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

Maria Paola Bonacina[1]

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

Extended version of the talk presented at the
5th Int. Symposium on Frontiers of Combining Systems (FroCoS)
Vienna, Austria, EU

19 September 2005

---

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

- Motivation
  - The big picture
  - Decision procedures
  - Little engines and big engines of proof
- 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

Certification of traditional systems (e.g., airplane wing)

- Build mathematical models (e.g., sets of differential equations) of the design, its environment, and requirements
- Use calculation to establish that the design in the context of the environment satisfies the requirements
- Only useful when *mechanized*
- Models are validated by testing
- Limited testing suffice because we are dealing with continuous systems
- This is *product-based* certification

Certification of software systems

- Mostly done by controlling, monitoring and documenting the process of software creation
- This is *process-based* certification
- Testing is product-based but not sufficient because we are dealing with discrete systems:
    - Complete testing is unfeasible
    - Extrapolation from incomplete tests unjustified

Product-based certification for software

- Build mathematical models of the design, its environment, and requirements
    - The "applied math" of Computer Science is formal logic
    - Models are formal descriptions in some logical systems
- Use calculation to establish that the design in the context of the environment satisfies the requirements
    - Calculation in formal logic is done by *theorem proving* or *model checking*:

      $$\mathit{assumptions} \; + \; \mathit{design} \; + \; \mathit{environment} \vdash \mathit{requirements}$$

      It can cover all modeled behaviors, even if numerous or infinite (the power of symbolic reasoning)
- Only useful when *mechanized*
    - So need **automated** theorem proving or model checking

However ...

- Formal calculations
  - Are undecidable in general
  - Even decidable problems have much greater computational complexity than mechanizations of continuous mathematics
- So full automation is impossible in general: need to
  - Rely on heuristics which will sometimes fail: *automated theorem proving with heuristic search*
  - Rely on human guidance: *interactive theorem proving*
  - Trade-off accuracy or completeness of the model for tractability and automation of calculation: *model checking*

Current practice

- Model checking used to look for errors (debugging)
- Verification (show the absence of errors) much less practiced
- **Challenges**:
    - Make model checking useful for verification
    - Make relevant theorem proving automated
    - Make model checking and theorem proving work together

Research context

- Model checking requires simple models (e.g., finite state)
  - But can be used to verify properties of a complex model if it has *property-preserving* abstraction
  - "Abstract-check-refine" paradigm
  - **First key idea**: use theorem proving to calculate the abstraction
- Classical verification poses correctness as a single "big theorem": failure to prove it (if true) means disaster
  - **Second key idea**: "fault-tolerant" theorem proving:
  - Prove lots of small theorems instead of a big one
  - In a context where some failures can be tolerated
- Automated abstraction provides precisely such a context!

Decision procedures

This notion of theorem proving is based on powerful decision procedures:

- Reasoning about software requires reasoning about theories of data types, e.g., lists, arrays, integers, trees, tuples or records, sets, reals.
- Some of these theories or fragments thereof are *decidable*.
- *Decision procedures* to be embedded in verification tools and proof assistants, interfaced with model checkers.

Decision procedure for T-satisfiability

An algorithm that takes in input a set S of *ground* T-literals and reports:

- *unsatisfiable* if no T-model satisfies S,
- *satisfiable* otherwise (should return the model as well).

If such an algorithm exists, T-satisfiability is decidable.

Problems that reduce to T-(un)satisfiability

Decision procedures do not handle quantifiers: either the problem is *ground* (i.e., no variables) or there are only $\forall$-quantified variables that are eliminated through negation and Skolemization:

- Word Problem: $T \models s \simeq t$, if $S = \{s \not\simeq t\}$ is T-unsat.
- Uniform Word Problem: $T \models \bigwedge_{i=1}^{n} p_i \simeq q_i \supset s \simeq t$, if $S = \{p_1 \simeq q_1, \ldots p_n \simeq q_n, s \not\simeq t\}$ is T-unsat.
- Clausal Validity Problem: $T \models \bigwedge_{i=1}^{n} p_i \simeq q_i \supset \bigvee_{j=1}^{m} s_j \simeq t_j$, if $\{p_1 \simeq q_1, \ldots p_n \simeq q_n, s_1 \not\simeq t_1, \ldots s_m \not\simeq t_m\}$ is T-unsat.
- $T \models \varphi$ (arbitrary formula), if each conjunction of literals from $DNF(\neg\varphi)$ is T-unsat (not practical if DNF is generated explicitly).
- S is T-sat: model is counter-example to original conjecture.

Example of set of literals

$$x \leq y, \qquad y \leq x + z$$

$$p(x - y) \simeq \mathit{true}, \qquad p(z - y) \simeq \mathit{true}, \qquad p(0) \simeq \mathit{false}$$

$$\mathit{select}(\mathit{store}(v, i, 0), j) \simeq z, \qquad \mathit{select}(v, j) \simeq y$$

combines:

- the theory of equality with *free* (*uninterpreted*) function symbols (e.g., p), and
- integer arithmetic with *defined* (*interpreted*) function symbols (e.g., $+, -, \leq$), and
- the theory of arrays, where $\mathit{select}, \mathit{store}$ are *defined* (*interpreted*) function symbols.

Little engines of proof I

Design, prove sound and complete, and implement a satisfiability procedure for *each theory*, e.g.:

- Theory of equality with free symbols: *congruence closure* [Kozen 1977; Shostak 1978; Downey-Sethi-Tarjan 1980]
- Theory of lists: congruence closure with *axioms built-in* [Nelson-Oppen 1980; Shostak 1984]
- Theory of arrays with extensionality: congruence closure with *pre-processing wrt axioms* and case analysis [Stump-Barrett-Dill-Levitt 2001]

Little engines of proof II

Combination of theories [Nelson-Oppen 1979; Shostak 1984]:

$$T_1, \dots T_n$$

- $\mathcal{T}_i$'s don't share function symbols: if a T_i-term r occurs under a T_j symbol f, rename as x (new var) and add $x \simeq r$ (e.g., $c \simeq 2 + car(l)$ becomes $c \simeq 2 + x, x \simeq car(l)$)
- Communication among procedures: only equalities between variables
- Complete for *convex* theories: if $T \models \Gamma \supset \bigvee_{j=1}^{m} s_j \simeq t_j$, then $T \models \Gamma \supset s_j \simeq t_j$ for some j, where Γ is a conjunction of equalities

Little engines of proof III

- Equality with free function symbols is convex:
  if a disjunction of ground equalities is valid, one is valid
- Linear arithmetic is convex if there are only equalities, not with disequalities:
  - $\mathcal{LA}(Q)$: $x \leq y \vee y \leq x$ is valid but neither disjunct is.
  - $\mathcal{LA}(Z)$: $1 \leq x \wedge x \leq 2 \supset x \simeq 1 \vee x \simeq 2$ is valid but neither $1 \leq x \wedge x \leq 2 \supset x \simeq 1$ nor $1 \leq x \wedge x \leq 2 \supset x \simeq 2$ is.
- The theory of arrays is not convex:
  $i \simeq j \vee \mathit{select}(\mathit{store}(a, i, v), j) \simeq \mathit{select}(a, j)$ is valid but neither disjunct is.
- Non-convex: case analysis or “splitting” (in practice: backtracking): non-deterministic

Big engines of proof (in brief)

Methods for theorem proving in first-order logic with equality:

- Herbrand theorem (1930): unsatisfiability is semi-decidable
- Ordering-based methods: resolution, hyperresolution, subsumption, paramodulation/superposition, simplification
- Non-deterministic: combine with *fair* search plan to get deterministic semi-decision procedure
- Any first-order theory T, any conjecture φ: $T \cup \{\neg\varphi\} \vdash^{?} \bot$
- May have theories built-in (equality for sure) (e.g., AC)

Issues with little engines

- Combination of theories:
  done by combining procedures rather than theories:
  complicated, *ad hoc*
- Soundness and completeness proof: if given, is *ad hoc*
- Implementation: usually from scratch: correctness?
  integration in different environments? duplicated work?
- **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: given once and for all for first-order inference system
- Implementation: (re-)use first-order prover (techniques, code)
- Proof generation: already there by default
- Model generation: final T-sat set (starting point)
- **Key issue**: prove termination

Decision procedures: summary

- **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: summary

- “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?

What kind of theorem prover?

First-order logic with equality

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

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

- *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]

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

Records: presentation

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

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

where x, 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 satisfiability procedure for the theories of records and records with extensionality.

Integer offsets: presentation

A fragment of the theory of the integers:

s: successor

p: predecessor

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

Infinitely many *acyclicity axioms*!

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

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

$\mathcal{I}$-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.

Integer offsets modulo: presentation

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

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

Finitely many axioms.

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

$\mathcal{I}$-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.

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

- $\succ$ $\mathcal{T}$-*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 superposition from compound terms across theories)
- Each $\mathcal{T}_i$ 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 $\mathcal{SP}_{\succ}$-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.

Two remarks on generality

- *Purely equational theories*:
  no trivial models ⇒ variable-inactive
- *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 means *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.

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 ...

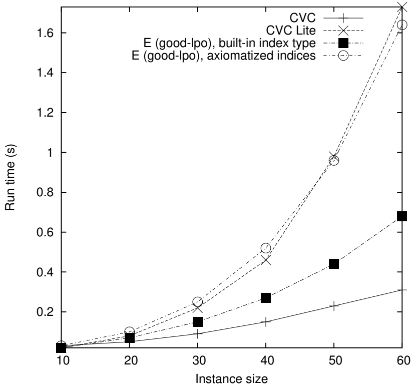
Synthetic benchmarks

- `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

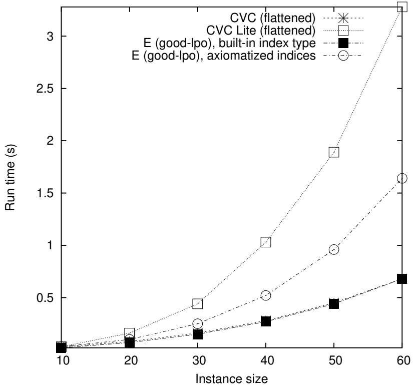
Parameter n: test *scalability*

Performances on valid $\texttt{STORECOMM}(n)$ instances



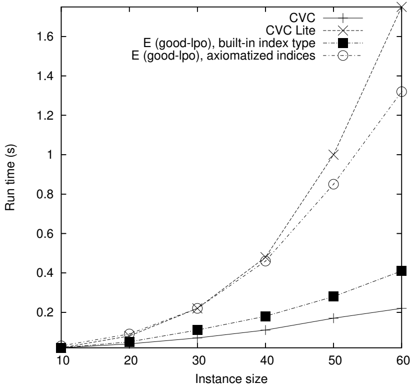
Native input: CVC wins but E better than CVC Lite

Performances on valid STORECOMM(n) instances



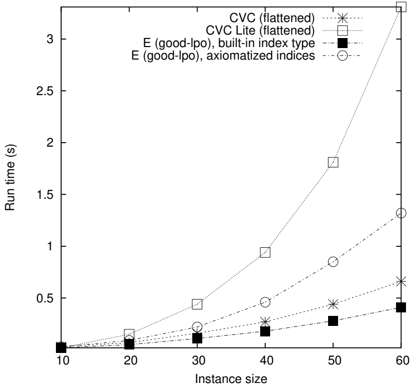
Flat input: E matches CVC

Performances on invalid `STORECOMM`(n) instances



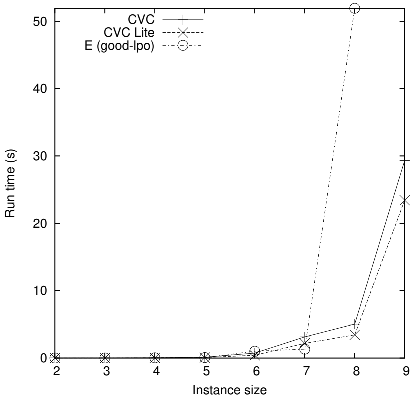
Native input: prover conceived for unsat handles sat even better

Performances on invalid `STORECOMM`(n) instances



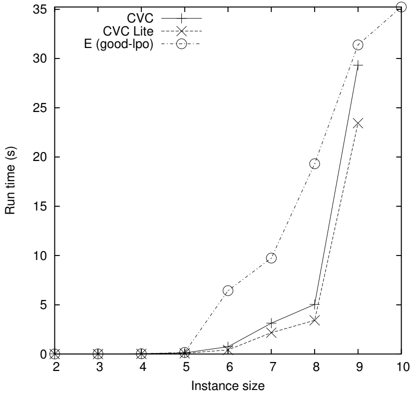
Flat input: E surpasses CVC

Performances on valid `SWAP`(n) instances



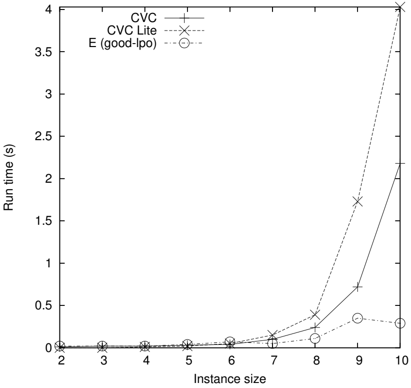
Harder problem: no system terminates for $n \geq 10$

Performances on valid `SWAP`(n) instances



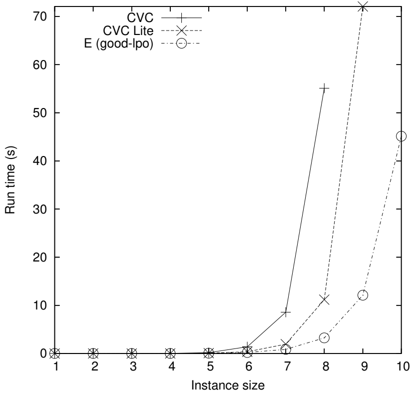
Added lemma for E: additional flexibility for the prover

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



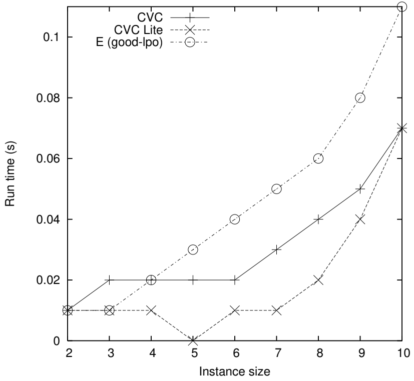
Easier problem, but E clearly ahead

Performances on valid `STOREINV`(n) instances



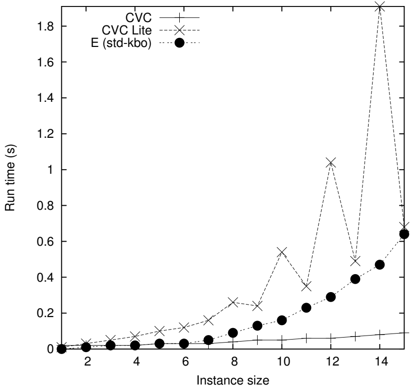
E(std-kbo) does it in *nearly constant time*!

Performances on invalid `STOREINV`(n) instances



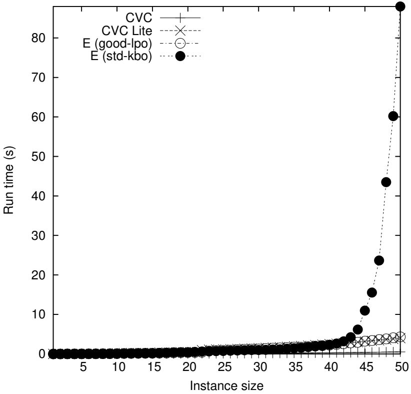
Not as good for E but run times are minimal

Performances on `IOS` instances



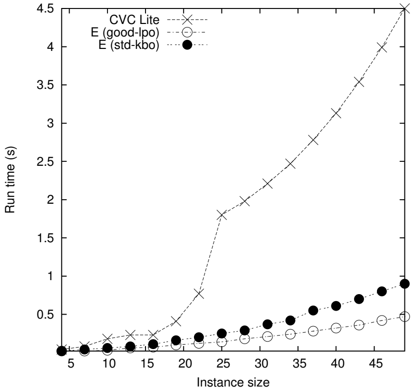
CVC and CVC Lite have built-in $\mathcal{LA}(\mathcal{R})$ and $\mathcal{LA}(\mathcal{I})$ respectively!

Performances on `QUEUE` instances (plain queues)



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

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



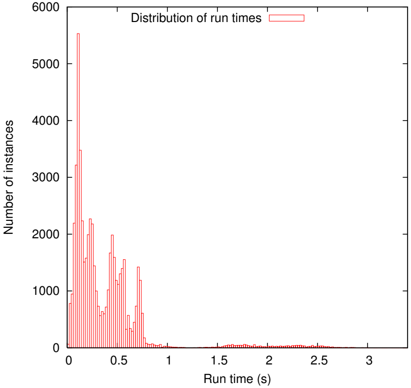
CVC does not handle integers mod k, E clearly wins

"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

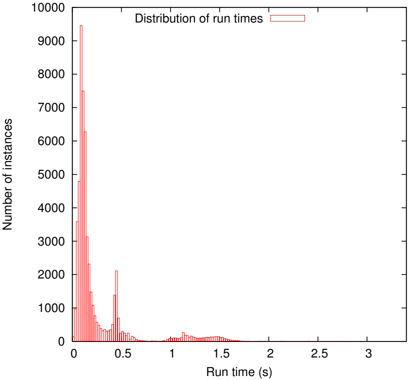
Test performance on huge sets of literals.

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 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 $\mathcal{LA}$ or bit-vectors
- $\mathcal{T}$-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
- … … …