)$ (as before).
  3. $P(x, x)$ (not a tautology!) resolution with both $\neg P(u, v)$ and $\neg P(v, u)$.

Definition

A clause $C \vee L$ is blocked by L in a formula F if all L-resolvents of $C \vee L$ with clauses in $F \setminus \{C \vee L\}$ are tautologies.

Blocked Clauses Are Redundant

- Redundancy of blocked clauses can be shown via Herbrand's Theorem:

Blocked Clauses Are Redundant

- Redundancy of blocked clauses can be shown via Herbrand's Theorem:

Theorem

A formula F (without equality) is satisfiable iff every finite set of ground instances of clauses in F is propositionally satisfiable.

Blocked Clauses Are Redundant

- Redundancy of blocked clauses can be shown via Herbrand's Theorem:

Theorem

A formula F (without equality) is satisfiable iff every finite set of ground instances of clauses in F is propositionally satisfiable.

- In propositional logic, we showed how satisfying assignments of $F \setminus \{C\}$ can be "repaired" to satisfy also C.

Blocked Clauses Are Redundant

- Redundancy of blocked clauses can be shown via Herbrand's Theorem:

Theorem

A formula F (without equality) is satisfiable iff every finite set of ground instances of clauses in F is propositionally satisfiable.

- In propositional logic, we showed how satisfying assignments of $F \setminus \{C\}$ can be "repaired" to satisfy also C.
- In first-order logic, one can show that satisfying assignments for sets of ground instances of clauses in $F \setminus \{C\}$ can be repaired:

Blocked Clauses Are Redundant

- Redundancy of blocked clauses can be shown via Herbrand's Theorem:

Theorem

A formula F (without equality) is satisfiable iff every finite set of ground instances of clauses in F is propositionally satisfiable.

- In propositional logic, we showed how satisfying assignments of $F \setminus \{C\}$ can be "repaired" to satisfy also C.
- In first-order logic, one can show that satisfying assignments for sets of ground instances of clauses in $F \setminus \{C\}$ can be repaired:
  - Repaired assignments also satisfy ground instances of C.

Blocked Clauses Are Redundant

- Redundancy of blocked clauses can be shown via Herbrand's Theorem:

Theorem

A formula F (without equality) is satisfiable iff every finite set of ground instances of clauses in F is propositionally satisfiable.

- In propositional logic, we showed how satisfying assignments of $F \setminus \{C\}$ can be "repaired" to satisfy also C.
- In first-order logic, one can show that satisfying assignments for sets of ground instances of clauses in $F \setminus \{C\}$ can be repaired:
  - Repaired assignments also satisfy ground instances of C.
  - May need to iterate, finitely many times.

Blocked Clauses Are Redundant

- Redundancy of blocked clauses can be shown via Herbrand's Theorem:

Theorem

A formula F (without equality) is satisfiable iff every finite set of ground instances of clauses in F is propositionally satisfiable.

- In propositional logic, we showed how satisfying assignments of $F \setminus \{C\}$ can be "repaired" to satisfy also C.
- In first-order logic, one can show that satisfying assignments for sets of ground instances of clauses in $F \setminus \{C\}$ can be repaired:
  - Repaired assignments also satisfy ground instances of C.
  - May need to iterate, finitely many times.
  - (Self-resolutions do not need to be considered.)

Blocked Clauses Are Redundant

- Redundancy of blocked clauses can be shown via Herbrand's Theorem:

Theorem

A formula F (without equality) is satisfiable iff every finite set of ground instances of clauses in F is propositionally satisfiable.

- In propositional logic, we showed how satisfying assignments of $F \setminus \{C\}$ can be "repaired" to satisfy also C.
- In first-order logic, one can show that satisfying assignments for sets of ground instances of clauses in $F \setminus \{C\}$ can be repaired:
  - Repaired assignments also satisfy ground instances of C.
  - May need to iterate, finitely many times.
  - (Self-resolutions do not need to be considered.)

➡ Blocked clauses are redundant in first-order logic (without equality).

Blocked Clauses in First-Order Logic With Equality

- In FOL with equality, our definition does not guarantee redundancy:

Blocked Clauses in First-Order Logic With Equality

- In FOL with equality, our definition does not guarantee redundancy:

Example

- Let $C = P(a)$ and $F = \{a \approx b, \neg P(b)\}$

Blocked Clauses in First-Order Logic With Equality

- In FOL with equality, our definition does not guarantee redundancy:

Example

- Let $C = P(a)$ and $F = \{a \approx b, \neg P(b)\}$
  - There are no resolvents of C since $P(a)$ and $P(b)$ are not unifiable (a and b are constants)

Blocked Clauses in First-Order Logic With Equality

- In FOL with equality, our definition does not guarantee redundancy:

Example

- Let $C = P(a)$ and $F = \{a \approx b, \neg P(b)\}$
  - There are no resolvents of C since $P(a)$ and $P(b)$ are not unifiable (a and b are constants)

➥ C is blocked w.r.t. F.

Blocked Clauses in First-Order Logic With Equality

- In FOL with equality, our definition does not guarantee redundancy:

Example

- Let $C = P(a)$ and $F = \{a \approx b, \neg P(b)\}$
  - There are no resolvents of C since $P(a)$ and $P(b)$ are not unifiable (a and b are constants)
- ➥ C is blocked w.r.t. F.
  - But F is satisfiable while $F \cup \{C\}$ is not:

Blocked Clauses in First-Order Logic With Equality

- In FOL with equality, our definition does not guarantee redundancy:

Example

- Let $C = P(a)$ and $F = \{a \approx b, \neg P(b)\}$
  - There are no resolvents of C since $P(a)$ and $P(b)$ are not unifiable (a and b are constants)

  ➥ C is blocked w.r.t. F.

  - But F is satisfiable while $F \cup \{C\}$ is not:
  - Every model of F must falsify $P(a)$.

Blocked Clauses in First-Order Logic With Equality

- In FOL with equality, our definition does not guarantee redundancy:

Example

- Let $C = P(a)$ and $F = \{a \approx b, \neg P(b)\}$
  - There are no resolvents of C since $P(a)$ and $P(b)$ are not unifiable (a and b are constants)
  - ➥ C is blocked w.r.t. F.
  - But F is satisfiable while $F \cup \{C\}$ is not:
  - Every model of F must falsify $P(a)$.

- Idea [KK2016]: Flatten literals before resolution:
  - $P(a) \longrightarrow x \not\approx a \vee P(x)$

Blocked Clauses in First-Order Logic With Equality

- In FOL with equality, our definition does not guarantee redundancy:

Example

- Let $C = P(a)$ and $F = \{a \approx b, \neg P(b)\}$
  - There are no resolvents of C since $P(a)$ and $P(b)$ are not unifiable (a and b are constants)
  - ➥ C is blocked w.r.t. F.
  - But F is satisfiable while $F \cup \{C\}$ is not:
  - Every model of F must falsify $P(a)$.

- Idea [KK2016]: Flatten literals before resolution:
  - $P(a) \longrightarrow x \not\approx a \vee P(x)$
  - $\neg P(b) \longrightarrow y \not\approx b \vee \neg P(y)$

Blocked Clauses in First-Order Logic With Equality

- In FOL with equality, our definition does not guarantee redundancy:

Example

- Let $C = P(a)$ and $F = \{a \approx b, \neg P(b)\}$
  - There are no resolvents of C since $P(a)$ and $P(b)$ are not unifiable (a and b are constants)
  - ➥ C is blocked w.r.t. F.
  - But F is satisfiable while $F \cup \{C\}$ is not:
  - Every model of F must falsify $P(a)$.

- Idea [KK2016]: Flatten literals before resolution:
  - $P(a) \longrightarrow x \not\approx a \vee P(x)$
  - $\neg P(b) \longrightarrow y \not\approx b \vee \neg P(y)$
  - Resolvent of $x \not\approx a \vee P(x)$ and $x \not\approx a \vee \neg P(y)$ upon $P(x)$:

$$x \not\approx a \vee x \not\approx b,$$

is not valid.

Blocked Clauses in First-Order Logic With Equality

Definition (Flattening)

Let $C = L(t_1, \ldots, t_n) \vee C'$. Flattening the literal $L(t_1, \ldots, t_n)$ in C yields the clause $C^- = \bigvee_{1 \leq i \leq n} x_i \not\approx t_i \vee L(x_1, \ldots, x_n) \vee C'$, with $x_i, \ldots, x_n$ being fresh variables not occurring in C.

Blocked Clauses in First-Order Logic With Equality

Definition (Flattening)

Let $C = L(t_1, \ldots, t_n) \vee C'$. Flattening the literal $L(t_1, \ldots, t_n)$ in C yields the clause $C^- = \bigvee_{1 \leq i \leq n} x_i \not\approx t_i \vee L(x_1, \ldots, x_n) \vee C'$, with $x_i, \ldots, x_n$ being fresh variables not occurring in C.

- A clause $C \vee L$ is then equality-blocked if, after flattening all literals with the same predicate as L, all flat L-resolvents are valid.

Blocked Clauses in First-Order Logic With Equality

Definition (Flattening)

Let $C = L(t_1, \ldots, t_n) \vee C'$. Flattening the literal $L(t_1, \ldots, t_n)$ in C yields the clause $C^- = \bigvee_{1 \leq i \leq n} x_i \not\approx t_i \vee L(x_1, \ldots, x_n) \vee C'$, with $x_i, \ldots, x_n$ being fresh variables not occurring in C.

- A clause $C \vee L$ is then equality-blocked if, after flattening all literals with the same predicate as L, all flat L-resolvents are valid.
- Redundancy can be shown using a variant of Herbrand's Theorem with equality axioms.

Blocked Clauses in First-Order Logic With Equality

Definition (Flattening)

Let $C = L(t_1, \ldots, t_n) \vee C'$. Flattening the literal $L(t_1, \ldots, t_n)$ in C yields the clause $C^- = \bigvee_{1 \leq i \leq n} x_i \not\approx t_i \vee L(x_1, \ldots, x_n) \vee C'$, with $x_i, \ldots, x_n$ being fresh variables not occurring in C.

- A clause $C \vee L$ is then equality-blocked if, after flattening all literals with the same predicate as L, all flat L-resolvents are valid.
- Redundancy can be shown using a variant of Herbrand's Theorem with equality axioms.
- Flipping whole equivalence classes!

Blocked Clauses in First-Order Logic With Equality

Definition (Flattening)

Let $C = L(t_1, \ldots, t_n) \vee C'$. Flattening the literal $L(t_1, \ldots, t_n)$ in C yields the clause $C^- = \bigvee_{1 \leq i \leq n} x_i \not\approx t_i \vee L(x_1, \ldots, x_n) \vee C'$, with $x_i, \ldots, x_n$ being fresh variables not occurring in C.

- A clause $C \vee L$ is then equality-blocked if, after flattening all literals with the same predicate as L, all flat L-resolvents are valid.
- Redundancy can be shown using a variant of Herbrand's Theorem with equality axioms.
- Flipping whole equivalence classes!
  - (cannot block on the equality literals)

Blocked Clauses in First-Order Logic With Equality

Definition (Flattening)

Let $C = L(t_1, \ldots, t_n) \vee C'$. Flattening the literal $L(t_1, \ldots, t_n)$ in C yields the clause $C^{-} = \bigvee_{1 \leq i \leq n} x_i \not\approx t_i \vee L(x_1, \ldots, x_n) \vee C'$, with $x_i, \ldots, x_n$ being fresh variables not occurring in C.

- A clause $C \vee L$ is then equality-blocked if, after flattening all literals with the same predicate as L, all flat L-resolvents are valid.
- Redundancy can be shown using a variant of Herbrand's Theorem with equality axioms.
- Flipping whole equivalence classes!
  - (cannot block on the equality literals)

➥ Equality-blocked clauses are redundant in first-order logic with equality.

Outline

1. Background:
    - Overview on preprocessing techniques.
    - Blocked clauses in propositional logic.
2. Blocked clauses in first-order logic without equality
3. Blocked clauses in first-order logic with equality
   (*equality-blocked clauses*)
4. **Complexity of detecting blocked clauses.**
5. Evaluation results for first-order blocked-clause elimination.

Complexity of Detecting Blocked Clauses

The easy part:

- Given a candidate clause $C = L \vee C'$ there are only linearly many partner clauses $D \in F \setminus \{C\}$ to check for L-resolvents.

Complexity of Detecting Blocked Clauses

The easy part:

- Given a candidate clause $C = L \vee C'$ there are only linearly many partner clauses $D \in F \setminus \{C\}$ to check for L-resolvents.

A single flat L-resolvent check (the $\approx$ case)

- Computing a flat L-resolvent is easy (take the obvious small *mgu*).
- R is valid iff $\neg\forall R$ is unsatisfiable.
- $\neg\forall R$ is a ground equational conjunction: use congruence closure.

Complexity of Detecting Blocked Clauses

The easy part:

- Given a candidate clause $C = L \vee C'$ there are only linearly many partner clauses $D \in F \setminus \{C\}$ to check for L-resolvents.

A single flat L-resolvent check (the $\approx$ case)

- Computing a flat L-resolvent is easy (take the obvious small *mgu*).
- R is valid iff $\neg\forall R$ is unsatisfiable.
- $\neg\forall R$ is a ground equational conjunction: use congruence closure.

A single L-resolvent check (blocked in the absence of $\approx$)

Rely on unification closure to avoid the risk of exponential unifiers.

Complexity of Detecting Blocked Clauses

The easy part:

- Given a candidate clause $C = L \vee C'$ there are only linearly many partner clauses $D \in F \setminus \{C\}$ to check for L-resolvents.

A single flat L-resolvent check (the $\approx$ case)

- Computing a flat L-resolvent is easy (take the obvious small *mgu*).
- R is valid iff $\neg\forall R$ is unsatisfiable.
- $\neg\forall R$ is a ground equational conjunction: use congruence closure.

A single L-resolvent check (blocked in the absence of $\approx$)

Rely on unification closure to avoid the risk of exponential unifiers.

The Main Challenge

Given $C = L \vee C'$ and $D = N_1 \vee \cdots \vee N_n \vee D'$ such that $L, \bar{N}_1, \ldots, \bar{N}_n$ unify, there are $2^n - 1$ L-resolvents whose validity should be checked!

Testing Validity of all L-resolvents

Algorithm 1:

Input:

Candidate $C = L \vee C'$ and a partner $D = N_1 \vee \ldots \vee N_n \vee D'$, where $N_1, \ldots, N_n$ are all the literals of D which pairwise unify with $\bar{L}$

Output:

Are all L-resolvents of $L \vee C'$ and D valid?

for $k \leftarrow 1, \ldots, n$ **do**
  $N \leftarrow \{N_k\}$
  while L unifiable with literals $\bar{N}$ via an *mgu* σ **do**
    $K \leftarrow$ all pairs of complementary literals in $C'\sigma \vee (D \setminus N)\sigma$
    if $K = \emptyset$ **then**
      return NO
    if every pair of complementary literals in K contains a literal $N_i\sigma$ **then**
      $N \leftarrow N \cup \{N_i \mid N_i\sigma$ is part of a complementary pair$\}$
    else
      break (the while loop)
return YES

Outline

1. Background:
   - Overview on preprocessing techniques.
   - Blocked clauses in propositional logic.
2. Blocked clauses in first-order logic without equality
3. Blocked clauses in first-order logic with equality (*equality-blocked clauses*)
4. Complexity of detecting blocked clauses.
5. **Evaluation results for first-order blocked-clause elimination.**

Blocked Clause Elimination (BCE) – Implementation

Implemented in automated theorem prover Vampire

- as an optional preprocessing step: `-bce on`
- blocked / equality-blocked based on the presence of $\approx$
- Vampire as a clausifier: `-mode clausify`

Blocked Clause Elimination (BCE) – Implementation

Implemented in automated theorem prover Vampire

- as an optional preprocessing step: `-bce on`
- blocked / equality-blocked based on the presence of $\approx$
- Vampire as a clausifier: `-mode clausify`

The main loop

- index clauses by predicate and polarity
- priority queue of candidates (C, L); the least effort first
- if candidate (C, L) and partner D do not yield a valid L-resolvent store (C, L) with D for potential later "resurrection"

Blocked Clause Elimination (BCE) – Implementation

Implemented in automated theorem prover Vampire

- as an optional preprocessing step: `-bce on`
- blocked / equality-blocked based on the presence of $\approx$
- Vampire as a clausifier: `-mode clausify`

The main loop

- index clauses by predicate and polarity
- priority queue of candidates (C, L); the least effort first
- if candidate (C, L) and partner D do not yield a valid L-resolvent store (C, L) with D for potential later "resurrection"

Single candidate-partner check

- cheap but safe approximation of Algorithm 1
- cheap approximation of congruence closure reasoning

Do Blocked Clauses Occur in Practice?

The setup:

- All first-order benchmarks from TPTP 6.4.0:
  - 15 942 problems: 8044 general FO, 7898 CNF
  - 73 % with equality
- 300 s for parsing, (clausification), and blocked-clause elimination

Do Blocked Clauses Occur in Practice?

The setup:

- All first-order benchmarks from TPTP 6.4.0:
  - 15 942 problems: 8044 general FO, 7898 CNF
  - 73 % with equality
- 300 s for parsing, (clausification), and blocked-clause elimination

Results

- Blocked-clause elimination finished on all but one problems.
- Average time: 0.238 s, median 0.001 s.
- 11.72 % of the collective total of 299 379 591 clauses are blocked.
- 59 % of problems contain a blocked clause.
- In more than 1000 problems, 25 % of the clauses could be eliminated.
- 113 satisfiable formulas directly solved by blocked-clause elimination.

Impact of BCE on Performance

The setup:

- 7619 TPTP problems where BCE eliminates at least one clause
- provers from CASC 2016, but fixed a single strategy
- 300 s for both BCE and proving

Impact of BCE on Performance

The setup:

- 7619 TPTP problems where BCE eliminates at least one clause
- provers from CASC 2016, but fixed a single strategy
- 300 s for both BCE and proving

Strategies for proving theorems (CASC 2016 FOF champions)

	unsatisfiable			satisfiable			total		
Vampire	**3172**	−28	+40	**458**	−0	+5	**3630**	−28	+45
E	**3097**	−20	+27	**363**	−1	+9	**3460**	−21	+36
CVC4	**2930**	−18	+37	**9**	−0	+68	**2939**	−18	+105

Impact of BCE on Performance

The setup:

- 7619 TPTP problems where BCE eliminates at least one clause
- provers from CASC 2016, but fixed a single strategy
- 300 s for both BCE and proving

Strategies for proving theorems (CASC 2016 FOF champions)

	unsatisfiable			satisfiable			total		
Vampire	**3172**	−28	+40	**458**	−0	+5	**3630**	−28	+45
E	**3097**	−20	+27	**363**	−1	+9	**3460**	−21	+36
CVC4	**2930**	−18	+37	**9**	−0	+68	**2939**	−18	+105

Satisfiability checking strategies (CASC 2016 FNT champions)

	satisfiable			unsatisfiable			total		
Vampire	**531**	−0	+24	**719**	−4	+5	**1250**	−4	+29
iProver	**558**	−0	+1	**755**	−6	+4	**1313**	−6	+5
CVC4	**489**	−1	+28	**1724**	−24	+20	**2213**	−25	+48

Mock Portfolio Construction

Portfolios?

- Provers usually employ many strategies to prove a conjecture.
- A new technique may interact differently with each of the strategies.

Mock Portfolio Construction

Portfolios?

- Provers usually employ many strategies to prove a conjecture.
- A new technique may interact differently with each of the strategies.

The setup:

- 302 satisfiable TPTP problems previous established hard for Vampire.
- 50 000 pairs of randomized strategies (w/o BCE, with BCE).
- 120 s time limit

Mock Portfolio Construction

Portfolios?

- Provers usually employ many strategies to prove a conjecture.
- A new technique may interact differently with each of the strategies.

The setup:

- 302 satisfiable TPTP problems previous established hard for Vampire.
- 50 000 pairs of randomized strategies (w/o BCE, with BCE).
- 120 s time limit

Results

- w/o BCE: 6766 successes, with BCE: **8414** successes
- only w/o BCE: 148, only with BCE: **1796**

Conclusion

Summary

- We lifted blocked clauses to first-order logic (both without and with equality).

Conclusion

Summary

- We lifted blocked clauses to first-order logic (both without and with equality).
- Blockedness can be checked in polynomial time.

Conclusion

Summary

- We lifted blocked clauses to first-order logic (both without and with equality).
- Blockedness can be checked in polynomial time.
- We implemented blocked-clause elimination in Vampire.

Conclusion

Summary

- We lifted blocked clauses to first-order logic (both without and with equality).
- Blockedness can be checked in polynomial time.
- We implemented blocked-clause elimination in Vampire.
- Encouraging experimental results.

Conclusion

Summary

- We lifted blocked clauses to first-order logic (both without and with equality).
- Blockedness can be checked in polynomial time.
- We implemented blocked-clause elimination in Vampire.
- Encouraging experimental results.

Outlook

- Combine with other first-order preprocessing techniques (such as predicate elimination [KK2016]).

Conclusion

Summary

- We lifted blocked clauses to first-order logic (both without and with equality).
- Blockedness can be checked in polynomial time.
- We implemented blocked-clause elimination in Vampire.
- Encouraging experimental results.

Outlook

- Combine with other first-order preprocessing techniques (such as predicate elimination [KK2016]).
- Lift other preprocessing techniques from SAT: [KS2017]

Conclusion

Summary

- We lifted blocked clauses to first-order logic
  (both without and with equality).
- Blockedness can be checked in polynomial time.
- We implemented blocked-clause elimination in Vampire.
- Encouraging experimental results.

Outlook

- Combine with other first-order preprocessing techniques
  (such as predicate elimination [KK2016]).
- Lift other preprocessing techniques from SAT: [KS2017]
- Beyond elimination: addition and decomposition in FO?

BCE and Unused Definition Elimination

Propositionally, BCE simulates UDE:

$$\neg p \lor \varphi \qquad p \lor \neg\varphi \qquad \psi[p]$$

for any $\neg p \lor C \in CNF(\varphi)$ and $p \lor D \in CNF(\neg\varphi)$ the resolvents

$$C \lor D$$

are tautologies (unless we name "asymmetrically" inside φ)

This breaks down with quantifiers:

Example

Given a definition $p(x) \equiv \exists y.q(x,y)$, trying to block

$$\neg p(x) \lor q(x, sk_{\exists y.q(x,y)}(x)) \qquad p(x) \lor \neg q(x,y)$$