On a rewriting approach to satisfiability procedures: extension, combination of theories and an experimental appraisal

Maria Paola Bonacina

Dipartimento di Informatica
Università degli Studi di Verona

19th of September, 2005

1Joint work with Alessandro Armando (Università degli Studi di Genova), Silvio Ranise (INRIA Lorraine) and Stephan Schulz (Università degli Studi di Verona)
Objective: Decision procedures for application of automated reasoning to verification
Objective: Decision procedures for application of automated reasoning to verification

Desiderata: Fast, expressive, easy to use, extend, integrate, prove sound and complete
Objective: Decision procedures for application of automated reasoning to verification

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

Issues:
- Combination of theories:
  - usually done by combining procedures: complicated? ad hoc?
- Soundness and completeness proof: usually ad hoc
- Implementation: usually from scratch: correctness? integration in different environments? duplicated work?
“Little” engines and “big” engines of proof

▶ “Little” engines, e.g., validity checkers for specific theories
  Built-in theory, quantifier-free conjecture, decidable
“Little” engines and “big” engines of proof

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

- “Big” engines, e.g., general first-order theorem provers
  Any first-order theory, any conjecture, semi-decidable
“Little” engines and “big” engines of proof

- “Little” engines, e.g., validity checkers for specific theories
  Built-in theory, quantifier-free conjecture, decidable
- “Big” engines, e.g., general first-order theorem provers
  Any first-order theory, any conjecture, semi-decidable
- Not an issue of size (e.g., lines of code) of systems!
- Continuity: e.g., “big” engines may have theories built-in
“Little” engines and “big” engines of proof

- “Little” engines, e.g., validity checkers for specific theories
  Built-in theory, quantifier-free conjecture, decidable
- “Big” engines, e.g., general first-order theorem provers
  Any first-order theory, any conjecture, semi-decidable
- Not an issue of size (e.g., lines of code) of systems!
- Continuity: e.g., “big” engines may have theories built-in
- **Challenge**: can we get something good for decision procedures from big engines?
From a big-engine perspective

- *Combination of theories*: give union of presentations as input to prover
From a big-engine perspective

- **Combination of theories**: give union of presentations as input to prover
- **Soundness and completeness proof**: already given for first-order inference system
From a big-engine perspective

- **Combination of theories**: give union of presentations as input to prover
- **Soundness and completeness proof**: already given for first-order inference system
- **Implementation**: take first-order provers off the shelf
- **Proof generation**: already there by default
- **Model generation**: final $T$-sat set (starting point)
From a big-engine perspective

- **Combination of theories**: give union of presentations as input to prover
- **Soundness and completeness proof**: already given for first-order inference system
- **Implementation**: take first-order provers off the shelf
- **Proof generation**: already there by default
- **Model generation**: final $T$-sat set (starting point)
- **How to make it possible?**
Motivation

Rewrite-based satisfiability: new results
   A rewrite-based methodology for $T$-satisfiability
   Theories of data structures
   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

Summary
What kind of theorem prover?

First-order logic with equality

$SP$ inference system: rewrite-based

- *Simplification by equations*: normalize clauses
- *Superposition*: generate clauses

*Complete simplification ordering (CSO) $\triangleright$* on terms, literals and clauses: $SP_{\triangleright}$

*(Fair) $SP_{\triangleright}$-strategy*: $SP_{\triangleright}$ + (fair) search plan
Rewrite-based methodology for \( T \)-satisfiability

- **\( T \)-satisfiability**: decide satisfiability of set \( S \) of ground literals in theory (or combination) \( T \)
Rewrite-based methodology for $T$-satisfiability

- **$T$-satisfiability**: decide satisfiability of set $S$ of ground literals in theory (or combination) $T$

- **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: prove that any fair $\mathcal{SP}_\succ$-strategy terminates when applied to a $T$-reduced flat problem, provided $\succ$ is $T$-good
Rewrite-based methodology for $T$-satisfiability

- **$T$-satisfiability**: decide satisfiability of set $S$ of ground literals in theory (or combination) $T$

- **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**: prove that any fair $SP_{\succ}$-strategy terminates when applied to a $T$-reduced flat problem, provided $\succ$ is $T$-good

- Everything *fully automated* except for termination proof
Covered theories

- \textit{EUF, lists, arrays} with and without extensionality, \textit{sets} with extensionality [Armando, Ranise, Rusinowitch 2003]

- \textit{Records} with and without extensionality, \textit{integer offsets}, \textit{integer offsets modulo} [Armando, Bonacina, Ranise, Schulz 2005]

In experiments: arrays, records, integer offsets, integer offsets modulo, EUF and combinations
Records: presentation

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

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

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

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

where \( x, y \) have sort \( \text{REC} \) and \( v \) has sort \( T_i \).

Extensionality is the third axiom.


**Motivation**

Rewrite-based satisﬁability: new results
Experimental appraisal
Summary

**Outline**

A rewrite-based methodology for $T$-satisﬁability
Theories of data structures
A modularity theorem for combination of theories

Records: termination of $SP$

$R$-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.
A fragment of the theory of the integers:

s: successor
p: predecessor

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

Infinitely many acyclicity axioms!
Integer offsets: termination of $\mathcal{SP}$

$I$-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:

$$\forall x. s^i(x) \not\simeq x \text{ for } 0 < i \leq n + 1$$

if there are $n$ occurrences of $s$.

$I$-good: any CSO.

Termination: case analysis of generated clauses.

Theorem: A fair $\mathcal{SP}_\triangleright$-strategy is a satisfiability procedure for the theory of integer offsets.
To reason with indices ranging over the integers mod $k$ ($k > 0$):

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

Finitely many axioms.
Integer offsets modulo: termination of $S\mathcal{P}$

$I$-reduction: same as above.

$I$-good: any CSO.

Termination: case analysis of generated clauses.

**Theorem:** A fair $S\mathcal{P}_\succ$-strategy is a satisfiability procedure for the theory of integer offsets modulo.

Termination also without $I$-reduction.
A modularity theorem for combination of theories

- **Modularity**: if $SP_{\geq}$-strategy decides $T_i$-sat problems then it decides $T$-sat problems, $T = \bigcup_{i=1}^{n} T_i$
A modularity theorem for combination of theories

- **Modularity**: if $SP_{\geq}$-strategy decides $\mathcal{T}_i$-sat problems then it decides $\mathcal{T}$-sat problems, $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$

- $\mathcal{T}_i$-reduction and flattening apply as for each theory
A modularity theorem for combination of theories

- **Modularity**: if $\mathcal{SP}_{\succ}$-strategy decides $\mathcal{T}_i$-sat problems then it decides $\mathcal{T}$-sat problems, $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$
- $\mathcal{T}_i$-reduction and flattening apply as for each theory
- Termination?
Three simple conditions

- $\succ T$-good, if $T_i$-good for all $i$, $1 \leq i \leq n$
Three simple conditions

- \( T \)-good, if \( T_i \)-good for all \( i, 1 \leq i \leq n \)
- The \( T_i \) do not share function symbols
  
  (Intuition: no superposition from compound terms across theories)
Three simple conditions

- \( \succ T\text{-good} \), if \( T_i\text{-good} \) for all \( i, 1 \leq i \leq n \)

- The \( T_i \) do not share function symbols
  (Intuition: no superposition from compound terms across theories)

- Each \( T_i \) is variable-inactive:
  no maximal literal in a ground instance of a clause is instance of an equation \( t \simeq x \) where \( x \notin Var(t) \)
  (Intuition: no superposition from variables across theories, since for \( t \simeq x \) where \( x \in Var(t), t \succ x \))
A modularity theorem

**Theorem:** if

- No shared function symbol (shared constants allowed),
- Variable-inactive presentations $\mathcal{T}_i$, $1 \leq i \leq n$,
- Fair $\mathcal{T}_i$-good $SP_{\succ}$-strategy is satisfiability procedure for $\mathcal{T}_i$,

then

a fair $\mathcal{T}$-good $SP_{\succ}$-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.
Two remarks on generality

- *Purely equational theories*: no trivial models $\Rightarrow$ variable-inactive
Two remarks on generality

- **Purely equational theories:** no trivial models $\Rightarrow$ variable-inactive

- **First-order theories:** variable-inactive excludes, e.g.,
  \[ a_1 \equiv x \lor \ldots \lor a_n \equiv x, \ a_i \text{ constants (\#)} \]
  Such a clause means not stably-infinite, hence not convex
  under the no trivial models hypothesis:
  if $T_i$ not variable-inactive for (\#), Nelson-Oppen does not apply either.
Experimental setting

- Three systems:
  - The E theorem prover: E 0.82 [Schulz 2002]
  - CVC 1.0a [Stump, Barrett and Dill 2002]
  - CVC Lite 1.1.0 [Barrett and Berezin 2004]
- Generator of pseudo-random instances of synthetic benchmarks
- 3.00GHz 512MB RAM Pentium 4 PC: max 150 sec and 256 MB per run
- Folklore: systems with built-in theories are out of reach for prover with presentation as input...
Synthetic benchmarks

- STORECOMM\((n)\), SWAP\((n)\), STOREINV\((n)\): arrays with extensionality
- IOS\((n)\): arrays and integer offsets
- QUEUE\((n)\): records, arrays, integer offsets
- CIRCULAR_QUEUE\((n, k)\): records, arrays, integer offsets mod \(k\)

STORECOMM\((n)\), SWAP\((n)\), STOREINV\((n)\): both valid and invalid instances

Parameter \(n\): test scalability
Performances on valid STORECOMM\((n)\) instances

Native input: CVC wins but E better than CVC Lite
Performances on valid STORECOMM(n) instances

Flat input: E matches CVC
Performances on invalid STORECOMM($n$) instances

Native input: prover conceived for unsat handles sat even better

![Graph showing run time vs instance size for CVC, CVC Lite, and E (good-lpo) with different index types.](image)
Performances on invalid STORECOMM($n$) instances

![Graph showing run time (s) vs instance size with four lines representing different input types: CVC (flattened), CVC Lite (flattened), E (good-lpo), built-in index type, and E (good-lpo), axiomatized indices.]

Flat input: E surpasses CVC
Performances on valid \textit{SWAP}(n) instances

Harder problem: no system terminates for $n \geq 10$
Performances on valid \textit{SWAP}\((n)\) instances

![Graph comparing run time (s) vs instance size for CVC, CVC Lite, and E (good-lpo)](image)

Added lemma for E: additional flexibility for the prover

Maria Paola Bonacina

On a rewriting approach to satisfiability procedures: extension
Performances on invalid \textit{SWAP}(n) instances

\begin{figure}
\centering
\includegraphics[width=\textwidth]{performance_plot.png}
\caption{Comparison of run times for CVC, CVC Lite, and E (good-lpo) on invalid \textit{SWAP}(n) instances.}
\end{figure}

Easier problem, but E clearly ahead
Performances on valid STOREINV(n) instances

\[
E(\text{std\text{-}kbo}) \text{ does it in nearly constant time!}
\]
Performances on invalid STOREINV(n) instances

Not as good for E but run times are minimal
Performances on IOS instances

CVC and CVC Lite have built-in $\mathcal{L}A(\mathcal{R})$ and $\mathcal{L}A(\mathcal{I})$ respectively!
Performances on QUEUE instances (plain queues)

CVC wins (built-in arithmetic!) but E matches CVC Lite
Performances on \texttt{CIRCULAR\_QUEUE}(n, k) instances $k = 3$

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

Test performance on huge sets of literals.
Run time distribution for \textit{E(auto)} on UCLID set

\textit{Auto mode}: prover chooses search plan by itself
Better run time distribution for E on UCLID set

Optimized strategy: found by testing on random sample of 500 problems (less than 1%)
Summary

- 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 off the shelf and
  - conceived for very different search problems
  compares amazingly well with state-of-the-art verification tools
Directions for further research

- Prover’s search plans for $T$-sat problems
- More or stronger termination results
- Precise relationship between variable-inactive and stably-infinite, convex
- Integration with approaches for full $\mathcal{L}A$ or bit-vectors
- $T$-decision procedures (arbitrary quantifier-free formulæ): integration with SAT-solver? Other approaches?
- Combination with automated model building
- In general: explore “big” engines technology for decision procedures
Big picture

Reasoning environments for verification (and more):

- SAT-solvers
- “Little” engines
- “Big” engines
- Good interfaces
- ... ... ...