A Unifying Principle for Clause Elimination in First-Order Logic

Benjamin Kiesl       Martin Suda

Institute for Logic and Computation, TU Wien
Preprocessing techniques for first-order theorem provers.

- Improve the efficiency of provers by simplifying the input.
Preprocessing techniques for first-order theorem provers.

- Improve the efficiency of provers by simplifying the input.

In particular, clause-elimination techniques:

- Remove redundant clauses from a formula in CNF.
Preprocessing techniques for first-order theorem provers.

- Improve the efficiency of provers by simplifying the input.

In particular, clause-elimination techniques:

- Remove redundant clauses from a formula in CNF.

Many clause-elimination techniques are used in SAT solving but not in first-order logic yet.
Preprocessing techniques for first-order theorem provers.

- Improve the efficiency of provers by simplifying the input.

In particular, clause-elimination techniques:

- Remove redundant clauses from a formula in CNF.

Many clause-elimination techniques are used in SAT solving but not in first-order logic yet.

We lifted SAT techniques to first-order logic without equality.
Preprocessing techniques for first-order theorem provers.  
  • Improve the efficiency of provers by simplifying the input.

In particular, clause-elimination techniques:
  • Remove redundant clauses from a formula in CNF.

Many clause-elimination techniques are used in SAT solving but not in first-order logic yet.

We lifted SAT techniques to first-order logic without equality.
  • We proved correctness in a uniform way by introducing the principle of implication modulo resolution.
First-order theorem proving and preprocessing in a nutshell.

Details on one successful approach for preprocessing:
  - Clause-elimination techniques.

Overview of techniques we lifted.

The unifying principle of implication modulo resolution.

Confluence results.

Future work.
First-Order Theorem Proving

- **Input**: Formula in first-order logic.
- **Output**: Proof

\[ Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]
First-Order Theorem Proving

- **Input**: Formula in first-order logic.
- **Output**: Proof
- **Applications**: Mathematics, verification of software and hardware, reasoning over knowledge bases, etc.

\[ Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]
Automatic First-Order Theorem Proving

\[ Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]
Automatic First-Order Theorem Proving

\[
Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a)))
\]
Automatic First-Order Theorem Proving

\[ Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]
Automatic First-Order Theorem Proving

\[ Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]
Automatic First-Order Theorem Proving

\[ Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]
Automatic First-Order Theorem Proving

\[ Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]
Resolution Rule: Derive $C \lor D$ from $C \lor L$ and $\neg L \lor D$:

$$
\begin{array}{c}
C \lor L \\
\neg L \lor D
\end{array}
\quad
\begin{array}{c}
\hline
\end{array}
\quad
\begin{array}{c}
C \lor D
\end{array}
$$

Every unsatisfiable formula can be refuted by resolution.

Example:

$$F = (\neg P \lor Q) \land (P) \land (\neg Q)$$

$$\neg P \lor Q \quad P \quad \neg Q \quad \bot$$
Resolution Refutations (Propositional Logic)

- Resolution Rule: Derive $C \lor D$ from $C \lor L$ and $\neg L \lor D$:

  $\begin{array}{c}
  C \lor L \\
  \neg L \lor D \\
  \hline
  C \lor D
  \end{array}$

$C \lor D$ is a resolvent of $C \lor L$ upon $L$. 

Every unsatisfiable formula can be refuted by resolution.

Example: $F = (\neg P \lor Q) \land (P \land \neg Q)$
Resolution Refutations (Propositional Logic)

- Resolution Rule: Derive $C \lor D$ from $C \lor L$ and $\neg L \lor D$:

  \[
  \begin{array}{c}
  C \lor L \\
  \neg L \lor D
  \end{array}
  \overline{\quad}
  C \lor D
  \]

  $C \lor D$ is a resolvent of $C \lor L$ upon $L$.

- Every unsatisfiable formula can be refuted by resolution.
Resolution Refutations (Propositional Logic)

**Resolution Rule:** Derive $C \lor D$ from $C \lor L$ and $\neg L \lor D$:

\[
\begin{array}{c}
C \lor L \\
\hline
\neg L \lor D \\
\hline
C \lor D
\end{array}
\]

$C \lor D$ is a **resolvent** of $C \lor L$ upon $L$.

**Every** unsatisfiable formula can be refuted by resolution.

**Example:** $F = (\neg P \lor Q) \land (P) \land (\neg Q)$

\[
\begin{array}{c}
\neg P \lor Q \\
\hline
P
\end{array}
\]

\[
\begin{array}{c}
Q \\
\hline
\neg Q
\end{array}
\]

\[
\bot
\]
Resolution Refutations (First-Order Logic)

- **Resolution Rule**: Derive \((C \lor D)\sigma\) from \(C \lor L(t_1, \ldots, t_n)\) and \(\neg L(s_1, \ldots, s_n) \lor D\) if \(\sigma\) unifies \(L(t_1, \ldots, t_n)\) and \(L(s_1, \ldots, s_n)\):

Intuitively, a mapping \(\sigma\) unifies literals if it makes them equal:

- \(P(x, y)\) and \(P(a, b)\) are unifiable \(\rightarrow \sigma(x) = a\) and \(\sigma(y) = b\).
- \(P(b, a)\) and \(P(b, a)\) are unifiable \(\rightarrow\) no mapping necessary.

**Example Refutation**:

\[\neg P(x, y) \lor P(y, x) \land P(a, b) \land \neg P(b, a)\]
Resolution Refutations (First-Order Logic)

- Resolution Rule: Derive \((C \lor D)\sigma\) from \(C \lor L(t_1, \ldots, t_n)\) and \(\neg L(s_1, \ldots, s_n) \lor D\) if \(\sigma\) unifies \(L(t_1, \ldots, t_n)\) and \(L(s_1, \ldots, s_n)\):

- Intuitively, a mapping \(\sigma\) unifies literals if it makes them equal:
  - \(P(x, y)\) and \(P(a, b)\) are unifiable \(\rightarrow \sigma(x) = a\) and \(\sigma(y) = b\).
Resolution Refutations (First-Order Logic)

- **Resolution Rule:** Derive \((C \lor D)\sigma\) from \(C \lor L(t_1, \ldots, t_n)\) and \(\neg L(s_1, \ldots, s_n) \lor D\) if \(\sigma\) unifies \(L(t_1, \ldots, t_n)\) and \(L(s_1, \ldots, s_n)\):

- Intuitively, a mapping \(\sigma\) unifies literals if it makes them equal:
  
  - \(P(x, y)\) and \(P(a, b)\) are unifiable \(\rightarrow \sigma(x) = a\) and \(\sigma(y) = b\).
  - \(P(b, a)\) and \(P(b, a)\) are unifiable \(\rightarrow\) no mapping necessary.
Resolution Refutations (First-Order Logic)

- **Resolution Rule:** Derive \((C \lor D)\sigma\) from \(C \lor L(t_1, \ldots, t_n)\) and \(\neg L(s_1, \ldots, s_n) \lor D\) if \(\sigma\) unifies \(L(t_1, \ldots, t_n)\) and \(L(s_1, \ldots, s_n)\):

Intuitively, a mapping \(\sigma\) unifies literals if it makes them equal:

- \(P(x, y)\) and \(P(a, b)\) are unifiable \(\rightarrow \sigma(x) = a\) and \(\sigma(y) = b\).
- \(P(b, a)\) and \(P(b, a)\) are unifiable \(\rightarrow\) no mapping necessary.

- **Example Refutation:**

\[F = (\neg P(x, y) \lor P(y, x)) \land P(a, b) \land \neg P(b, a)\]

\[
\begin{array}{c}
\neg P(x, y) \lor P(y, x) \\
P(a, b) \\
\hline
P(b, a) \\
\hline
\neg P(b, a)
\end{array}
\]

\(\bot\)
Automatic First-Order Theorem Proving

\[ Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]

Resolution Refutation
Automatic First-Order Theorem Proving

\[ Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]

What’s going on here?

\[ (\neg P(x, y) \lor P(y, x)) \land P(a, b) \land \neg P(b, a) \]

Resolution Refutation
Automatic First-Order Theorem Proving

What’s going on here?
Preprocessing Pipeline
Preprocessing Pipeline

Q(a, b) \land (\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))

Simplifications on Formula Level

\neg P(x, y) \lor P(y, x) \land P(a, b) \land \neg P(b, a)

Negation & Clausification

Simplifications on Clause Level
Preprocessing Pipeline

\[ Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]
Preprocessing Pipeline

$Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a)))$

Simplifications on Formula Level
Preprocessing Pipeline

\[ Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \to (\neg P(a, b) \lor P(b, a))) \]

Simplifications on Formula Level

\[ ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \to (\neg P(a, b) \lor P(b, a))) \]
Preprocessing Pipeline

\[ Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]

Simplifications on Formula Level

\[ ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]

Negation & Clasification

\[ (\neg P(a, b) \lor P(b, a)) \land P(a, b) \land \neg P(b, a) \]}
Preprocessing Pipeline

\[ Q(a, b) \land ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]

Simplifications on Formula Level

\[ ((\forall x \forall y P(x, y) \leftrightarrow P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]

Negation & Clausification

\[ (P(x, y) \lor \neg P(y, x)) \land (\neg P(x, y) \lor P(y, x)) \land P(a, b) \land \neg P(b, a) \]
Preprocessing Pipeline

\[ Q(a, b) \land \left( (\forall x \forall y P(x, y) \iff P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a)) \right) \]

Simplifications on Formula Level

\[ (\forall x \forall y P(x, y) \iff P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a)) \]

Negation & Clausification

\[ (P(x, y) \lor \neg P(y, x)) \land (\neg P(x, y) \lor P(y, x)) \land P(a, b) \land \neg P(b, a) \]

Simplifications on Clause Level
Preprocessing Pipeline

\[ Q(a, b) \land ((\forall x \forall y P(x, y) \iff P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]

Simplifications on Formula Level

\[ ((\forall x \forall y P(x, y) \iff P(y, x)) \rightarrow (\neg P(a, b) \lor P(b, a))) \]

Negation & Clausification

\[ (P(x, y) \lor \neg P(y, x)) \land (\neg P(x, y) \lor P(y, x)) \land P(a, b) \land \neg P(b, a) \]

Simplifications on Clause Level

\[ (\neg P(x, y) \lor P(y, x)) \land P(a, b) \land \neg P(b, a) \]
Preprocessing Pipeline

- Topic of this talk: Simplifications on the clause level.

\[(P(x, y) \lor \neg P(y, x)) \land (\neg P(x, y) \lor P(y, x)) \land P(a, b) \land \neg P(b, a)\]

Simplifications on Clause Level

\[(\neg P(x, y) \lor P(y, x)) \land P(a, b) \land \neg P(b, a)\]
Clause-elimination techniques remove redundant clauses.
Clause-Elimination Techniques in Theory

- Clause-elimination techniques remove redundant clauses.
- A clause is redundant if its removal preserves unsatisfiability.
Clause-Elimination Techniques in Theory

- Clause-elimination techniques remove redundant clauses.
- A clause is redundant if its removal preserves unsatisfiability.

\[ \text{If we can refute the formula before removing the clause, we can still refute it afterwards.} \]
Clause-Elimination Techniques in Theory

- Clause-elimination techniques remove redundant clauses.
- A clause is redundant if its removal preserves unsatisfiability.
  
  ⇨ If we can refute the formula before removing the clause, we can still refute it afterwards.

**Definition**

A clause $C$ is redundant with respect to a formula $F$ if $F$ and $F \setminus \{C\}$ are equisatisfiable.
Clause-Elimination Techniques in Theory

- Clause-elimination techniques remove redundant clauses.
- A clause is redundant if its removal preserves unsatisfiability.
  - If we can refute the formula before removing the clause, we can still refute it afterwards.

**Definition**

A clause $C$ is redundant with respect to a formula $F$ if $F$ and $F \setminus \{C\}$ are equisatisfiable.

- **Remark:** Redundant clauses need not be implied!
Problem: Checking if a clause is redundant is undecidable.
Problem: Checking if a clause is redundant is undecidable.

Define efficiently decidable criteria that ensure redundancy.
Clause-Elimination Techniques in Practice

- Problem: Checking if a clause is redundant is undecidable.
- Define efficiently decidable criteria that ensure redundancy.
- Examples:
Problem: Checking if a clause is redundant is undecidable.

Define efficiently decidable criteria that ensure redundancy.

Examples: A clause $C$ is redundant if . . .
Problem: Checking if a clause is redundant is undecidable.

Define efficiently decidable criteria that ensure redundancy.

Examples: A clause $C$ is redundant if . . .

- . . . it contains two complementary literals $L$ and $\neg L$. (Tautology)
Clause-Elimination Techniques in Practice

- Problem: Checking if a clause is redundant is undecidable.
- Define efficiently decidable criteria that ensure redundancy.
- Examples: A clause $C$ is redundant if . . .
  - ... it contains two complementary literals $L$ and $\neg L$. (Tautology)
  - ... all resolvents upon one of its literals are tautologies. (Blocked clause)
Problem: Checking if a clause is redundant is undecidable.

Define efficiently decidable criteria that ensure redundancy.

Examples: A clause $C$ is redundant if . . .

- . . . it contains two complementary literals $L$ and $\neg L$. (Tautology)
- . . . all resolvents upon one of its literals are tautologies. (Blocked clause)
- . . . there exist another clause $D$ and a substitution $\lambda$ such that $D\lambda \subseteq C$. (Subsumed clause)
Problem: Checking if a clause is redundant is undecidable.

Define efficiently decidable criteria that ensure redundancy.

Examples: A clause $C$ is redundant if . . .

- . . . it contains two complementary literals $L$ and $\neg L$. (Tautology)
- . . . all resolvents upon one of its literals are tautologies. (Blocked clause)

- . . . there exist another clause $D$ and a substitution $\lambda$ such that $D\lambda \subseteq C$. (Subsumed clause)
- . . .
Clause-elimination is successfully used in SAT and QSAT solving:

- *Effective Preprocessing in SAT Through Variable and Clause Elimination* (Eén and Biere, SAT, 2005)
- *Clause Elimination for SAT and QSAT* (Heule et al., JAIR, 2010)
- *Covered Clause Elimination* (Heule et al., LPAR, 2010)
- *Blocked Clause Elimination* (Järvisalo et al., TACAS, 2010)
- *Enhancing Search-Based QBF solving by Dynamic Blocked Clause Elimination* (Lonsing et al., LPAR, 2015)
- ...

Blocked-clause elimination can speed up first-order provers:

- *Blocked Clauses in First-Order Logic* (Kiesl, Suda, Seidl, Tompits, and Biere, LPAR, 2017)
Clause-Elimination Techniques: Success Stories

- Clause-elimination is successfully used in SAT and QSAT solving:
  - *Effective Preprocessing in SAT Through Variable and Clause Elimination* (Eén and Biere, SAT, 2005)
  - *Clause Elimination for SAT and QSAT* (Heule et al., JAIR, 2010)
  - *Covered Clause Elimination* (Heule et al., LPAR, 2010)
  - *Blocked Clause Elimination* (Järvisalo et al., TACAS, 2010)
  - *Enhancing Search-Based QBF solving by Dynamic Blocked Clause Elimination* (Lonsing et al., LPAR, 2015)
  - ...

- Blocked-clause elimination can speed up first-order provers:
  - *Blocked Clauses in First-Order Logic* (Kiesl, Suda, Seidl, Tompits, and Biere, LPAR, 2017)
(Some) Types of Redundant Clauses in SAT Solving

- Asymmetric Tautologies
- Covered Clauses
- Resolution Asymmetric Tautologies
- Resolution Subsumed Clauses
- Blocked Clauses
- Tautologies
- Asymmetric Blocked Clauses
- Subsumed Clauses
- Asymmetric Covered Clauses
(Some) Types of Redundant Clauses in SAT Solving

- Asymmetric Tautologies
- Covered Clauses
- Resolution Asymmetric Tautologies
- Resolution Subsumed Clauses
- Asymmetric Blocked Clauses
- Asymmetric Covered Clauses

- Not available in first-order logic before!
Some Types of Redundant Clauses in SAT Solving

- Asymmetric Tautologies
- Covered Clauses
- Resolution Asymmetric Tautologies
- Resolution Subsumed Clauses
- Asymmetric Blocked Clauses
- Asymmetric Covered Clauses

- Not available in first-order logic before!

- We lifted them.
A clause $C$ is blocked in a formula $F$ if all resolvents upon one of its literals are tautologies.
A clause \( C \) is \textit{blocked} in a formula \( F \) if all resolvents upon one of its literals are tautologies.

\[
P \lor Q \lor R
\]

\[
P \lor Q \lor \neg Q
\]

\[
\neg S \lor P \lor Q
\]

\[
\neg R \lor \neg Q
\]

\[
\neg R \lor \neg P
\]

\[
\neg T \lor S \lor Q
\]
A clause $C$ is blocked in a formula $F$ if all resolvents upon one of its literals are tautologies.
Example: Blocked Clauses in Propositional Logic

- A clause $C$ is blocked in a formula $F$ if all resolvents upon one of its literals are tautologies.

$P \lor Q \lor R$

\[
\begin{align*}
\neg S \lor P \lor Q \\
\neg R \lor \neg Q \\
\neg R \lor \neg P \\
\neg T \lor S \lor Q
\end{align*}
\]

$P \lor Q \lor \neg Q$

$P \lor Q \lor \neg P$

$P \lor Q \lor R$ is a blocked clause.
Blocked Clauses in First-Order Logic

- Blocked clauses for first-order logic can be defined in a similar way as in propositional logic.
Blocked Clauses in First-Order Logic

- Blocked clauses for first-order logic can be defined in a similar way as in propositional logic.
- Proving redundancy of blocked clauses in propositional logic is (relatively) simple.
Blocked Clauses in First-Order Logic

- Blocked clauses for first-order logic can be defined in a similar way as in propositional logic.

- Proving redundancy of blocked clauses in propositional logic is (relatively) simple.

- Proving redundancy of blocked clauses in first-order logic requires heavy machinery.
  - Herbrand’s theorem,
  - factorization,
  - non-trivial properties of (most general) unification, etc.

- Required: A general theorem that helps us prove redundancy of several types of clauses in a unified way.
The Principle of Implication Modulo Resolution

To prove correctness of the new techniques, we introduced the principle of implication modulo resolution.

- A first-order variant of quantified implied outer resolvents (Heule, Seidl, and Biere, JAR, 2017).
To prove correctness of the new techniques, we introduced the principle of implication modulo resolution.

- A first-order variant of quantified implied outer resolvents (Heule, Seidl, and Biere, JAR, 2017).

**Definition**

A clause $C$ is **implied modulo resolution** by a formula $F$ if all resolvents of $C$ upon one of its literals are implied by $F \setminus \{C\}$.
The Principle of Implication Modulo Resolution

To prove correctness of the new techniques, we introduced the principle of implication modulo resolution.

- A first-order variant of quantified implied outer resolvents (Heule, Seidl, and Biere, JAR, 2017).

**Definition**

A clause $C$ is implied modulo resolution by a formula $F$ if all resolvents of $C$ upon one of its literals are implied by $F \setminus \{C\}$.

$F \setminus \{C\}$ might not imply $C$, but it implies all conclusions derived from $C$ via resolution upon one of its literals.
To prove correctness of the new techniques, we introduced the principle of implication modulo resolution.

- A first-order variant of quantified implied outer resolvents (Heule, Seidl, and Biere, JAR, 2017).

**Definition**

A clause $C$ is implied modulo resolution by a formula $F$ if all resolvents of $C$ upon one of its literals are implied by $F \setminus \{C\}$.

**Theorem (Main Result)**

If a formula $F$ implies a clause $C$ modulo resolution, then $C$ is redundant with respect to $F$. 
### Implication Modulo Resolution: Examples

<table>
<thead>
<tr>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>A clause $C$ is <strong>implied modulo resolution</strong> by a formula $F$ if all resolvents of $C$ upon one of its literals are implied by $F \setminus {C}$.</td>
</tr>
</tbody>
</table>

- **Blocked clauses** are implied modulo resolution:
  - Every resolvent is a tautology $\Rightarrow$ every resolvent is implied.
Definition

A clause \( C \) is implied modulo resolution by a formula \( F \) if all resolvents of \( C \) upon one of its literals are implied by \( F \setminus \{ C \} \).

- **Blocked clauses** are implied modulo resolution:
  - Every resolvent is a tautology \( \Rightarrow \) every resolvent is implied.

- **Clauses with pure literals**:
  - Pure literals are literals whose predicate symbol occurs in only one polarity in \( F \).
Implication Modulo Resolution: Examples

Definition

A clause $C$ is implied modulo resolution by a formula $F$ if all resolvents of $C$ upon one of its literals are implied by $F \setminus \{C\}$.

- **Blocked clauses** are implied modulo resolution:
  - Every resolvent is a tautology $\Rightarrow$ every resolvent is implied.

- **Clauses with pure literals**:
  - Pure literals are literals whose predicate symbol occurs in only one polarity in $F$.
  - There are no resolvents upon a pure literal $\Rightarrow$ every resolvent is implied.
Implication Modulo Resolution: Examples

Definition

A clause \( C \) is implied modulo resolution by a formula \( F \) if all resolvents of \( C \) upon one of its literals are implied by \( F \setminus \{ C \} \).

- **Blocked clauses** are implied modulo resolution:
  - Every resolvent is a tautology \( \Rightarrow \) every resolvent is implied.

- **Clauses with pure literals**:
  - Pure literals are literals whose predicate symbol occurs in only one polarity in \( F \).
  - There are no resolvents upon a pure literal \( \Rightarrow \) every resolvent is implied.

- **Resolution asymmetric tautologies (RATs)**, resolution-subsumed clauses, etc.
Confluent Clause-Elimination Techniques

- **Confluence**: Eliminating clauses in a different order yields the same result.
Confluent Clause-Elimination Techniques

- **Confluence**: Eliminating clauses in a different order yields the same result.

- **Example** (boxes are clauses, orange clauses are redundant according to some redundancy notion):

```
1 3 4 2 5

¯
```
Confluent Clause-Elimination Techniques

- **Confluence**: Eliminating clauses in a different order yields the same result.

- **Example** (boxes are clauses, orange clauses are redundant according to some redundancy notion):

```
1

[Clauses]
```

We don't need to bother about the elimination order.
Confluent Clause-Elimination Techniques

- **Confluence**: Eliminating clauses in a different order yields the same result.

- **Example** (boxes are clauses, *orange* clauses are redundant according to some redundancy notion):

```
1

2
```

...
Confluent Clause-Elimination Techniques

- **Confluence**: Eliminating clauses in a different order yields the same result.
- **Example** (boxes are clauses, orange clauses are redundant according to some redundancy notion):

```
1
2
3
4
5
```

1
2
3
4
5
**Confluent Clause-Elimination Techniques**

- **Confluence**: Eliminating clauses in a different order yields the same result.
- **Example** (boxes are clauses, orange clauses are redundant according to some redundancy notion):

```
1  3  4  2  5
```

We don't need to bother about the elimination order.
Confluent Clause-Elimination Techniques

- **Confluence**: Eliminating clauses in a different order yields the same result.

- **Example** (boxes are clauses, orange clauses are redundant according to some redundancy notion):

```
1 3 4 2 5
```

We don't need to bother about the elimination order.
Confluent Clause-Elimination Techniques

- **Confluence**: Eliminating clauses in a different order yields the same result.

- **Example** (boxes are clauses, orange clauses are redundant according to some redundancy notion):

```
   1  3  4  2  5

   1  3  4  2  5
```

```
   1  3  4  2  5
```

```
   1  3  4  2  5
```
Confluent Clause-Elimination Techniques

- **Confluence**: Eliminating clauses in a different order yields the same result.
- **Example** (boxes are clauses, orange clauses are redundant according to some redundancy notion):

```
  1  3  4  2  5
[ ] [ ] [ ] [ ] [ ]
```

```
  1  5  2  3  4
[ ] [ ] [ ] [ ] [ ]
```

We don't need to bother about the elimination order.
**Confluent Clause-Elimination Techniques**

- **Confluence**: Eliminating clauses in a different order yields the same result.
- **Example** (boxes are clauses, orange clauses are redundant according to some redundancy notion):

```
1  3  4  2  5

1     2
2  1
```
Confluent Clause-Elimination Techniques

- **Confluence**: Eliminating clauses in a different order yields the same result.

- **Example** (boxes are clauses, orange clauses are redundant according to some redundancy notion):

```plaintext
1 3 4 2 5

3 2 1
```
Confluent Clause-Elimination Techniques

**Confluence**: Eliminating clauses in a different order yields the same result.

**Example** (boxes are clauses, orange clauses are redundant according to some redundancy notion):

1  3  4  5

3  4  2  1
Confluent Clause-Elimination Techniques

- **Confluence**: Eliminating clauses in a different order yields the same result.

- **Example** (boxes are clauses, orange clauses are redundant according to some redundancy notion):

```
1  3  4  2  5

3  4  2  1  5
```
**Confluent Clause-Elimination Techniques**

- **Confluence**: Eliminating clauses in a different order yields the same result.

- **Example** (boxes are clauses, orange clauses are redundant according to some redundancy notion):

  ![Diagram of clauses and elimination order](image)

  We don’t need to bother about the elimination order.
<table>
<thead>
<tr>
<th>Technique</th>
<th>Confluent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocked-Clause Elimination</td>
<td>✓</td>
</tr>
<tr>
<td>Covered-Clause Elimination</td>
<td>✓</td>
</tr>
<tr>
<td>Asymmetric-Tautology Elimination</td>
<td>✗</td>
</tr>
<tr>
<td>Resolution-Asymmetric-Tautology Elimination</td>
<td>✗</td>
</tr>
<tr>
<td>Resolution-Subsumed-Clause Elimination</td>
<td>✗</td>
</tr>
</tbody>
</table>
## Confluence Results

<table>
<thead>
<tr>
<th>Technique</th>
<th>Confluent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocked-Clause Elimination</td>
<td>✔️</td>
</tr>
<tr>
<td>Covered-Clause Elimination</td>
<td>✔️</td>
</tr>
<tr>
<td>Asymmetric-Tautology Elimination</td>
<td>❌</td>
</tr>
<tr>
<td>Resolution-Asymmetric-Tautology Elimination</td>
<td>❌</td>
</tr>
<tr>
<td>Resolution-Subsumed-Clause Elimination</td>
<td>❌</td>
</tr>
<tr>
<td>Covered-Literal Addition</td>
<td>✔️</td>
</tr>
<tr>
<td>Asymmetric-Literal Addition</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Future Work

- Implication modulo resolution for first-order logic with equality.
  - Lift all preprocessing techniques to first-order logic with equality.
- Implement and evaluate a preprocessor with our techniques.
  - Blocked-clause elimination is already implemented.
  - Preprocessor is based on Vampire.
Summary

- Lifted clause-elimination techniques from SAT to first-order logic.
- Correctness proofs via principle of implication modulo resolution.
- Confluence analysis.
- Not in this talk but in the paper:
  - Short correctness proof for predicate elimination (Khasidashvili and Korovin, SAT, 2016) via implication modulo resolution.