The Rewriting Approach to Decision Procedures

Alessandro Armando

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

Security & Trust Research Unit
FBK-IRST
Trento

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, ...

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, ...

"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 componenent (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 componenent (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 componenent (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 componenent (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 componenent (e.g. for rewriting, dealing with quantifiers, ...)
- **Challenge**: can big engines be (effectively) used as small engines?

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

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.

$$
\begin{array}{rcl}
\{f(f(f(a))) = b\} & \leadsto & \{f(a) = c_1, f(f(c_1)) = b\} \\
& \leadsto & \{f(a) = c_1, f(c_1) = c_2, f(c_2) = b\} \\
\{g(h(d))) \neq a\} & \leadsto & \{h(a) = c_1, g(c_1) \neq a\} \\
& \leadsto & \{h(a) = c_1, g(c_1) = c_2, c_2 \neq a\}
\end{array}
$$

- 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	$\dfrac{c = c' \qquad c = d}{c' = d}$	if $c \succ c'$ and $c \succ d$
Cong$_1$	$\dfrac{c_j = c'_j \qquad f(c_1, \ldots, c_j, \ldots, c_n) = c_{n+1}}{f(c_1, \ldots, c'_j, \ldots, c_n) = c_{n+1}}$	if $c_j \succ c'_j$
Cong$_2$	$\dfrac{f(c_1, \ldots, c_n) = c'_{n+1} \qquad f(c_1, \ldots, c_n) = c_{n+1}}{c_{n+1} = c'_{n+1}}$	if $c_{n+1} \succ c'_{n+1}$
DH	$\dfrac{c = c' \qquad c \neq d}{c' \neq d}$	if $c \succ c'$ and $c \succ d$
UN	$\dfrac{c \neq c}{\square}$	

Notice that we **only need to compare constants**!

A (extended) set of inference rules for CSAT(T_{UF})

CP	$\dfrac{c = c' \qquad c = d}{c' = d}$	if $c \succ c'$ and $c \succ d$
Cong$_1$	$\dfrac{c_j = c'_j \qquad f(c_1, \ldots, c_j, \ldots, c_n) = c_{n+1}}{f(c_1, \ldots, c'_j, \ldots, c_n) = c_{n+1}}$	if $c_j \succ c'_j$
Cong$_2$	$\dfrac{f(c_1, \ldots, c_n) = c'_{n+1} \qquad f(c_1, \ldots, c_n) = c_{n+1}}{c_{n+1} = c'_{n+1}}$	if $c_{n+1} \succ c'_{n+1}$
DH	$\dfrac{c = c' \qquad c \neq d}{c' \neq d}$	if $c \succ c'$ and $c \succ d$
UN	$\dfrac{c \neq c}{\square}$	

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 $\Box$ 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

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:

$$\begin{aligned} \mathrm{read}(\mathrm{write}(A, I, E), I) &= E \\ I = J \vee \mathrm{read}(\mathrm{write}(A, I, E), J) &= \mathrm{read}(A, J) \end{aligned}$$

where A, I, 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)

Name	Rule	Conditions
Sup.	$\dfrac{\Gamma \to \Delta, l[u'] = r \quad \Pi \to \Sigma, u = v}{\Gamma, \Pi \to \Delta, \Sigma, l[v] = r}$	$u \not\preceq v$, $l[u'] \not\preceq r$, $*$
Par.	$\dfrac{\Gamma, l[u'] = r \to \Delta \quad \Pi \to \Sigma, u = v}{l[v] = r, \Gamma, \Pi \to \Delta, \Sigma}$	$u \not\preceq v$, $l[u'] \not\preceq r$, $*$
Ref.	$\dfrac{\Gamma, u' = u \to \Delta}{\Gamma \to \Delta}$	$(u' = u) \not\prec (\Gamma \cup \Delta)$
Fac.	$\dfrac{\Gamma \to \Delta, u = v, u' = v'}{\Gamma, v = v' \to \Delta, u = v'}$	$u \not\preceq v$, $u \not\preceq \Gamma$, $(u = v) \not\prec \{u' = v'\} \cup \Delta$

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

$**$ $\sigma = mgu(u, u')$ implicitly applied to consequents and conditions

Contraction rules of $\mathcal{SP}$ (II)

Name	Rule	Conditions
Subsumption	$\dfrac{S \cup \{C, C'\}}{S \cup \{C\}}$	for some θ, $\theta(C) \subseteq C'$, and for no ρ, $\rho(C') = C$
Simplification	$\dfrac{S \cup \{C[\theta(l)], l = r\}}{S \cup \{C[\theta(r)], l = r\}}$	$\theta(l) \succ \theta(r)$, $C[\theta(l)] \succ (\theta(l) = \theta(r))$
Deletion	$\dfrac{S \cup \{\Gamma \rightarrow \Delta, t = t\}}{S}$	

Orderings

- Requirement: $f(c_1, \ldots, c_n) \succ c_0$
for each non-constant symbol f and constant c_i ($i = 0, 1, ..., n$)
- [Definition:] $(a = b) \succ (c = d)$ iff $\{a, b\} \succ\!\succ \{c, d\}$
(where $\succ\!\succ$ 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 $\mathrm{car}(\mathrm{cons}(X, Y)) = X$ and $\mathrm{cons}(c_1, c_2) = d$ yielding $\mathrm{car}(d) = c_1$ and
  - ▷ a superposition between $\mathrm{cdr}(\mathrm{cons}(X, Y)) = Y$ and $\mathrm{cons}(c_1, c_2) = d$ yielding $\mathrm{cdr}(d) = c_2$

Example: $\mathcal{SP}$ on lists (III)

- 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

Theory of lists: some remarks

- Recall that in the proof of termination of $\mathcal{SP}$ on $Ax(\mathcal{L}) \cup S$, we have observed that inferences between clauses in S were already considered for the ground case
- So, if we consider a signature $\Sigma := \{\text{cons}, \text{car}, \text{cdr}\} \cup \Sigma_{UF}$, where Σ_{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 $\mathcal{L} \cup T_{UF} \cup S$, 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 $\mathcal{SP}$

Lemma

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

- Termination follows

Lemma

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

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

Theorem

$\mathcal{SP}$ is a decision procedure for $\mathcal{L}$.

A rewriting approach: theories of **lists**

- Theory of uninterpreted functions: Σ_{UF} := finite set of function symbols, $Ax(UF) := \emptyset$
- Theory of lists *à la* Shostak: $\Sigma_{\mathcal{L}_{Sh}} := \{\mathrm{cons}, \mathrm{car}, \mathrm{cdr}\} \cup \Sigma_{UF}$,

$$
\begin{array}{rcl}
Ax(\mathcal{L}_{Sh}) & := & \{\mathrm{car}(\mathrm{cons}(X, Y)) = X, \mathrm{cdr}(\mathrm{cons}(X, Y)) = Y, \\
& & \mathrm{cons}(\mathrm{car}(X), \mathrm{cdr}(X)) = X\}
\end{array}
$$

- Theory of lists *à la* Nelson-Oppen: $\Sigma_{\mathcal{L}_{NO}} := \{\mathrm{cons}, \mathrm{car}, \mathrm{cdr}, \mathrm{atom}\} \cup \Sigma_{UF}$,

$$
\begin{array}{rcl}
Ax(\mathcal{L}_{NO}) & := & \{\mathrm{car}(\mathrm{cons}(X, Y)) = X, \mathrm{cdr}(\mathrm{cons}(X, Y)) = Y, \\
& & \neg\mathrm{atom}(\mathrm{cons}(X, Y)) \\
& & \mathrm{atom}(X) \vee \mathrm{cons}(\mathrm{car}(X), \mathrm{cdr}(X)) = X\}
\end{array}
$$

A rewriting approach: theories of **arrays**

- arrays w/ extensionality: $\Sigma_{\mathcal{A}^s} := \{\text{rd}, \text{wr}\} \cup \Sigma_{UF}$,

$$
\begin{array}{lcl}
Ax(\mathcal{A}^s) & := & \left\{ \begin{array}{l} \text{rd}(\text{wr}(A, I, E), I) = E \\ I = J \vee \text{rd}(\text{wr}(A, I, E), J) = \text{rd}(A, J) \end{array} \right\} \\
Ax(\mathcal{A}^s_e) & := & Ax(\mathcal{A}^s) \cup \\
 & & \{\forall A, B.(\forall I.(\text{rd}(A, I) = \text{rd}(B, I)) \Longrightarrow A = B)\}
\end{array}
$$

A rewriting approach: theories of **records**

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

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

$$Ax(\mathcal{R}^s_e) \quad := \quad Ax(\mathcal{A}^s) \cup \{\forall X, Y.(\bigwedge_{i=1}^{n} \text{rsel}_i(X) = \text{rsel}_i(Y) \Longrightarrow X = Y)\}$$

A rewriting approach: small fragments of Arithmetics

- Integer Offsets: $\Sigma_{\mathcal{I}} := \{\mathrm{succ}, \mathrm{prec}\} \cup \Sigma_{UF}$,

$$Ax(\mathcal{I}) := \left\{ \begin{array}{ll} \mathrm{succ}(\mathrm{prec}(X)) = X, \mathrm{prec}(\mathrm{succ}(X)) = X, & \\ \underbrace{\mathrm{succ}^i(X) \neq X}_{\text{acyclicity}} & \text{for } i > 0 \end{array} \right\}$$

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

- Integer Offsets Modulo: $\Sigma_{\mathcal{I}_k} := \{\mathrm{succ}, \mathrm{prec}\} \cup \Sigma_{UF}$,

$$Ax(\mathcal{I}_k) := \left\{ \begin{array}{ll} \mathrm{succ}(\mathrm{prec}(X)) = X, \mathrm{prec}(\mathrm{succ}(X)) = X, & \\ \underbrace{\mathrm{succ}^i(X) \neq X}_{k\text{-acyclicity}} & \text{for } 1 \leq i \leq k-1 \\ \mathrm{succ}^k(X) = X & \end{array} \right\}$$

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

1. A. Armando, S. Ranise, M. Rusinowitch. **Uniform Derivation of Decision Procedures by Superposition.** In the Proceedings on the Annual Conference on Computer Science Logic (CSL01), Paris, France, 10-13 September 2001, pp. 513-527.
2. A. Armando, S. Ranise, M. Rusinowitch. **The Rewriting Approach to Satisfiability Procedures.** Information and Computation 183 (2003) pp. 140-164.

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 $\mathcal{T}$-reduced flat problem. We call $\mathcal{T}$-good any such $\succ$.
- Everything fully automated except for termination proof

1. A. Armando, S. Ranise, M. Rusinowitch. **Uniform Derivation of Decision Procedures by Superposition.** In the Proceedings on the Annual Conference on Computer Science Logic (CSL01), Paris, France, 10-13 September 2001, pp. 513-527.
2. A. Armando, S. Ranise, M. Rusinowitch. **The Rewriting Approach to Satisfiability Procedures.** Information and Computation 183 (2003) pp. 140-164.

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 $\mathcal{T}$-reduced flat problem. We call $\mathcal{T}$-good any such $\succ$.
- Everything fully automated except for termination proof

1. A. Armando, S. Ranise, M. Rusinowitch. **Uniform Derivation of Decision Procedures by Superposition.** In the Proceedings on the Annual Conference on Computer Science Logic (CSL01), Paris, France, 10-13 September 2001, pp. 513-527.
2. A. Armando, S. Ranise, M. Rusinowitch. **The Rewriting Approach to Satisfiability Procedures.** Information and Computation 183 (2003) pp. 140-164.

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 $\mathcal{T}$-reduced flat problem. We call $\mathcal{T}$-good any such $\succ$.
- Everything fully automated except for termination proof

1. A. Armando, S. Ranise, M. Rusinowitch. **Uniform Derivation of Decision Procedures by Superposition.** In the Proceedings on the Annual Conference on Computer Science Logic (CSL01), Paris, France, 10-13 September 2001, pp. 513-527.
2. A. Armando, S. Ranise, M. Rusinowitch. **The Rewriting Approach to Satisfiability Procedures.** Information and Computation 183 (2003) pp. 140-164.

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 $\mathcal{T}$-reduced flat problem. We call $\mathcal{T}$-good any such $\succ$.
- Everything fully automated except for termination proof

1. A. Armando, S. Ranise, M. Rusinowitch. **Uniform Derivation of Decision Procedures by Superposition.** In the Proceedings on the Annual Conference on Computer Science Logic (CSL01), Paris, France, 10-13 September 2001, pp. 513-527.
2. A. Armando, S. Ranise, M. Rusinowitch. **The Rewriting Approach to Satisfiability Procedures.** Information and Computation 183 (2003) pp. 140-164.

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 $\mathcal{T}$-reduced flat problem. We call $\mathcal{T}$-good any such $\succ$.
- Everything fully automated except for termination proof

1. A. Armando, S. Ranise, M. Rusinowitch. **Uniform Derivation of Decision Procedures by Superposition.** In the Proceedings on the Annual Conference on Computer Science Logic (CSL01), Paris, France, 10-13 September 2001, pp. 513-527.
2. A. Armando, S. Ranise, M. Rusinowitch. **The Rewriting Approach to Satisfiability Procedures.** Information and Computation 183 (2003) pp. 140-164.

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]

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

A modularity theorem for combination of theories

Question: If $\mathcal{SP}$ 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 for combination of theories

Question: If $\mathcal{SP}$ 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 for combination of theories

Question: If $\mathcal{SP}$ 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 $x \notin Var(t)$); 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.

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

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(good-lpo) and E(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

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

where x and y have sort ARRAY,
z has sort INDEX, and
v has sort ELEM.

Arrays: termination of $\mathcal{SP}$

$\mathcal{A}$-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 $\mathcal{SP}_{\succ}$-strategy is a satisfiability procedure for the theories of arrays and arrays with extensionality.

Benchmarks for arrays

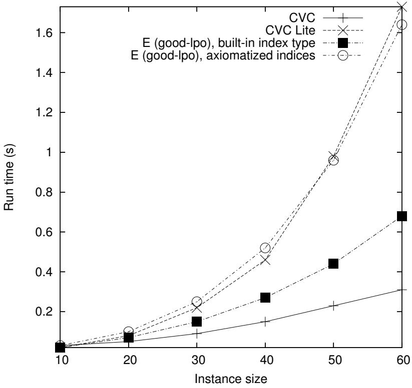
Parametric problem instances to assess scalability.

- 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.

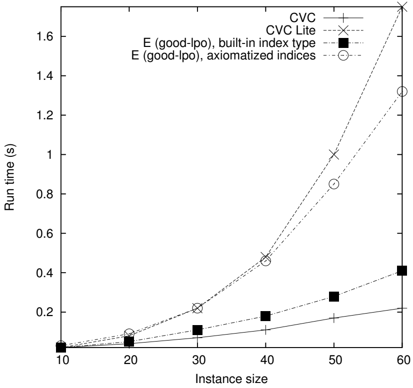
Both valid and invalid instances generated.

Performances on STORECOMM(n) instances

valid instances

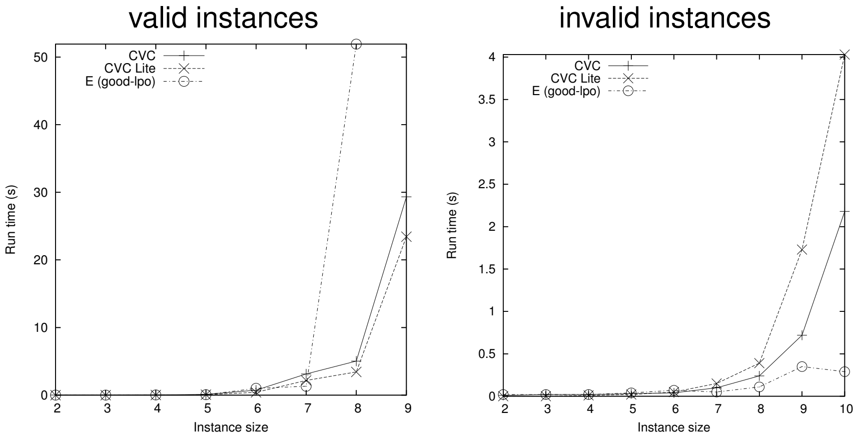


invalid instances



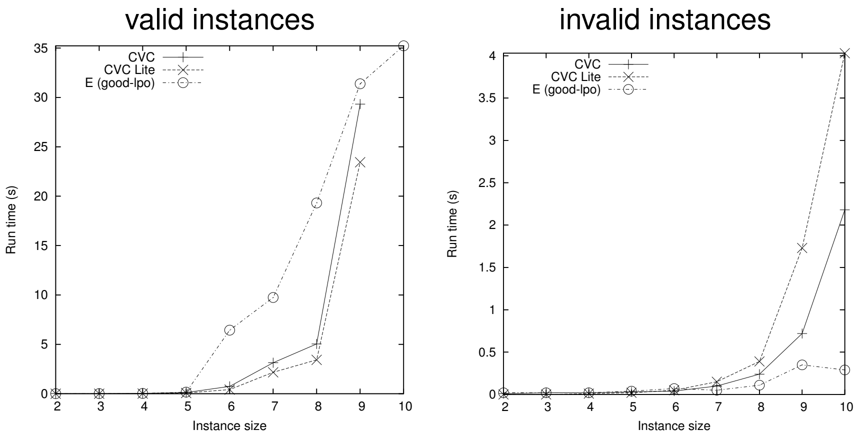
CVC wins but E better than CVC Lite

Performances on SWAP(n) instances

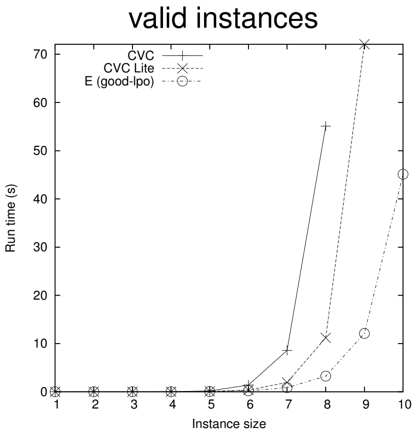


CVC and CVC Light win on valid instances, E wins on invalid ones.

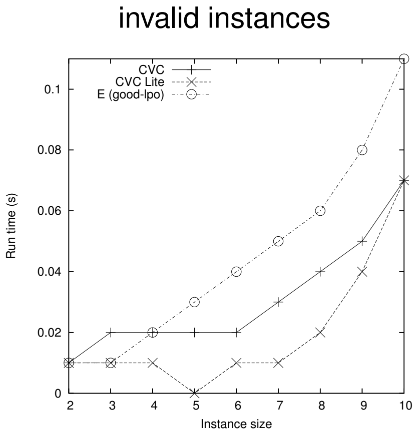
Performances on SWAP(n) instances



CVC and CVC Light win on valid instances, E wins on invalid ones.
The situation improves by adding a lemma to E.



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



Not as good for E but run times are minimal

Integer offsets: presentation

A fragment of the theory of the integers:
s: successor
p: predecessor

Theory of integer offsets

$$\begin{array}{ll} \forall x. & s(p(x)) \simeq x \\ \forall x. & p(s(x)) \simeq x \\ \forall x. & s^i(x) \not\simeq x \quad \text{for } i > 0 \end{array}$$

Infinitely many acyclicity axioms!

$\mathcal{I}$-reduction:

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

$\mathcal{I}$-good: any CSO.

Termination: case analysis of generated clauses.

Theorem: A fair $\mathcal{SP}_{\succ}$-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.

	Theories	
	arrays	ios
STORECOMM, SWAP, STOREINV	•	
IOS	•	•

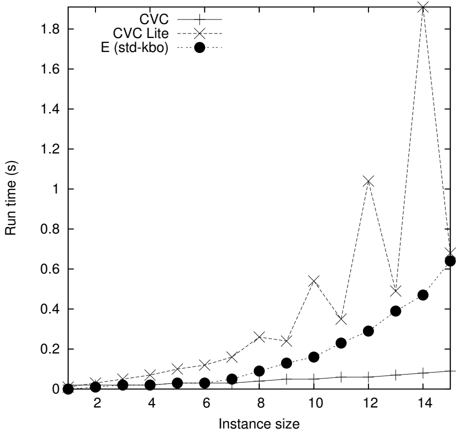
Based on the following observation:

```
for(k=1;k<=n;k++)
  a[i+k]=a[i]+k;
```

```
for(k=1;k<=n;k++)
  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`.

Performances on IOS instances



CVC and CVC Lite have built-in $\mathcal{LA}(\mathcal{R})$ and $\mathcal{LA}(\mathcal{I})$ respectively!

Records: presentation

Sort $\text{REC}(id_1 : \text{T}_1, \ldots, id_n : \text{T}_n)$

Theory of records

$$
\begin{array}{lll}
\forall x, v. & \text{rselect}_i(\text{rstore}_i(x, v)) \simeq v & 1 \leq i \leq n \\
\forall x, v. & \text{rselect}_j(\text{rstore}_i(x, v)) \simeq \text{rselect}_j(x) & 1 \leq i \neq j \leq n \\
\forall x, y. & (\bigwedge_{i=1}^{n} \text{rselect}_i(x) \simeq \text{rselect}_i(y) \supset x \simeq y) &
\end{array}
$$

where x, y have sort REC and v has sort T_i.

Records: termination of $\mathcal{SP}$

$\mathcal{R}$-reduction: eliminate disequalities between records by resolution with extensionality + splitting.

$\mathcal{R}$-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 $\mathcal{R}$-good $\mathcal{SP}_{\succ}$-strategy is a satisfiability procedure for the theories of records and records with extensionality.

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

Synthetic benchmarks

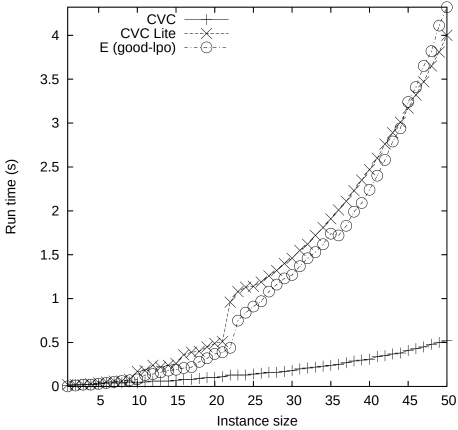
Queues can be defined on top a combination of theories of arrays, records and integer offsets:

	Theories		
	arrays	ios	records
STORECOMM, SWAP, STOREINV	•		
IOS	•	•	
QUEUE	•	•	•

$$
\begin{aligned}
\mathrm{enqueue}(v, x) &= \mathrm{rstore}_t(\mathrm{rstore}_i(x, \mathrm{store}(\mathrm{rselect}_i(x), \mathrm{rselect}_t(x), v)), \\
&\qquad \mathrm{s}(\mathrm{rselect}_t(x))) \\
\mathrm{dequeue}(x) &= \mathrm{rstore}_h(x, \mathrm{s}(\mathrm{rselect}_h(x))) \\
\mathrm{first}(x) &= \mathrm{select}(\mathrm{rselect}_i(x), \mathrm{rselect}_h(x)) \\
\mathrm{last}(x) &= \mathrm{select}(\mathrm{rselect}_i(x), \mathrm{p}(\mathrm{rselect}_t(x))) \\
\mathrm{reset}(x) &= \mathrm{rstore}_h(x, \mathrm{rselect}_t(x))
\end{aligned}
$$

$\mathtt{QUEUE}(n)$ expresses the property that if $q \in \mathtt{QUEUE}$ 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 $\mathrm{first}(q) = e_m$.

Performances on QUEUE instances



CVC wins (built-in arithmetic!) but E matches CVC Lite

Integer offsets modulo: presentation

To reason with indices ranging over the integers mod k ($k > 0$):

Theory of integer offsets modulo

$$\begin{array}{ll} \forall x. & \mathrm{s}(\mathrm{p}(x)) \simeq x \\ \forall x. & \mathrm{p}(\mathrm{s}(x)) \simeq x \\ \forall x. & \mathrm{s}^i(x) \not\simeq x \quad 1 \leq i \leq k-1 \\ \forall x. & \mathrm{s}^k(x) \simeq x \end{array}$$

Finitely many axioms.

Integer offsets modulo: termination of $\mathcal{SP}$

$\mathcal{I}$-reduction: same as above.

$\mathcal{I}$-good: any CSO.

Termination: case analysis of generated clauses.

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

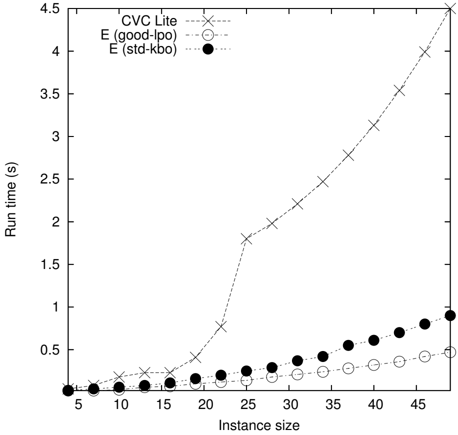
Termination also without $\mathcal{I}$-reduction.

Benchmarks for circular queues

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

	Theories			
	arrays	ios	records	mod_ios
STORECOMM, SWAP, STOREINV	•			
IOS	•	•		
QUEUE	•	•	•	
CIRCULAR_QUEUE	•	•		•

Performances on CIRCULAR_QUEUE(n, k) instances $k = 3$



CVC does not handle integers mod k, E clearly wins

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

“Real-world” problems

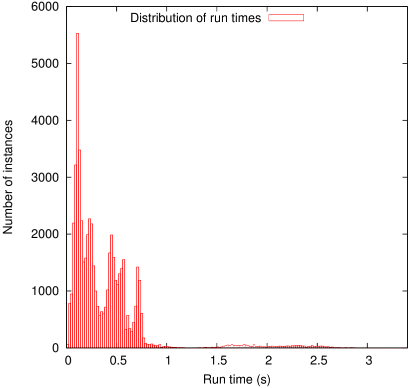
- UCLID [Bryant, Lahiri, Seshia 2002]: suite of problems
- haRVey [Déharbe and Ranise 2003]: extract T-sat problems
- over 55,000 proof tasks: integer offsets and equality
- all valid

	Theories				
	arrays	ios	records	mod_ios	euf
STORECOMM, SWAP, STOREINV	•				
IOS	•	•			
QUEUE	•	•	•		
CIRCULAR_QUEUE	•	•		•	
UCLID		•			•

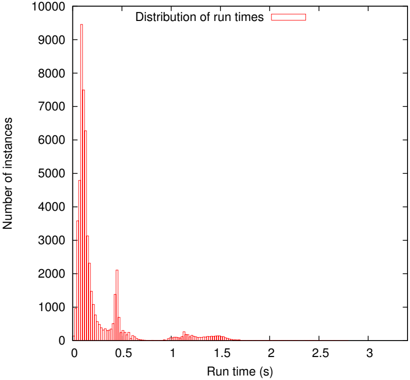
Test performance on huge sets of literals.

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%)



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 essentially off the shelf and
  - conceived for very different search problems

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