The Rewriting Approach to Decision Procedures

Alessandro Armando

Artificial Intelligence Laboratory (Al-Lab)
DIST, University of Genova
Genova

Security & Trust Research Unit
FBK-IRST
Trento
**Objective**: Decision procedures for automated verification

**Desiderata**: Fast, expressive, easy to use, extend, integrate, prove sound and complete

**Issues**:
- Soundness and completeness proofs: usually involved (e.g. based on model theoretic arguments) and ad hoc
- Combination of theories: usually done by combining procedures: often complex
- Implementation: usually from scratch: correctness, duplication of work, integration with other reasoning modules, ...
**Motivation**

- **Objective**: Decision procedures for automated verification

- **Desiderata**: Fast, expressive, easy to use, extend, integrate, prove sound and complete

- **Issues**:
  - Soundness and completeness proofs: usually involved (e.g. based on model theoretic arguments) and ad hoc
  - Combination of theories: usually done by combining procedures: often complex.
  - Implementation: usually from scratch: correctness, duplication of work, integration with other reasoning modules, ...
**Motivation**

- **Objective:** Decision procedures for automated verification
- **Desiderata:** Fast, expressive, easy to use, extend, integrate, prove sound and complete
- **Issues:**
  - Soundness and completeness proofs: usually involved (e.g. based on model theoretic arguments) and ad hoc
  - Combination of theories: usually done by combining procedures: often complex.
  - Implementation: usually from scratch: correctness, duplication of work, integration with other reasoning modules, ...
Motivation

- **Objective**: Decision procedures for automated verification
- **Desiderata**: Fast, expressive, easy to use, extend, integrate, prove sound and complete
- **Issues**:
  - Soundness and completeness proofs: usually involved (e.g. based on model theoretic arguments) and ad hoc
  - Combination of theories: usually done by combining procedures: often complex.
  - Implementation: usually from scratch: correctness, duplication of work, integration with other reasoning modules, ...
Motivation

- **Objective**: Decision procedures for automated verification

- **Desiderata**: Fast, expressive, easy to use, extend, integrate, prove sound and complete

- **Issues**:
  - Soundness and completeness proofs: usually involved (e.g. based on model theoretic arguments) and ad hoc
  - Combination of theories: usually done by combining procedures: often complex.
  - Implementation: usually from scratch: correctness, duplication of work, integration with other reasoning modules, ...
**Objective**: Decision procedures for automated verification

**Desiderata**: Fast, expressive, easy to use, extend, integrate, prove sound and complete

**Issues**:

- Soundness and completeness proofs: usually involved (e.g. based on model theoretic arguments) and *ad hoc*

- Combination of theories: usually done by combining procedures: often complex.

- Implementation: usually from scratch: correctness, duplication of work, integration with other reasoning modules, ...
“Little” engines and “big” engines of proof

- “Little” engines, e.g., validity checkers for specific theories
  - Built-in (decidable) theory, quantifier-free conjecture

- “Big” engines, e.g., general first-order theorem provers
  - Any first-order (semi-decidable) theory, any conjecture

- Not an issue of size (e.g., lines of code) of systems!

- Continuity: e.g.,
  - “big” engines may have theories built-in and
  - “little” engines may support theory-independent reasoning component (e.g. for rewriting, dealing with quantifiers, ...)

- **Challenge**: can big engines be (effectively) used as small engines?
“Little” engines and “big” engines of proof

- “Little” engines, e.g., validity checkers for specific theories
  Built-in (decidable) theory, quantifier-free conjecture
- “Big” engines, e.g., general first-order theorem provers
  Any first-order (semi-decidable) theory, any conjecture

- Not an issue of size (e.g., lines of code) of systems!
- Continuity: e.g.,
  - “big” engines may have theories built-in and
  - “little” engines may support theory-independent reasoning component (e.g. for rewriting, dealing with quantifiers, ...)

- **Challenge**: can big engines be (effectively) used as small engines?
“Little” engines and “big” engines of proof

- “Little” engines, e.g., validity checkers for specific theories
  Built-in (decidable) theory, quantifier-free conjecture

- “Big” engines, e.g., general first-order theorem provers
  Any first-order (semi-decidable) theory, any conjecture

- Not an issue of size (e.g., lines of code) of systems!

- Continuity: e.g.,
  - “big” engines may have theories built-in and
  - “little” engines may support theory-independent reasoning component (e.g. for rewriting, dealing with quantifiers, ...)

- **Challenge**: can big engines be (effectively) used as small engines?
“Little” engines and “big” engines of proof

- “Little” engines, e.g., validity checkers for specific theories
  Built-in (decidable) theory, quantifier-free conjecture

- “Big” engines, e.g., general first-order theorem provers
  Any first-order (semi-decidable) theory, any conjecture

- Not an issue of size (e.g., lines of code) of systems!

- Continuity: e.g.,
  - “big” engines may have theories built-in and
  - “little” engines may support theory-independent reasoning component (e.g. for rewriting, dealing with quantifiers, ...)

- Challenge: can big engines be (effectively) used as small engines?
“Little” engines and “big” engines of proof

- “Little” engines, e.g., validity checkers for specific theories
  - Built-in (decidable) theory, quantifier-free conjecture

- “Big” engines, e.g., general first-order theorem provers
  - Any first-order (semi-decidable) theory, any conjecture

- Not an issue of size (e.g., lines of code) of systems!

- Continuity: e.g.,
  - “big” engines may have theories built-in and
  - “little” engines may support theory-independent reasoning component (e.g. for rewriting, dealing with quantifiers, ...)

- **Challenge**: can big engines be (effectively) used as small engines?
Soundness and completeness proof: already given for first-order inference system

Combination of theories: give union of presentations as input to the prover

Implementation: take and use first-order provers off-the-shelf

Proof generation: it comes for free

Counterexample generation: can be extracted from saturated set of clauses
Soundness and completeness proof: already given for first-order inference system

Combination of theories: give union of presentations as input to the prover

Implementation: take and use first-order provers off-the-shelf

Proof generation: it comes for free

Counterexample generation: can be extracted from saturated set of clauses
From a big-engine perspective

- **Soundness and completeness proof**: already given for first-order inference system
- **Combination of theories**: give union of presentations as input to the prover
- **Implementation**: take and use first-order provers off-the-shelf
  - **Proof generation**: it comes for free
  - **Counterexample generation**: can be extracted from saturated set of clauses
From a big-engine perspective

- **Soundness and completeness proof**: already given for first-order inference system
- **Combination of theories**: give union of presentations as input to the prover
- **Implementation**: take and use first-order provers off-the-shelf
- **Proof generation**: it comes for free
- **Counterexample generation**: can be extracted from saturated set of clauses
From a big-engine perspective

- **Soundness and completeness proof**: already given for first-order inference system
- **Combination of theories**: give union of presentations as input to the prover
- **Implementation**: take and use first-order provers off-the-shelf
- **Proof generation**: it comes for free
- **Counterexample generation**: can be extracted from saturated set of clauses
Roadmap

1 Motivation

2 Rewrite-based satisfiability
   - A rewrite-based methodology for $T$-satisfiability
   - A modularity theorem for combination of theories

3 Experimental appraisal
   - Comparison of E with CVC and CVC Lite
   - Synthetic benchmarks (valid and invalid): evaluate scalability
   - “Real-world” problems
1 Motivation

2 Rewrite-based satisfiability
   - A rewrite-based methodology for $T$-satisfiability
   - A modularity theorem for combination of theories

3 Experimental appraisal
   - Comparison of E with CVC and CVC Lite
   - Synthetic benchmarks (valid and invalid): evaluate scalability
   - “Real-world” problems
Trick: flattening

- Flatten terms by introducing “fresh” constants, e.g.

\[
\{f(f(f(a))) = b\} \leadsto \{f(a) = c_1, f(f(c_1)) = b\} \\
\leadsto \{f(a) = c_1, f(c_1) = c_2, f(c_2) = b\} \\
\{g(h(d))) \neq a\} \leadsto \{h(a) = c_1, g(c_1) \neq a\} \\
\leadsto \{h(a) = c_1, g(c_1) = c_2, c_2 \neq a\}
\]

- **Exercise**: show that this transformation preserves satisfiability
- The number of constants introduced is equal to the number of sub-terms occurring in the input set of literals

- **Key observation**: after flattening, literals are “close” to literals built out of constants only... we need to take care of substitution in a very simple way...
A (extended) set of inference rules for CSAT($T_{UF}$)

**CP**

\[
\frac{c = c' \quad c = d}{c' = d}
\]

if $c \succ c'$ and $c \succ d$

**Cong$_1$**

\[
\frac{c_j = c'_j \quad f(c_1, \ldots, c_j, \ldots, c_n) = c_{n+1}}{f(c_1, \ldots, c'_j, \ldots, c_n) = c_{n+1}}
\]

if $c_j \succ c'_j$

**Cong$_2$**

\[
\frac{f(c_1, \ldots, c_n) = c'_{n+1} \quad f(c_1, \ldots, c_n) = c_{n+1}}{c_{n+1} = c'_{n+1}}
\]

if $c_{n+1} \succ c'_{n+1}$

**DH**

\[
\frac{c = c' \quad c \neq d}{c' \neq d}
\]

if $c \succ c'$ and $c \succ d$

**UN**

\[
\frac{c \neq c}{\Box}
\]

Notice that we *only need to compare constants!*
A (extended) set of inference rules for CSAT($T_{UF}$)

**CP**
\[
\frac{c = c'}{c' = d} \quad \frac{c = d}{c' = d}
\]
if $c \succ c'$ and $c \succ d$

**Cong$_1$**
\[
\frac{c_j = c'_j}{f(c_1, ..., c_j, ..., c_n) = c_{n+1}} \quad \frac{f(c_1, ..., c_j, ..., c_n) = c_{n+1}}{f(c_1, ..., c_j, ..., c_n) = c_{n+1}}
\]
if $c_j \succ c'_j$

**Cong$_2$**
\[
\frac{f(c_1, ..., c_n) = c'_{n+1}}{c_{n+1} = c'_{n+1}} \quad \frac{f(c_1, ..., c_n) = c_{n+1}}{c_{n+1} = c'_{n+1}}
\]
if $c_{n+1} \succ c'_{n+1}$

**DH**
\[
\frac{c = c'}{c' \neq d} \quad \frac{c \neq d}{c' \neq d}
\]
if $c \succ c'$ and $c \succ d$

**UN**
\[
\frac{c \neq c}{\Box}
\]

Notice that we **only need to compare constants**!
A decision procedure for CSAT(UF): summary

1. Flatten literals
2. Exhaustive application of the rules in the previous slide
3. If is derived, then return unsatisfiable
4. Otherwise, return satisfiable

In the worst case, the complexity is quadratic in the number of sub-terms occurring in the input set of UF literals

Exercise: explain why.

You can do better (i.e. $O(n \log n)$) by using a dynamic ordering over constants...

⇒ [Bachmair, Tiwari, and Vigneron] for more on this point.
Outline

1. The constraint satisfiability problem for $T_{UF}$

2. Deciding the constraint satisfiability problem for $T_{UF}$
   - Equality as a graph
   - Convexity
   - Rewriting techniques for $T_{UF}$

3. Superposition for extensions of $T_{UF}$
   - The Superposition Calculus
   - A catalogue of theories
   - Limitations of the rewriting approach

4. References
Can we extend the approach to other theories?

• Yes, but using more general concepts:
  ▶ rewriting on arbitrary terms (not only constants)
  ▶ considering arbitrary clauses since many interesting theories are axiomatized by formulae which are more complex than simple equalities or disequalities, e.g. the theory of arrays:

\[
\begin{align*}
  \text{read(write}(A, l, E), l) &= E \\
  l &= J \lor \text{read(write}(A, l, E), J) = \text{read}(A, J)
\end{align*}
\]

where \(A, l, J, E\) are implicitly universally quantified variables.
Our goal

- **Given**
  - a presentation of a theory $T$ extending UF
    (Notice that $T$ is *not restricted* to equations!)

- **We want to derive**
  - a satisfiability decision procedure capable of establishing whether $S$ is $T$-satisfiable, i.e. $S \cup T$ is satisfiable (where $S$ is a set of *ground literals*)
Our approach to the problem

- Based on the **rewriting approach**
  - uniform and simple
  - efficient alternative to the congruence closure approach
- **Tune** a general (off-the-shelf) *refutation complete superposition inference system* (from [Nieuwenhuis and Rubio]) in order to obtain *termination*

on some interesting theories
An overview of a rewriting approach

Our methodology consists of two steps: given an axiomatization $Ax(T)$ of a theory $T$ and a constraint $S$ in $T$

1. flatten all the literals in $S$ (by extending the signature introducing “fresh” constants)
   - recall that this preserves satisfiability

2. exhaustively apply the rules of the superposition calculus
## Expansion rules of \( \mathcal{SP} \) (I)

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sup.</strong></td>
<td>( \Gamma \rightarrow \Delta, l[u'] = r \quad \Pi \rightarrow \Sigma, u = v ) [\frac{\Gamma, \Pi \rightarrow \Delta, \Sigma, l[v] = r}{\Gamma, \Pi \rightarrow \Delta, \Sigma, l[u'] = r} ] ( u \not\preceq v, l[u'] \not\preceq r, * )</td>
<td></td>
</tr>
<tr>
<td><strong>Par.</strong></td>
<td>( \Gamma, l[u'] = r \rightarrow \Delta \quad \Pi \rightarrow \Sigma, u = v ) [\frac{l[v] = r, \Gamma, \Pi \rightarrow \Delta, \Sigma}{l[v] = r, \Gamma, \Pi \rightarrow \Delta, \Sigma} ] ( u \not\preceq v, l[u'] \not\preceq r, * )</td>
<td></td>
</tr>
<tr>
<td><strong>Ref.</strong></td>
<td>( \Gamma, u' = u \rightarrow \Delta ) [\frac{\Gamma, u' = u \rightarrow \Delta}{\Gamma \rightarrow \Delta} ] ( (u' = u) \not\preceq (\Gamma \cup \Delta) )</td>
<td></td>
</tr>
<tr>
<td><strong>Fac.</strong></td>
<td>( \Gamma \rightarrow \Delta, u = v, u' = v' ) [\frac{\Gamma, v = v' \rightarrow \Delta, u = v'}{\Gamma, v = v' \rightarrow \Delta, u = v'} ] ( u \not\preceq v, u \not\preceq \Gamma, (u = v) \not\preceq {u' = v'} \cup \Delta )</td>
<td></td>
</tr>
</tbody>
</table>

\* (\( u = v \) \( \not\preceq (\Pi \cup \Sigma) \), (\( l[u'] = r \) \( \not\preceq (\Gamma \cup \Delta) \) 

\*\* \( \sigma = \text{mgu}(u, u') \) implicitly applied to consequents and conditions
## Contraction rules of $\mathcal{SP}$ (II)

<table>
<thead>
<tr>
<th>Name</th>
<th>Rule</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subsumption</strong></td>
<td>$S \cup {C, C'}$</td>
<td>for some $\theta$, $\theta(C) \subseteq C'$, and for no $\rho$, $\rho(C') = C$</td>
</tr>
<tr>
<td>Simplification</td>
<td>$S \cup {C[\theta(l)], l = r}$</td>
<td>$\theta(l) \succ \theta(r)$, $C[\theta(l)] \succ (\theta(l) = \theta(r))$</td>
</tr>
<tr>
<td>Deletion</td>
<td>$S \cup {\Gamma \rightarrow \Delta, t = t}$</td>
<td>$S$</td>
</tr>
</tbody>
</table>
Orderings

• Requirement: $f(c_1, \ldots, c_n) \succ c_0$
  for each non-constant symbol $f$ and constant $c_i$ ($i = 0, 1, \ldots, n$)

• [Definition:] $(a = b) \succ (c = d) \text{ iff } \{a, b\} \succ \{c, d\}$
  (where $\succ$ is the multiset extension of $\succ$ on terms)

• multisets of literals are compared by the multiset extension of $\succ$ on literals

• clauses are considered as multisets of literals

• **Intuition:** the ordering $\succ$ is such that only maximal sides of maximal instances of literals are involved in inferences
Refutation Completeness

- The **exhaustive and fair** application of the rules of the superposition calculus allows us to detect unsatisfiability in a finite amount of time!

- Problem: for which theories do we have finite (fair) derivations?
Refutation Completeness

- The **exhaustive** and **fair** application of the rules of the superposition calculus allows us to detect unsatisfiability in a finite amount of time!

- Problem: **for which theories do we have finite (fair) derivations?**
Example: \(\mathcal{SP}\) on lists (I)

- Consider the following (simplified) theory of lists
  \[ Ax(\mathcal{L}) := \{ \text{car}(\text{cons}(X, Y)) = X, \text{cdr}(\text{cons}(X, Y)) = Y \} \]

- Recall that a literal in \(S\) has one of the four possible forms: (a) \(\text{car}(c) = d\), (b) \(\text{cdr}(c) = d\), (c) \(\text{cons}(c_1, c_2) = d\), and (d) \(c \neq d\).

- There are three cases to consider:
  1. inferences between two clauses in \(S\)
  2. inferences between two clauses in \(Ax(\mathcal{L})\)
  3. inferences between a clause in \(Ax(\mathcal{L})\) and a clause in \(S\)
Example: \( \mathcal{SP} \) on lists (II)

- Case 1: inferences between two clauses in \( S \)
  It has already been considered when considering equality only (please, keep in mind this point)
- Case 2: inferences between two clauses in \( Ax(\mathcal{L}) \)
  This is not very interesting since there are no possible inferences between the two axioms in \( Ax(\mathcal{L}) \)
- Case 3: inferences between a clause in \( Ax(\mathcal{L}) \) and a clause in \( S \)
  - a superposition between \( \text{car}(\text{cons}(X, Y)) = X \) and \( \text{cons}(c_1, c_2) = d \) yielding \( \text{car}(d) = c_1 \) and
  - a superposition between \( \text{cdr}(\text{cons}(X, Y)) = Y \) and \( \text{cons}(c_1, c_2) = d \) yielding \( \text{cdr}(d) = c_2 \)
• We are almost done, it is sufficient to notice that
  ▶ only finitely many equalities of the form (a) and (b) can be
generated this way out of a set of clauses built on a finite signature
  ▶ so, we are entitled to conclude that $\mathcal{SP}$ can only generate finitely
  many clauses on set of clauses of the form $Ax(\mathcal{L}) \cup S$
• A decision procedure for the satisfiability problem of $\mathcal{L}$ can be built by
  simply using $\mathcal{SP}$ after flattening the input set of literals
Recall that in the proof of termination of $SP$ on $Ax(L) \cup S$, we have observed that inferences between clauses in $S$ were already considered for the ground case

So, if we consider a signature $\Sigma := \{\text{cons, car, cdr}\} \cup \Sigma_{UF}$, where $\Sigma_{UF}$ is a finite set of function symbols, the proof of termination above continues to hold.

In other words, we are capable of solving the satisfiability problem for $L \cup T_{UF} \cup S$, where $S$ is a set of ground literals built out of the interpreted function symbols cons, car, cdr and arbitrary uninterpreted function symbols.

The above holds for all satisfiability procedure built by the rewriting approach described here.
Rewriting-based dec proc for lists: summary

- Analysis of the possible inferences in $SP$

**Lemma**

Let $S$ be a finite set of flat $\Sigma_L$-literals. The clauses occurring in the saturations of $S \cup Ax(L)$ by $SP$ can only be the empty clause, ground flat literals, or the equalities in $Ax(L)$.

- Termination follows

**Lemma**

Let $S$ be a finite set of flat $\Sigma_L$-literals. All the saturations of $S \cup Ax(L)$ by $SP$ are finite.

- From termination, fairness, and refutation completeness...

**Theorem**

$SP$ is a decision procedure for $L$. 
A rewriting approach: theories of lists

- Theory of uninterpreted functions: $\Sigma_{UF} := \text{finite set of function symbols}$, $Ax(UF) := \emptyset$
- Theory of lists à la Shostak: $\Sigma_{L_{Sh}} := \{\text{cons, car, cdr}\} \cup \Sigma_{UF}$,
  $$Ax(L_{Sh}) := \{\text{car(cons}(X, Y)) = X, \text{cdr(cons}(X, Y)) = Y, \text{cons(car}(X), \text{cdr}(X)) = X\}$$
- Theory of lists à la Nelson-Oppen: $\Sigma_{L_{NO}} := \{\text{cons, car, cdr, atom}\} \cup \Sigma_{UF}$,
  $$Ax(L_{NO}) := \{\text{car(cons}(X, Y)) = X, \text{cdr(cons}(X, Y)) = Y, \neg\text{atom(cons}(X, Y)) \text{atom}(X) \lor \text{cons(car}(X), \text{cdr}(X)) = X\}$$
A rewriting approach: theories of arrays

arrays w/ extensionality: \( \Sigma_{As} := \{\text{rd}, \text{wr}\} \cup \Sigma_{UF} \),

\[
Ax(\mathcal{A}^s) := \left\{ \begin{array}{l}
\text{rd}(\text{wr}(A, I, E), I) = E \\
I = J \lor \text{rd}(\text{wr}(A, I, E), J) = \text{rd}(A, J)
\end{array} \right. \\
Ax(\mathcal{A}^s_e) := Ax(\mathcal{A}^s) \cup \\
\{ \forall A, B. (\forall I. (\text{rd}(A, I) = \text{rd}(B, I)) \implies A = B) \}
\]
A rewriting approach: theories of records

- records w/ extensionality: \( \Sigma_{R^s} := \{ \text{rsel}_i, \text{rst}_i | i = 1, \ldots, n \} \cup \Sigma_{UF} \),

\[
\begin{align*}
Ax(R^s) &:= \left\{ \begin{array}{l}
\text{rsel}_i(\text{rst}_i(X, V)) = V \\
\text{rsel}_j(\text{rst}_i(X, V)) = \text{rsel}_j(X)
\end{array} \right\} \text{ for all } i, 1 \leq i \leq n \\
&\quad \text{rsel}_i(\text{rst}_i(X, V)) = \text{rsel}_j(X) \text{ for all } i, j, 1 \leq i \neq j \leq n
\end{align*}
\]

\[
Ax(R^s_e) := Ax(A^s) \cup \{ \forall X, Y. (\bigwedge_{i=1}^{n} \text{rsel}_i(X) = \text{rsel}_i(Y) \implies X = Y) \}
\]
A rewriting approach: small fragments of Arithmetics

- **Integer Offsets:** $\Sigma_I := \{\text{succ, prec}\} \cup \Sigma_{UF}$,

  
  \[
  Ax(I) := \begin{cases} 
    \text{succ(prec}(X)) = X, \text{prec(succ}(X)) = X, \\
    \text{succ}^i(X) \neq X & \text{for } i > 0 \\
    \text{acyclicity}
  \end{cases}
  \]

  where $\text{succ}^1(x) = \text{succ}(x)$, $\text{succ}^{i+1}(x) = \text{succ}(\text{succ}^i(x))$ for $i \geq 1$

- **Integer Offsets Modulo:** $\Sigma_{I_k} := \{\text{succ, prec}\} \cup \Sigma_{UF}$,

  
  \[
  Ax(I_k) := \begin{cases} 
    \text{succ(prec}(X)) = X, \text{prec(succ}(X)) = X, \\
    \text{succ}^i(X) \neq X & \text{for } 1 \leq i \leq k - 1 \\
    k\text{-acyclicity} \\
    \text{succ}^k(X) = X
  \end{cases}
  \]
Rewrite-based methodology for $T$-satisfiability

- **$T$-satisfiability**: decide satisfiability of set $S$ of ground literals in theory $T$

  Methodology:
  - $T$-reduction: apply inferences (e.g., to remove certain literals or symbols) to get equisatisfiable $T$-reduced problem
  - Flattening: flatten all ground literals (by introducing new constants) to get equisatisfiable $T$-reduced flat problem
  - Ordering selection and termination: select a CSO $\succ$ and prove that any fair $\mathcal{SP}_\succ$-strategy terminates when applied to a $T$-reduced flat problem. We call $T$-good any such $\succ$.

  Everything fully automated except for termination proof


Rewrite-based methodology for $T$-satisfiability

- **$T$-satisfiability**: decide satisfiability of set $S$ of ground literals in theory $T$

- **Methodology**:
  - $T$-reduction: apply inferences (e.g., to remove certain literals or symbols) to get equisatisfiable $T$-reduced problem
  - Flattening: flatten all ground literals (by introducing new constants) to get equisatisfiable $T$-reduced flat problem
  - Ordering selection and termination: select a CSO $\triangleright$ and prove that any fair $SP_{\triangleright}$-strategy terminates when applied to a $T$-reduced flat problem. We call $T$-good any such $\triangleright$.

- Everything fully automated except for termination proof


**Rewrite-based methodology for** \( T \)-satisfiability

- **\( T \)-satisfiability**: decide satisfiability of set \( S \) of ground literals in theory \( T \)

**Methodology**:
- **\( T \)-reduction**: apply inferences (e.g., to remove certain literals or symbols) to get equisatisfiable \( T \)-reduced problem
- **Flattening**: flatten all ground literals (by introducing new constants) to get equisatisfiable \( T \)-reduced flat problem
- **Ordering selection and termination**: select a CSO \( \succ \) and prove that any fair \( SP_{\succ} \)-strategy terminates when applied to a \( T \)-reduced flat problem. We call \( T \)-good any such \( \succ \).

Everything fully automated except for termination proof

---


Rewrite-based methodology for $T$-satisfiability

- **$T$-satisfiability**: decide satisfiability of set $S$ of ground literals in theory $T$

**Methodology:**
- **$T$-reduction**: apply inferences (e.g., to remove certain literals or symbols) to get equisatisfiable $T$-reduced problem
- **Flattening**: flatten all ground literals (by introducing new constants) to get equisatisfiable $T$-reduced flat problem
- **Ordering selection and termination**: select a CSO $\succ$ and prove that any fair $SP_{\succ}$-strategy terminates when applied to a $T$-reduced flat problem. We call $T$-good any such $\succ$.

Everything fully automated except for termination proof


**Rewrite-based methodology for \( T \)-satisfiability**

- \( T \)-satisfiability: decide satisfiability of set \( S \) of ground literals in theory \( T \)
- Methodology:
  - \( T \)-reduction: apply inferences (e.g., to remove certain literals or symbols) to get equisatisfiable \( T \)-reduced problem
  - Flattening: flatten all ground literals (by introducing new constants) to get equisatisfiable \( T \)-reduced flat problem
  - Ordering selection and termination: select a CSO \( \succ \) and prove that any fair \( SPF \succ \)-strategy terminates when applied to a \( T \)-reduced flat problem. We call \( T \)-good any such \( \succ \).

Everything fully automated except for termination proof

Rewrite-based methodology for $T$-satisfiability

- **$T$-satisfiability**: decide satisfiability of set $S$ of ground literals in theory $T$

**Methodology:**
- **$T$-reduction**: apply inferences (e.g., to remove certain literals or symbols) to get equisatisfiable $T$-reduced problem
- **Flattening**: flatten all ground literals (by introducing new constants) to get equisatisfiable $T$-reduced flat problem
- **Ordering selection and termination**: select a CSO $\succ$ and prove that any fair $SP_{\succ}$-strategy terminates when applied to a $T$-reduced flat problem. We call $T$-good any such $\succ$.

- Everything fully automated except for termination proof


Covered theories

- **EUF, lists, arrays** with and without extensionality, **sets** with extensionality [Armando, Ranise, Rusinowitch 2003]

- **Records** with and without extensionality, **integer offsets**, **integer offsets modulo** [Armando, Bonacina, Ranise, Schulz 2005]

- **Theory of inductively defined data structures** [Bonacina, Echenim 2006]
Motivation

Rewrite-based satisfiability
- A rewrite-based methodology for $T$-satisfiability
- A modularity theorem for combination of theories

Experimental appraisal
- Comparison of E with CVC and CVC Lite
- Synthetic benchmarks (valid and invalid): evaluate scalability
- “Real-world” problems
Question: If $SP$ terminates on $T_i$-sat problems, then does it terminate on $T$-sat problems with $T = \bigcup_{i=1}^{n} T_i$?

- $T_i$-reduction and flattening apply as for each theory
- Termination?
A modularity theorem for combination of theories

**Question**: If $SP$ terminates on $T_i$-sat problems, then does it terminate on $T$-sat problems with $T = \bigcup_{i=1}^{n} T_i$?

- $T_i$-reduction and flattening apply as for each theory.
- Termination?
**Question:** If $S\mathcal{P}$ terminates on $\mathcal{T}_i$-sat problems, then does it terminate on $\mathcal{T}$-sat problems with $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$?

- $\mathcal{T}_i$-reduction and flattening apply as for each theory
- Termination?
A modularity theorem

**Theorem** [Armando, Bonacina, Ranise, Schulz 2005]: If

- No shared function symbol (shared constants allowed),
- Variable-inactive presentations $\mathcal{T}_i$, $1 \leq i \leq n$ (no max literal in a ground instance of a clause is instance of an equation $t \simeq x$ where $x \not\in \text{Var}(t)$); it disables *Superpos* from variables across theories.
- Fair $\mathcal{T}_i$-good $\mathcal{SP}_\succ$-strategy is satisfiability procedure for $\mathcal{T}_i$,

then

a fair $\mathcal{T}$-good $\mathcal{SP}_\succ$-strategy is a satisfiability procedure for $\mathcal{T}$.

EUF, arrays (with or without extensionality), records (with or without extensionality), integer offsets and integer offsets modulo, all satisfy these hypotheses.
Motivation

Rewrite-based satisfiability
- A rewrite-based methodology for $T$-satisfiability
- A modularity theorem for combination of theories

Experimental appraisal
- Comparison of E with CVC and CVC Lite
- Synthetic benchmarks (valid and invalid): evaluate scalability
- "Real-world" problems
Experimental setting

- Three systems:
  - The E theorem prover: E 0.82 [Schulz 2002]
  - CVC 1.0a [Stump, Barrett and Dill 2002]
  - CVC Lite Lite 1.1.0 [Barrett and Berezin 2004]

- Two very simple strategies for E: $E(\text{good-lpo})$ and $E(\text{std-kbo})$

- Benchmarks:
  - Parametric synthetic problems
  - “Real world” problems from UCLID

- 3.00GHz 512MB RAM Pentium 4 PC: max 150 sec and 256 MB per run
Arrays: presentation

Theory of arrays with extensionality

\[ \forall x, z, v. \ select(\text{store}(x, z, v), z) \simeq v \]
\[ \forall x, z, w, v. \ (z \not\simeq w \supset \text{select}(\text{store}(x, z, v), w) \simeq \text{select}(x, w)) \]
\[ \forall x, y. \ (\forall z. \text{select}(x, z) \simeq \text{select}(y, z) \supset x \simeq y) \]

where \( x \) and \( y \) have sort \text{ARRAY},
\( z \) has sort \text{INDEX}, and
\( v \) has sort \text{ELEM}. 

Alessandro Armando (U. of Genova & FBK-IRI)
The Rewriting Approach
Arrays: termination of $SP$

$\mathcal{A}$-reduction: eliminate disequalities between arrays by resolution with extensionality.

$\mathcal{A}$-good: $t \succ c$ for all ground compound terms $t$ and constants $c + a \succ e \succ j$, for all constants $a$ of sort ARRAY, $e$ of sort ELEM and $j$ of sort INDEX.

Termination: case analysis of generated clauses (CSO plays key role).

**Theorem:** A fair $\mathcal{A}$-good $SP_\succ$-strategy is a satisfiability procedure for the theories of arrays and arrays with extensionality.
Benchmarks for arrays

Parametric problem instances to assess scalability.

- **STORECOMM**($n$). Encodes the fact that the result of storing a set of elements in different positions within an array is not affected by the relative order of the store operations.

- **SWAP**($n$). Encodes the fact that swapping an element at position $i_1$ with an element at position $i_2$ is equivalent to swapping the element at position $i_2$ with the element at position $i_1$.

- **STOREINV**($n$). Encodes the fact that if the arrays resulting from exchanging elements of an array $a$ with the elements of an array $b$ occurring in the same positions are equal, then $a$ and $b$ must have been equal to begin with.

Both valid and invalid instances generated.
Performances on $\text{STORECOMM}(n)$ instances

valid instances

<table>
<thead>
<tr>
<th>Instance size</th>
<th>Run time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVC</td>
<td>+</td>
</tr>
<tr>
<td>CVC Lite</td>
<td>−−</td>
</tr>
<tr>
<td>E (good-lpo), built-in index type</td>
<td>■</td>
</tr>
<tr>
<td>E (good-lpo), axiomatized indices</td>
<td>○</td>
</tr>
</tbody>
</table>

invalid instances

<table>
<thead>
<tr>
<th>Instance size</th>
<th>Run time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVC</td>
<td>+</td>
</tr>
<tr>
<td>CVC Lite</td>
<td>−−</td>
</tr>
<tr>
<td>E (good-lpo), built-in index type</td>
<td>■</td>
</tr>
<tr>
<td>E (good-lpo), axiomatized indices</td>
<td>○</td>
</tr>
</tbody>
</table>

CVC wins but E better than CVC Lite
Performances on $\text{SWAP}(n)$ instances

CVC and CVC Light win on valid instances, E wins on invalid ones.
Performances on $\text{SWAP}(n)$ instances

CVC and CVC Light win on valid instances, E wins on invalid ones. The situation improves by adding a lemma to E.
Performances on $\text{STOREINV}(n)$ instances

**valid instances**

**invalid instances**

E(std-kbo) does it in nearly constant time!

Not as good for E but run times are minimal
A fragment of the theory of the integers:

s: successor

p: predecessor

Theory of integer offsets

\[ \forall x. \ s(p(x)) \sim x \]

\[ \forall x. \ p(s(x)) \sim x \]

\[ \forall x. \ s^i(x) \not\sim x \quad \text{for } i > 0 \]

Infinitely many acyclicity axioms!
**I-reduction:**
- eliminate p by replacing \( p(c) \simeq d \) with \( c \simeq s(d) \): first two axioms no longer needed.
- Bound the number of acyclicity axioms:
  \( \forall x. s^i(x) \not\simeq x \) for \( 0 < i \leq n + 1 \)
  if there are \( n \) occurrences of \( s \) in the conjecture.

**I-good:** any CSO.

**Termination:** case analysis of generated clauses.

**Theorem:** A fair \( SP_{\succ} \)-strategy is a satisfiability procedure for the theory of integer offsets.
Benchmarks for integer offsets

**IOS**(n): needs combination of theories of arrays and integer offsets.

<table>
<thead>
<tr>
<th>Theories</th>
<th>arrays</th>
<th>ios</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STORECOMM, SWAP, STOREINV</td>
<td>●</td>
</tr>
<tr>
<td>IOS</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

Based on the following observation:

\[
\text{for} (k=1; k<=n; k++) \\
\quad a[i+k] = a[i] + k; \\
\text{for} (k=1; k<=n; k++) \\
\quad a[i+n-k] = a[i+n] - k;
\]

If the execution of either fragment produces the same result in the array \(a\), then \(a[i+n] = a[i] + n\) must hold initially for any value of \(i\), \(k\), \(a\), and \(n\).
CVC and CVC Lite have built-in $\mathcal{LA}(\mathcal{R})$ and $\mathcal{LA}(\mathcal{I})$ respectively!
Records: presentation

Sort $\text{REC}(id_1 : T_1, \ldots, id_n : T_n)$

Theory of records

\[
\forall x, v. \quad \text{rselect}_i(\text{rstore}_i(x, v)) \simeq v \quad 1 \leq i \leq n
\]

\[
\forall x, v. \quad \text{rselect}_j(\text{rstore}_i(x, v)) \simeq \text{rselect}_j(x) \quad 1 \leq i \neq j \leq n
\]

\[
\forall x, y. \quad (\bigwedge_{i=1}^n \text{rselect}_i(x) \simeq \text{rselect}_i(y) \supset x \simeq y)
\]

where $x, y$ have sort $\text{REC}$ and $v$ has sort $T_i$. 
**R**-reduction: eliminate disequalities between records by resolution with extensionality + splitting.

**R-good**: \( t \succ c \) for all ground compound terms \( t \) and constants \( c \).

**Termination**: case analysis of generated clauses (CSO plays key role).

**Theorem**: A fair **R**-good \( SP_\succ \)-strategy is a satisfiability procedure for the theories of records and records with extensionality.
Motivation

Rewrite-based satisfiability
- A rewrite-based methodology for $T$-satisfiability
- A modularity theorem for combination of theories

Experimental appraisal
- Comparison of E with CVC and CVC Lite
- Synthetic benchmarks (valid and invalid): evaluate scalability
- “Real-world” problems
Queues can be defined on top a combination of theories of arrays, records and integer offsets:

<table>
<thead>
<tr>
<th>Theories</th>
<th>arrays</th>
<th>ios</th>
<th>records</th>
</tr>
</thead>
<tbody>
<tr>
<td>STORECOMM,</td>
<td>⬜</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SWAP, STOREINV</td>
<td>⬜</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOS</td>
<td>⬜</td>
<td>⬜</td>
<td></td>
</tr>
<tr>
<td>QUEUE</td>
<td>⬜</td>
<td>⬜</td>
<td>⬜</td>
</tr>
</tbody>
</table>

enqueue(v, x) = rstore_t(rstore_i(x, store(rselect_i(x), rselect_t(x), v))), s(rselect_t(x)))

dequeue(x) = rstore_h(x, s(rselect_h(x)))

first(x) = select(rselect_i(x), rselect_h(x))

last(x) = select(rselect_i(x), p(rselect_t(x)))

reset(x) = rstore_h(x, rselect_t(x))

\(\text{QUEUE}(n)\) expresses the property that if \(q \in \text{QUEUE}\) is obtained from a properly initialized queue by adding elements \(e_0, e_1, \ldots, e_n\), for \(n > 0\), and performing \(0 \leq m \leq n\) dequeue operations then \(\text{first}(q) = e_m\).
Performances on **QUEUE** instances

CVC wins (built-in arithmetic!) but E matches CVC Lite
To reason with indices ranging over the integers mod $k$ ($k > 0$):

**Theory of integer offsets modulo**

\[
\forall x. \ s(p(x)) \simeq x \\
\forall x. \ p(s(x)) \simeq x \\
\forall x. \ s^i(x) \not\simeq x \quad 1 \leq i \leq k - 1 \\
\forall x. \ s^k(x) \simeq x
\]

Finitely many axioms.
**I-reduction**: same as above.

**I-good**: any CSO.

**Termination**: case analysis of generated clauses.

**Theorem**: A fair $SP_\succ$-strategy is a satisfiability procedure for the theory of integer offsets modulo.

Termination also without $I$-reduction.
**Benchmarks for circular queues**

**CIRCULAR_QUEUE**($n, k$) as **QUEUE**($n, k$) but with integer offsets modulo $k$.

<table>
<thead>
<tr>
<th>Theories</th>
<th>arrays</th>
<th>ios</th>
<th>records</th>
<th>mod_ios</th>
</tr>
</thead>
<tbody>
<tr>
<td>STORECOMM, SWAP, STOREINV</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOS</td>
<td>•</td>
<td>•</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>QUEUE</strong></td>
<td>•</td>
<td>•</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td><strong>CIRCULAR_QUEUE</strong></td>
<td>•</td>
<td>•</td>
<td></td>
<td>•</td>
</tr>
</tbody>
</table>
Performances on $\text{CIRCULAR\_QUEUE}(n, k)$ instances

$k = 3$

CVC does not handle integers mod $k$, E clearly wins
Motivation

Rewrite-based satisfiability
- A rewrite-based methodology for $T$-satisfiability
- A modularity theorem for combination of theories

Experimental appraisal
- Comparison of E with CVC and CVC Lite
- Synthetic benchmarks (valid and invalid): evaluate scalability
- “Real-world” problems
“Real-world” problems

- **UCLID** [Bryant, Lahiri, Seshia 2002]: suite of problems
- **haRVey** [Déharbe and Ranise 2003]: extract $T$-sat problems
- over 55,000 proof tasks: integer offsets and equality
- all valid

<table>
<thead>
<tr>
<th>Theories</th>
<th>arrays</th>
<th>ios</th>
<th>records</th>
<th>mod_ios</th>
<th>euf</th>
</tr>
</thead>
<tbody>
<tr>
<td>STORECOMM, SWAP, STOREINV</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOS</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUEUE</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIRCULAR_QUEUE</td>
<td>●</td>
<td>●</td>
<td></td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>UCLID</td>
<td>●</td>
<td></td>
<td></td>
<td></td>
<td>●</td>
</tr>
</tbody>
</table>

Test performance on huge sets of literals.
Run time distribution on UCLID set

E in auto mode

E with optimized strategy found by testing on random sample of 500 problems (less than 1%)
General methodology for rewrite-based $T$-sat procedures and its application to several theories of data structures

Modularity theorem for combination of theories

Experiments: first-order prover
- taken essentially off the shelf and
- conceived for very different search problems

compares surprisingly well with state-of-the-art verification tools