Big proof engines as little proof engines: new results on rewrite-based satisfiability procedures

Maria Paola Bonacina[1]

Dipartimento di Informatica
Università degli Studi di Verona
Verona, Italy, EU

Talk given at the Dept. of Computer Science and Engineering
Chalmers University of Technology and Gothenburg University
Gothenburg, Sweden, EU

12 May 2006

[1] Joint work with Alessandro Armando, Mnacho Echenim, Silvio Ranise, and Stephan Schulz

Decision procedures

- **Objective**: Decision procedures for application of automated reasoning to verification
- **Desiderata**: Efficient, scalable, expressive, proof-producing, easy to build, combine, 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, combined by Nelson-Oppen scheme
- "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
- *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

Rewrite-based methodology for T-satisfiability

Theories of equality, data structures, fragments integer arithmetic

General 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: huge sets of literals

Summary

What kind of theorem prover?

First-order logic with equality

$\mathcal{SP}$ inference system: rewrite-based

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

Complete simplification ordering (CSO) $\succ$ on terms, literals and clauses: $\mathcal{SP}_\succ$

(Fair) $\mathcal{SP}_\succ$*-strategy* : $\mathcal{SP}_\succ$ + (fair) search plan

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 $\mathcal{T}$-reduced flat problem, provided $\succ$ is *$\mathcal{T}$-good*
- Everything *fully automated* except for termination proof

Covered theories

- *Non-empty lists*, *arrays* with and without extensionality, *finite sets* with extensionality [Armando, Ranise, Rusinowitch 2003]
- *Records* with and without extensionality, *possibly empty lists*, *integer offsets*, *integer offsets modulo* [Armando, Bonacina, Ranise, Schulz 2005]
- *Equality* [Lankford 1975]

In experiments: arrays, records, integer offsets, integer offsets modulo, equality and combinations (queues, circular queues)

The theory of records

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

Presentation $\mathcal{R}$:

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

where x and y have sort REC and v has sort T_i.
Extensionality is the third axiom.

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 polynomial $\mathcal{R}$-satisfiability procedure (with or without extensionality).

The theory of integer offsets

Fragment of the theory of the integers:
s: successor   p: predecessor

Presentation $\mathcal{I}$:

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

Infinitely many acyclicity axioms (Ac)
Remark: these axioms imply that s is *injective* (Inj)

$$\forall x, y.\ \mathrm{s}(x) \simeq s(y) \supset x \simeq y$$

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

$\mathcal{I}$-*reduction*: eliminate p by replacing $\mathsf{p}(c) \simeq d$ with $c \simeq \mathsf{s}(d)$: first two axioms no longer needed, provided *Inj* is added.
Bound the number of acyclicity axioms:
$\forall x.\, \mathsf{s}^i(x) \not\simeq x$ for $0 < i \leq n$
if there are n occurrences of s.

$\mathcal{I}$-*good*: $t \succ c$ for all constants c and terms t with top symbol s.

Termination: case analysis of generated clauses.

Theorem: A fair $\mathcal{I}$-good $\mathcal{SP}_{\succ}$-strategy is an exponential $\mathcal{I}$-satisfiability procedure (polynomial on *Ac* only).

The theory of integer offsets modulo

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

Presentation $\mathcal{I}_k$:

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

Finitely many axioms.

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

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

$\mathcal{I}_k$*-good*: same as $\mathcal{I}$-good.

Termination: case analysis of generated clauses.

Theorem: A fair $\mathcal{I}$-good $\mathcal{SP}_{\succ}$-strategy is an exponential $\mathcal{I}_k$-satisfiability procedure.

The theory of possibly empty lists

Presentation $\mathcal{L}$:

$$\begin{aligned}
\forall x, y.\ \ \mathsf{car}(\mathsf{cons}(x, y)) &\simeq x \\
\forall x, y.\ \ \mathsf{cdr}(\mathsf{cons}(x, y)) &\simeq y \\
\forall x, y.\ \ \mathsf{cons}(x, y) &\not\simeq \mathsf{nil} \\
\forall y.\ y \not\simeq \mathsf{nil} \supset \mathsf{cons}(\mathsf{car}(y), \mathsf{cdr}(y)) &\simeq y \\
\mathsf{car}(\mathsf{nil}) &\simeq \mathsf{nil} \\
\mathsf{cdr}(\mathsf{nil}) &\simeq \mathsf{nil}
\end{aligned}$$

Unsorted, possibly cyclic lists.

Possibly empty lists: termination of $\mathcal{SP}$

$\mathcal{L}$-reduction: none.

$\mathcal{L}$-good:

1. $t \succ c$ for all ground compound terms t and constants c,
2. $t \succ$ nil for all terms t with top symbol cons.

Termination: case analysis of generated clauses.

Theorem: A fair $\mathcal{L}$-good $\mathcal{SP}_{\succ}$-strategy is an exponential $\mathcal{L}$-satisfiability procedure.

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 for $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$
- $\mathcal{T}_i$-reduction and flattening apply as for each theory
- Termination?

Three simple conditions

- $\succ$ $\mathcal{T}$-*good*, if $\mathcal{T}_i$-*good* for all i, $1 \leq i \leq n$
- The $\mathcal{T}_i$ do not share function symbols
  (*Intuition*: no paramodulation from compound terms across theories)
- Each $\mathcal{T}_i$ is *variable-inactive*:
  in no persistent clause $t \simeq x$ with $x \notin Var(t)$ is maximal
  (*Intuition*: no paramodulation from variables across theories, since for $t \simeq x$ where $x \in Var(t)$ it is $t \succ x$)

The modularity theorem

Theorem: if

- No shared function symbol (shared constants allowed),
- Variable-inactive theories $\mathcal{T}_i$, $1 \leq i \leq n$,
- A fair $\mathcal{T}_i$-good $\mathcal{SP}_{\succ}$-strategy is $\mathcal{T}_i$-satisfiability procedure,

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

Equality, *arrays* (with or without extensionality), *records* (with or without extensionality), *integer offsets*, *integer offsets modulo* and *possibly empty lists* all satisfy these hypotheses.

A few remarks on generality I

- *Purely equational theories*:
  no trivial models $\Rightarrow$ variable-inactive
- *Horn theories*:
  no trivial models + maximal unit strategy $\Rightarrow$ variable-inactive
- *Maximal unit strategy*:
  restricts superposition to unit clauses and paramodulates unit clauses into maximal negative literals [Dershowitz 1990]

A few remarks on generality II

- *First-order theories*: variable-inactive excludes, e.g., $a_1 \simeq x \vee \ldots \vee a_n \simeq x$, a_i constants (*)
  Such a clause implies *not stably-infinite*, hence *not convex* under the no trivial models hypothesis:
  if $\mathcal{T}_i$ not variable-inactive for (*), Nelson-Oppen does not apply either.
- $\mathcal{T}$ *convex*:
  $\mathcal{T} \models H \supset \bigvee_{i=1}^{n} P_i$ implies $\mathcal{T} \models H \supset P_j$ for some j.
- $\mathcal{T}$ *stably infinite*:
  quantifier-free $\mathcal{T}$-formula has $\mathcal{T}$-model iff has infinite $\mathcal{T}$-model.

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 ...

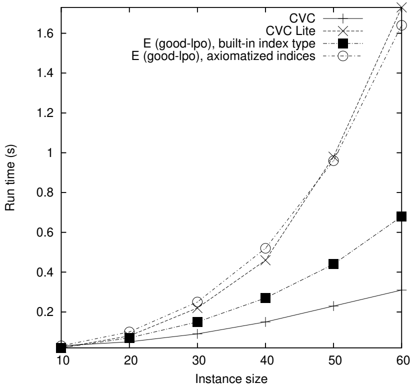
Synthetic benchmarks

- 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

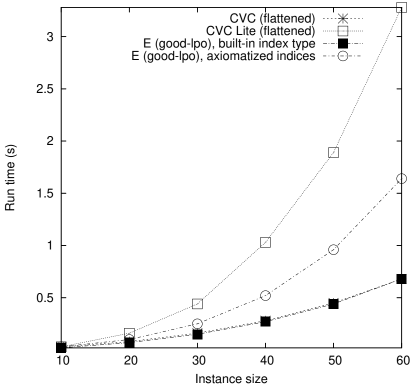
Parameter n: test *scalability*

Performances on valid STORECOMM(n) instances



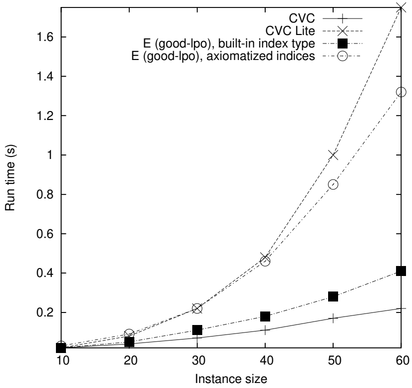
Native input: CVC wins but E better than CVC Lite

Performances on valid STORECOMM(n) instances



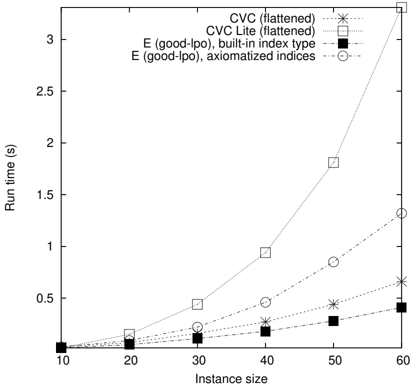
Flat input: E matches CVC

Performances on invalid STORECOMM(n) instances



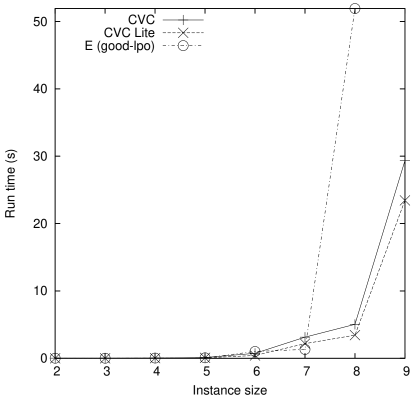
Native input: prover conceived for unsat handles sat even better

Performances on invalid STORECOMM(n) instances



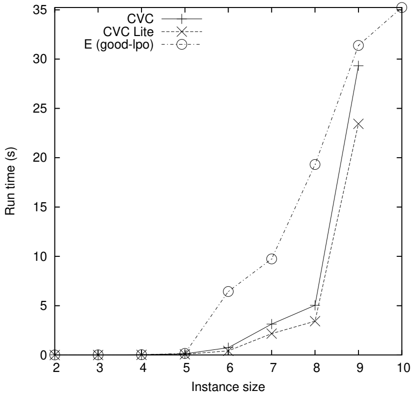
Flat input: E surpasses CVC

Performances on valid $\mathtt{SWAP}(n)$ instances



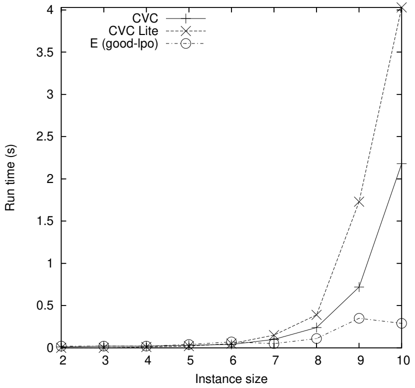
Harder problem: no system terminates for $n \geq 10$

Performances on valid SWAP(n) instances



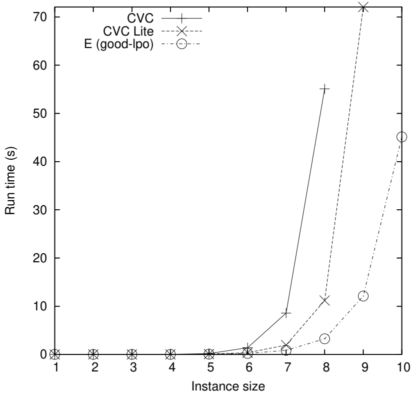
Added lemma for E: additional flexibility for the prover

Performances on invalid SWAP(n) instances



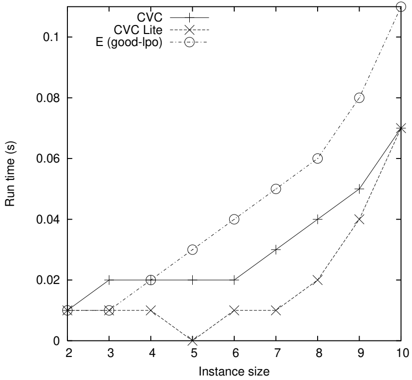
Easier problem, but E clearly ahead

Performances on valid STOREINV(n) instances



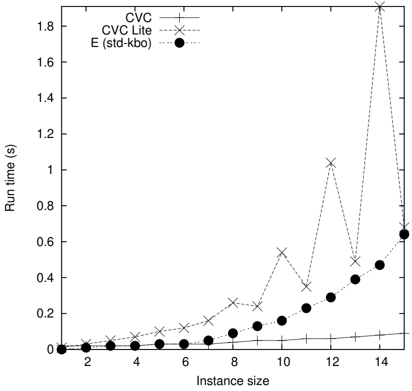
E(std-kbo) does it in *nearly constant time*!

Performances on invalid STOREINV(n) instances



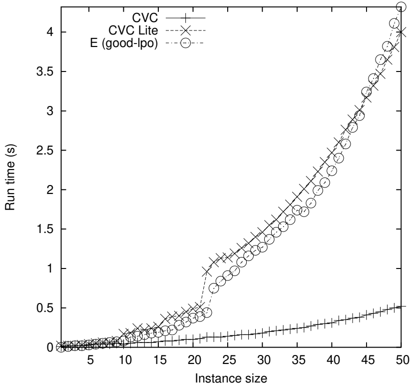
Not as good for E but run times are minimal

Performances on IOS instances



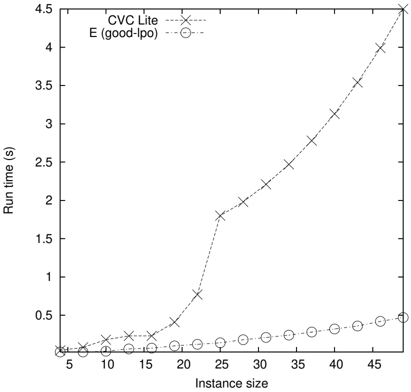
CVC and CVC Lite have built-in $\mathcal{LA}(\mathcal{R})$ and $\mathcal{LA}(\mathcal{I})$ respectively!

Performances on QUEUE instances (plain queues)



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

Performances on $\texttt{CIRCULAR\_QUEUE}(n, k)$ instances $k = 3$



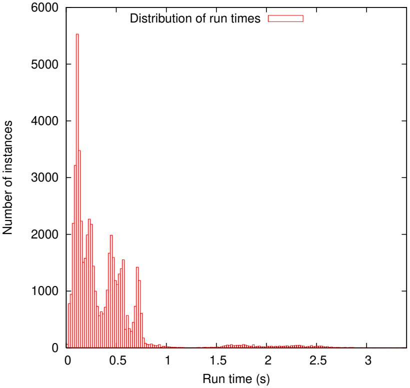
CVC does not handle integers mod k, E clearly wins

“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

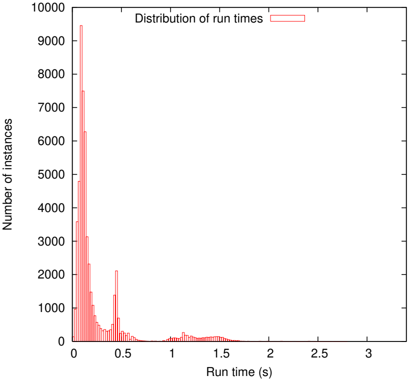
Test performance on huge sets of literals.

Run time distribution for *E(auto)* on UCLID set



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 relevant to verification
- *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 built-in theories validity checkers

Current work

- Precise relationship between variable-inactive and stably-infinite, convex (e.g., $\mathcal{R}$ and $\mathcal{A}$, arrays, are not convex) [Bonacina, Ghilardi, Nicolini, Ranise, Zucchelli 2006]
- $\mathcal{T}$-satisfiability procedures for all theories of *recursive data structures*:
  one constructor and k selector ($k = 1$: integer offsets, $k = 2$: lists) [Bonacina, Echenim 2006]
- $\mathcal{T}$-decision procedures (arbitrary quantifier-free formulæ) [Bonacina, Echenim 2006]

Directions for future work

- Search plans for $\mathcal{T}$-sat problems
- Finer complexity results for specific search plans
- More or stronger termination results
- Integration with approaches for full $\mathcal{LA}$ or bit-vectors
- $\mathcal{T}$-decision procedures: integration with SAT-solver?
- Combination with automated model building
- In general: explore "big" engines technology for decision procedures

Big picture

Reasoning environments for verification (and more):

- SAT-solvers (e.g., DPLL, Stålmarck's method)
- "Little" engines
- "Big" engines (e.g., Rewrite-based, Stålmarck's method extended)
- Good interfaces
- ... ...