Automated reasoning for verification: recent results and current challenges[1]

Maria Paola Bonacina

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

Talk given at the Dept. of Mathematical Sciences, Tsinghua University, Beijing, P.R. China

21 May 2007

---

[1] Partly joint work with Mnacho Echenim

Verification problems

A variety of verification problems:

- Microprocessor verification
- Program verification: proving a program correct
- Program analysis: proving a program free of certain bugs
- Hybrid or reactive systems: to be proved safe or deadlock-free

Reasoning-based verification systems

- HW/SW Model checkers,
- Program analyzers,
- Proof assistants,
- Interactive theorem provers,
- ...

Common architecture:

- *Front-end*: interface and problem modelling
- *Back-end reasoner*: problem solving

Our focus: the **back-end reasoner**

Problems for the back-end reasoner

- $\mathcal{T}$**-decision problem**: to decide validity of a ground formula modulo a background theory $\mathcal{T}$
- **Objective**: $\mathcal{T}$-decision procedure
- **Desiderata**:
  - *Efficient* (it's only a sub-task of the verification task)
  - *Scalable* (practical problems are huge: 1MB for one formula)
  - *Proof-producing* (so that the proof can be checked)
  - *Model-producing* (a model is a counter-example: bug finding)
  - *Expressive* (propositional logic not enough: equality, arithmetic, data structures)
- Reasoning in **combinations of theories** is crucial

Theories

- Linear arithmetic on the integers, on the reals
- Theories of data structures, e.g.:
  - Lists
  - Arrays (e.g., to model registers in microprocessors)
  - Functions
  - Sets
  - Records
  - Bitvectors

Very common theories in verification problems.

Example: The theory of lists

Without nil:

$$\forall x, y.\ \mathrm{car}(\mathrm{cons}(x, y)) \simeq x \tag{1}$$

$$\forall x, y.\ \mathrm{cdr}(\mathrm{cons}(x, y)) \simeq y \tag{2}$$

$$\forall y.\ \mathrm{cons}(\mathrm{car}(y), \mathrm{cdr}(y)) \simeq y \tag{3}$$

With nil: replace (3) by

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

Example: The theory of records

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

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

where x and y have sort REC and v has sort T_i.
The third axiom is the *extensionality* axiom.

Examples of problems

- $x \leq y \wedge y \leq x + car(cons(0, x)) \wedge P(h(x) - h(y)) \wedge \neg P(0)$
- $store(v, i, e)[k] \simeq x \wedge v[k] \simeq f \wedge (x \leq e \vee x \leq f)$
- $i_1 \not\simeq i_2 \supset$
  $store(store(a, i_1, e_1), i_2, e_2) \simeq store(store(a, i_2, e_2), i_1, e_1)$

Tiny examples that can be done by hand

Reasoning refutationally

- Deciding $\mathcal{T}$-validity of φ by deciding $\mathcal{T}$-unsatisfiability of $\neg\varphi$
- *$\mathcal{T}$-satisfiability procedure*: sets of ground unit clauses
- *$\mathcal{T}$-decision procedure*: sets of ground clauses

Current approaches

- "Little" engines of proof (SMT-solvers with $\mathcal{T}$ built-in):
  - Eager approach
  - Lazy/Hybrid approach
- "Big" engines of proof (FOL+= provers with $\mathcal{T}$ as input):
  - Hierarchic approach
  - Direct approach

Eager approach

- *Idea*: reduce to SAT and apply SAT-solver

  [Bryant, Velev 2001] [Bryant, Lahiri, Seshia 2002] [Meir, Strichman 2005]
- *Advantage*: efficiency of SAT-solvers (e.g., DPLL-based)

  [Davis, Putnam, Logemann, Loveland 1962] [Zhang: SATO 1997] [Malik et al.: Chaff 2001]
- *Open problems*:
  - Growth of the formula: even $O(n^2)$ reduction not good enough
  - Proof generation?
  - Model generation?
  - No reasoning with quantified variables

Systems: UCLID, Alloy

Lazy/hybrid approach

- *Idea*: integration of SAT-solver and $\mathcal{T}$-solver

  [Barrett, Dill, Stump 2002] [de Moura, Rueß, Sorea 2002]
  - Congruence closure algorithm for equality
  - Nelson-Oppen scheme to combine $\mathcal{T}$-sat procedures

    [Shostak 1978] [Nelson, Oppen 1979] [Downey, Sethi, Tarjan 1980] [Nelson, Oppen 1980]
- *Advantages*: efficient SAT-solvers, *built-in* theories
- *Open problems*:
  - Difficult balance of SAT-reasoning and $\mathcal{T}$-reasoning
  - Ad hoc combination of theories
  - Proof and model generation?
  - No reasoning with quantified variables (only heuristics)

Systems: Simplify; CVC, CVCLite, CVC3; ICS, Simplics, Yices; ZAP; Ario; MathSAT; Barcelogic

Hierarchic approach

- *Idea*: combination of theories as extension of theories, total/partial functions [Ganzinger, Sofronie, Waldmann 2006]
- *Advantages*: native quantifier reasoning, locality (only certain instances needed),
- *Disadvantages or open problems*:
  - Need to change semantics ("undefined" values)
  - Need to change inference system, new completeness proof
  - Need to change/redo implementation
  - Model generation?

Systems: SPASS+$\mathcal{T}$ (only in part)

Direct approach

- *Idea*: if inference system $\mathcal{I}$ *terminates* on $\mathcal{T}$-sat problems, a fair $\mathcal{I}$-strategy is *decision procedure* for $\mathcal{T}$-sat

  [Armando, Ranise, Rusinowitch 2003]

  Early forerunner: Knuth-Bendix completion for ground equality [Lankford 1975]
- *Advantages*:
  - No need of new ad hoc proofs ($\mathcal{I}$ is sound and complete)
  - Combination of theories: give union of presentations as input
  - No implementation effort: take prover "off the shelf"
  - Proof generation: already there by default
  - Native quantifier reasoning

Issues with the direct approach

1. Combination of theories: give general *modularity* result to avoid having to prove termination for each combination
2. Generalize from $\mathcal{T}$*-satisfiability* to $\mathcal{T}$*-decision* problems
3. Handle theories such as arithmetic or bitvectors that do not lend themselves to deduction
4. Experimental evidence of *efficiency* and *scalability*
5. *Model generation*: final $\mathcal{T}$-sat set as starting point

Topics 1, 2, 3, 4: this talk
Topic 5: future work

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

A few preliminaries

Good ordering: $t \succ c$ for all compound terms t and constants c

$depth(t) = 0$ if t is a constant or variable
$depth(t) = 1 + max\{depth(t_i) : 1 \leq i \leq n\}$ otherwise

$depth(l \bowtie r) = depth(l) + depth(r)$

A positive literal is *flat* if its depth is 0 or 1
A negative literal is *flat* if its depth is 0
A literal is *strictly flat* if its depth is 0

A clause is *flat* (*strictly flat*) if all its literals are

Flattening

Input: finite set of ground Σ-clauses S
Output: finite set of ground Σ'-clauses $S_1 \uplus S_2$

- Σ' is Σ + finitely many additional constants
- S_1: unit flat clauses
- S_2: strictly flat clauses
- $\mathcal{T} \cup S$ and $\mathcal{T} \cup S_1 \cup S_2$ equisatisfiable

$\mathcal{T}$-satisfiability problem: $S_2 = \emptyset$

Example

$$S = \{f(a) \not\simeq f(b) \vee f(a) \not\simeq f(c)\}$$

$$S_1 = \{f(a) \simeq a',\ f(b) \simeq b',\ f(c) \simeq c'\}$$

$$S_2 = \{a' \not\simeq b' \vee a' \not\simeq c'\}$$

where a', b', c' are fresh constants

Rewrite-based $\mathcal{T}$-satisfiability procedures for

- Lists
  - *non-empty possibly cyclic*
  - *possibly empty possibly cyclic*
- *Arrays*, *sets* and *records* with or without extensionality
- Fragments of linear arithmetic:
  - *integer offsets*
  - *integer offsets modulo*
- *Recursive data structures* with one constructor and k selectors:
  - $k = 1$: integer offsets (*pred* and *succ*)
  - $k = 2$: non-empty acyclic lists (*cons*, *car* and *cdr*)

[Armando, Ranise, Rusinowitch 2003] [Armando, Bonacina, Ranise, Schulz 2005] [Bonacina, Echenim 2006]

Modularity of termination for combination of theories

Modularity of termination:
if $\mathcal{SP}_{\succ}$-strategy decides $\mathcal{T}_i$-sat problems then it decides $\mathcal{T}$-sat problems for $\mathcal{T} = \bigcup_{i=1}^{n} \mathcal{T}_i$

Standard hypothesis:
the $\mathcal{T}_i$ do not share function symbols (shared constants allowed)

Variable-inactivity

Clause C *variable-inactive*: no maximal literal in C is equation $t \simeq x$ where $x \notin Var(t)$

Set of clauses *variable-inactive*: all its clauses are

$\mathcal{T}$ *variable-inactive*: the limit $S_\infty = \bigcup_{j \geq 0} \bigcap_{i \geq j} S_i$ of a fair derivation from $\mathcal{T} \cup S$ is variable-inactive

Examples

$$\begin{aligned} C_1 &= \mathrm{car}(\mathrm{cons}(x, y)) \simeq x \\ C_2 &= z \simeq w \vee \mathrm{select}(\mathrm{store}(x, z, v), w) \simeq \mathrm{select}(x, w) \\ C_3 &= \bigvee_{1 \leq j < k \leq n} (x_j \simeq x_k) \end{aligned}$$

C_1 variable-inactive
C_2 variable-inactive
C_3 not variable-inactive

The modularity theorem

Theorem: if

- $\mathcal{T}_i$, $1 \leq i \leq n$, variable-inactive
- fair $\mathcal{SP}_{\succ}$-strategy is $\mathcal{T}_i$-sat procedure, $1 \leq i \leq n$,

then it is a $\mathcal{T}$-satisfiability procedure.

Intuition:

- No shared function symbol: no paramodulation from compound terms across theories
- Variable-inactivity: no paramodulation from variables across theories, since for $t \simeq x$ where $x \in Var(t)$ it is $t \succ x$

All above mentioned theories satisfy the hypotheses of the theorem.

[Armando, Bonacina, Ranise, Schulz 2005]

From $\mathcal{T}$-satisfiability to $\mathcal{T}$-decision

Key observation: inferences between variable-inactive and strictly flat clauses: only paramodulations from constants into constants

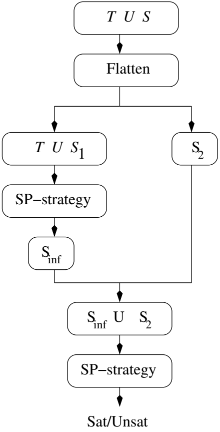
Theorem: if

- $\mathcal{T}$ is variable inactive
- $\mathcal{SP}_{\succ}$-strategy is $\mathcal{T}$-sat procedure

then it is also $\mathcal{T}$-decision procedure

[Bonacina, Echenim 2007]

A "pure" approach based on variable-inactivity



Another approach: $\mathcal{T}$-decision by decomposition

- **Problem**:
  - FOL$+=$ provers are not as efficient as SAT-solvers on the Boolean part
  - Integration of FOL$+=$ prover as $\mathcal{T}$-procedure and SAT-solver: either not tight or too complicated
- **Solution**:
  - decompose $\mathcal{T}$-decision problem
  - solve it *by stages*, by pipe-lining FOL$+=$ prover and SMT-solver

Preliminaries

Decomposition: e.g., flattening, where S is decomposed into S_1 and S_2; it suffices that S_1 be made of *flat unit clauses*

$\mathcal{T}$-compatibility: S is $\mathcal{T}$-compatible with A if A entails every clause generated from premise in S and premise in $\mathcal{T}$

Instance of $\mathcal{T}$-compatibility: S is $\mathcal{T}$-compatible with $\bar{S}$ where $S_\infty = \mathcal{T} \cup \bar{S}$ is the limit generated by the inference system from $\mathcal{T} \cup S$

$\mathcal{T}$-stability: ensures that $\mathcal{T}$-compatibility is preserved by all inferences in the inference system

$\mathcal{T}$-decision by stages: the theorem

Theorem: under $\mathcal{T}$-stability, if A and A' are sets of clauses such that

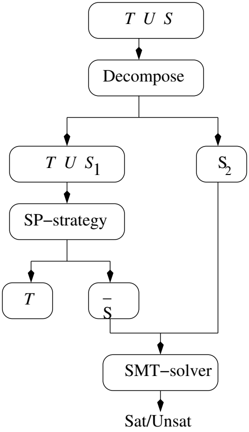
- $\mathcal{T} \cup S_1 \models A$
- $\mathcal{T} \cup S_2 \models A'$
- S_1 is $\mathcal{T}$-compatible with A
- S_2 is $\mathcal{T}$-compatible with A'

then $\mathcal{T} \cup S_1 \cup S_2$ and $A \cup A'$ are equisatisfiable.

Instance of the theorem: $A \leftarrow \bar{S}$ and $A' \leftarrow S_2$ where $S_\infty = \mathcal{T} \cup \bar{S}$ is the limit generated by the inference system from $\mathcal{T} \cup S_1$

[Bonacina, Echenim 2007]

$\mathcal{T}$-decision by stages: the scheme



Handling arithmetic

How about theories such as arithmetic or bitvectors that do not lend themselves to deduction?

This part of the problem can be left into S_2 and passed on directly to the SMT-solver.

SMT-solvers typically features very fast implementation of the *simplex algorithm* for linear arithmetic.

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 ...

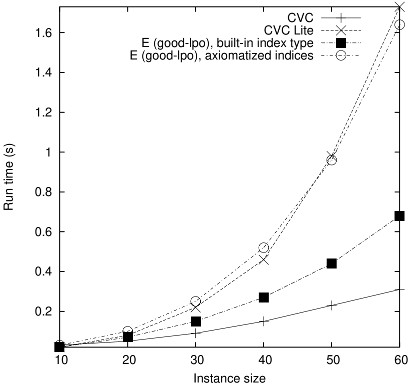
Synthetic benchmarks

- `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

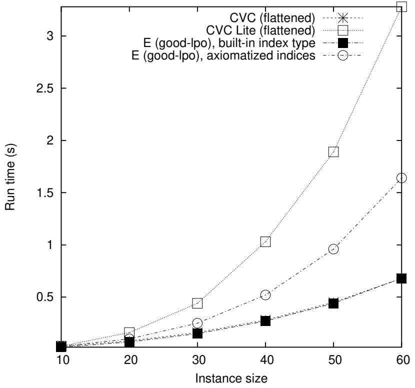
Parameter n: test *scalability*

Performances on valid `STORECOMM`(n) instances



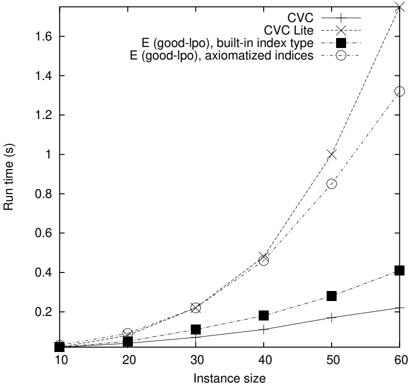
Native input: CVC wins but E better than CVC Lite

Performances on valid `STORECOMM`(n) instances



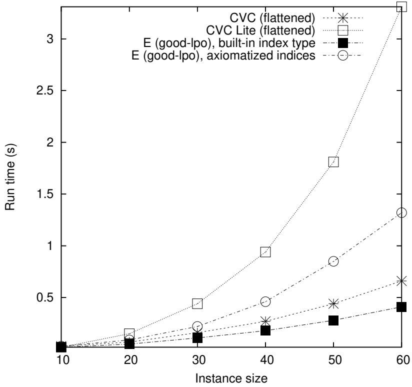
Flat input: E matches CVC

Performances on invalid `STORECOMM(n)` instances



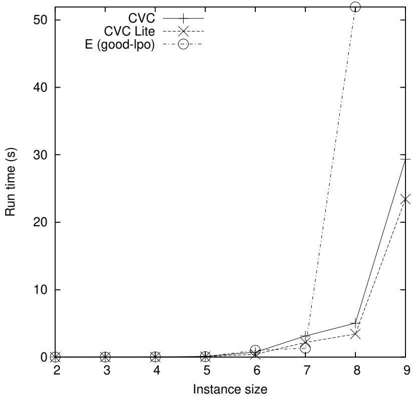
Native input: prover conceived for unsat handles sat even better

Performances on invalid `STORECOMM`(n) instances



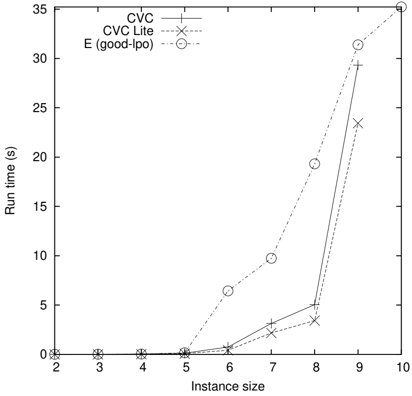
Flat input: E surpasses CVC

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



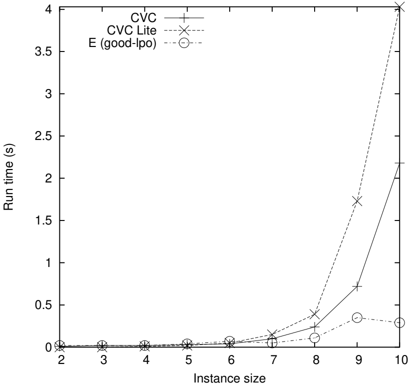
Harder problem: no system terminates for $n \geq 10$

Performances on valid `SWAP`(n) instances



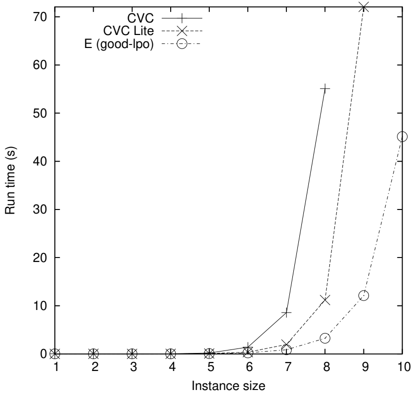
Added lemma for E: additional flexibility for the prover

Performances on invalid `SWAP`(n) instances



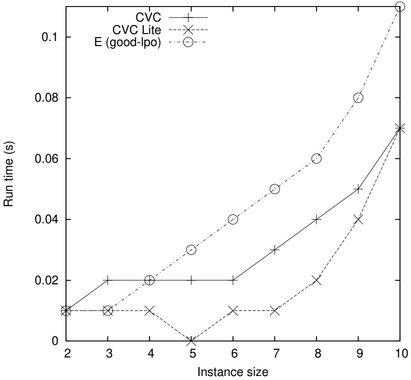
Easier problem, but E clearly ahead

Performances on valid `STOREINV`(n) instances



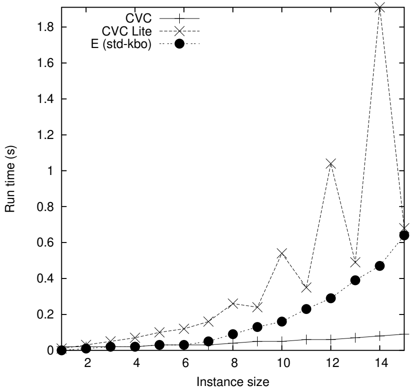
E(std-kbo) does it in *nearly constant time*!

Performances on invalid `STOREINV`(n) instances



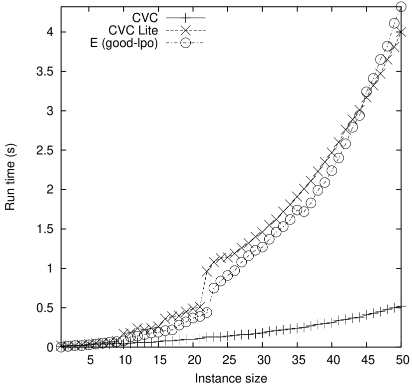
Not as good for E but run times are minimal

Performances on `IOS` instances



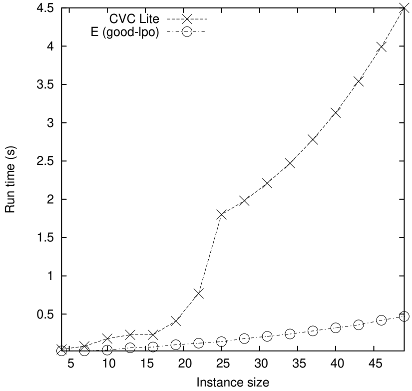
CVC and CVC Lite have built-in $\mathcal{LA}(\mathcal{R})$ and $\mathcal{LA}(\mathcal{I})$ respectively!

Performances on `QUEUE` instances (plain queues)



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

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



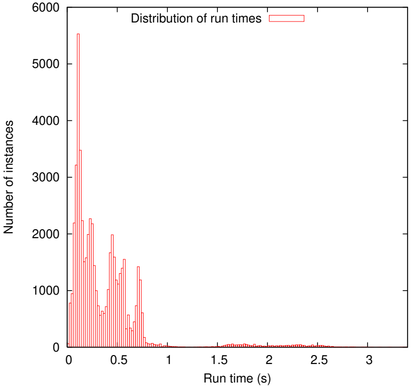
CVC does not handle integers mod k, E clearly wins

"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

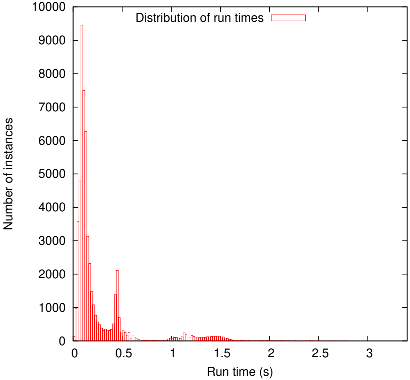
Test performance on huge sets of literals.

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

- *Uniform methodology* for rewrite-based $\mathcal{T}$-sat procedures
- *Modularity theorem* for combination of theories
- *Generalization* to $\mathcal{T}$-decision procedures:
  - A "pure" approach
  - A decomposition approach that pipe-lines prover and SMT-solver
- Experiments on $\mathcal{T}$-sat problems:
  a prover *taken off the shelf* and conceived for very different search problems compares amazingly well with built-in theories validity checkers

Current and future work

- Experiments with $\mathcal{T}$-decision problems
- Search plans for $\mathcal{T}$-sat and $\mathcal{T}$-decision problems
- Combination with automated model building:
  AMB for theory-reasoning