Deciding satisfiability problems by general-purpose deduction: Experiments in the theory of arrays

Maria Paola Bonacina, Dip. Informatica, Università degli Studi di Verona, Italy

Joint work with:
Alessandro Armando, DIST, Università degli Studi di Genova, Italy
Silvio Ranise, LORIA & INRIA-Lorraine, Nancy, France
Michaël Rusinowitch, LORIA & INRIA-Lorraine, Nancy, France
Aditya Kumar Sehgal, Dept. of Computer Science, U. Iowa, USA

Outline

- Motivation
- Background on satisfiability procedures
- A deduction-based approach
- Theory of arrays : synthetic benchmarks
- Experimental results with E and CVC
- Discussion of results and research directions

Motivation

- HW/SW verification requires reasoning with theories of data types, e.g., integer, real, arrays, lists, trees, tuples, sets.
- E.g., use arrays to model registers and memories in formalizing HW verification problems.
- Some of these theories are decidable.
- Built-in theories for verification tools and proof assistants.

Some background on

Satisfiability Procedures

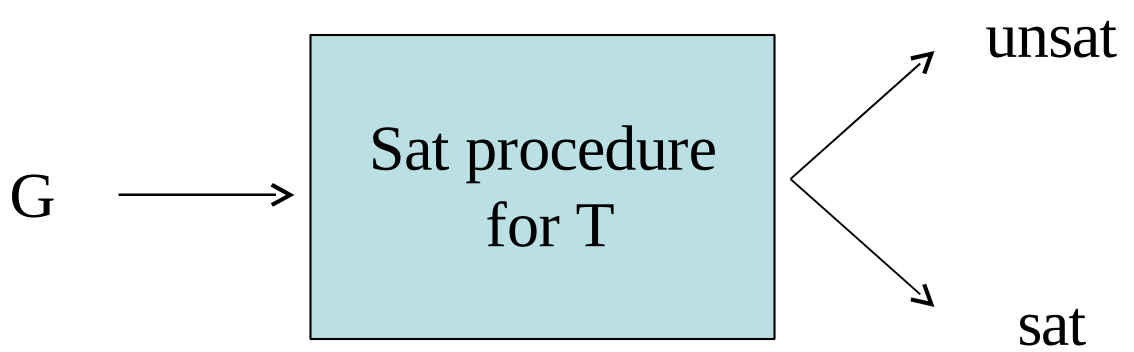
Satisfiability procedure

T : background theory, possibly with intended interpretation

φ : quantifier-free formula

φ' : DNF (¬φ)   /* not in practice : more later */

G : conjunction (set) of ground literals from φ'



Common approach

Design, prove sound and complete, and implement a satisfiability procedure for each decidable theory of interest.

Issues:

- Most problems involve multiple theories: combination of theories / procedures [Nelson-Oppen, Shostak, …]
- Proofs for concrete procedures [e.g., Shankar, Stump] or abstract frameworks [e.g., Tiwari, Ganzinger]
- Implement from scratch data structures and algorithms for each procedure: correctness of implementation? SW reuse?

Un-interpreted and interpreted symbols

- Un-interpreted function and predicate symbols :
  all properties are stated by axioms
  Conjunction of ground equalities and inequalities with only un-interpreted function symbols : congruence closure [Nelson-Oppen, Downey-Sethi-Tarjan 1980]

- Interpreted function and predicate symbols :
  built-in properties, e.g.,from $x - 1 = y$ to $x = y + 1$
  Conjunction of ground equalities and inequalities including interpreted symbols: combination of congruence closure and specialized procedures

Combination of theories : Nelson-Oppen 1979

$T_1 .. T_k$ $G_1 .. G_k$

- Disjoint theories : if r_i occurs in G_j rename as x and add $x = r$ to G_i
- Communication among procedures : only equalities between variables
- Convex theories :

if $T \models \bigvee_{i=1..n} s_i = r_i$ then $T \models s_i = r_i$ for some i

Non-convex theories : splitting of disjunctions

Combination : Shostak 1984

[Cyrluk-Lincoln-Shankar, CADE 1996]
[Ruess-Shankar, LICS 2001]

- Theories with canonical form :

  $T \models s = r$ iff $\sigma(s) = \sigma(r)$

  $vars(\sigma(s)) \subseteq vars(s)$ and $\sigma(x) = x$

  $\sigma(\sigma(s)) = \sigma(s)$

  if $\sigma(s) = f(s_1 .. s_n)$ then $\sigma(s_i) = s_i$ $i = 1 \ldots n$

  Canonical form is representative of equivalence class

- And algebraically solvable :

  solve (s = r) is in tree normal form (idempotent substitution)

Relation to general deduction

- Congruence closure : ground completion
- Algebraic solvability : semantic unification
  solve ( ) : T- unification algorithm
- Canonical form : T- normal form

Abstract frameworks

- Abstract Congruence Closure

[Tiwari 2000] [Bachmair-Tiwari-Vigneron, JAR 2002]
CC algorithms as ground completion strategies with same inference rules (including flattening) and different search plans
Ground completion with indexing is fastest !

- Shostak Light

[Barrett-Dill-Stump, FroCoS 2002]
[Ganzinger, CADE 2002]
Shostak as completion modulo T_i (T_i - unification) or
Nelson-Oppen combination of CC and T_i - unification

General-purpose ordering-based theorem-proving :

Can it help ?

Theorem proving would help:

- Combination of theories: give union of the axiomatizations in input to the prover
- No need of ad hoc proofs for each procedure
- Reuse code of existing provers

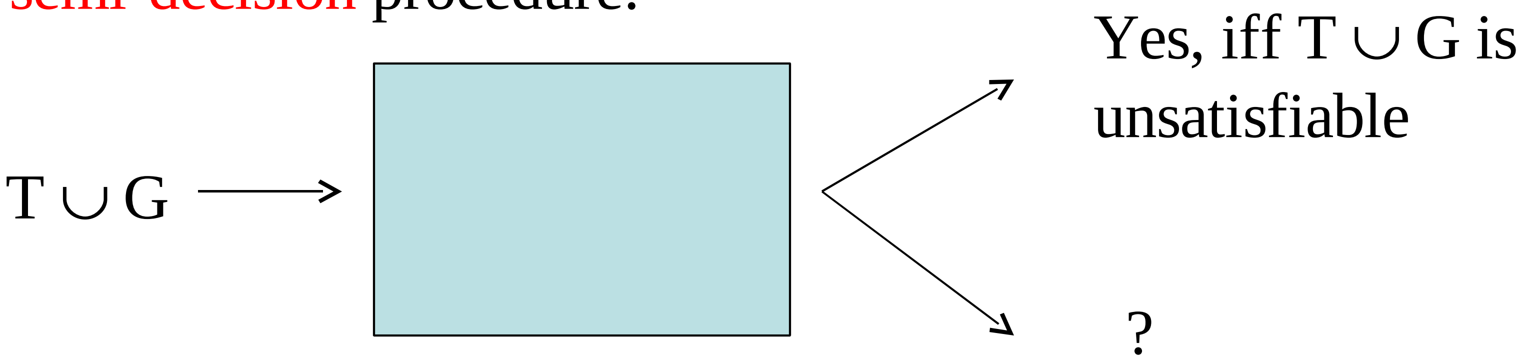
Termination ?

$C = < I, \Sigma >$: theorem-proving strategy

I : refutationally complete inference system with superposition/ paramodulation, simplification, subsumption …

Σ: fair search plan

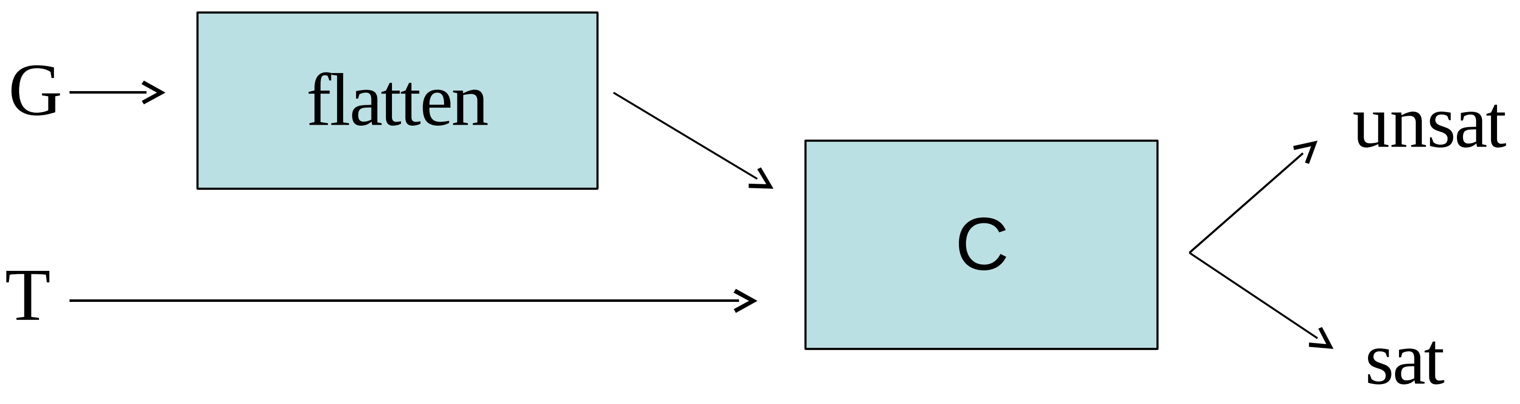
is a semi-decision procedure:



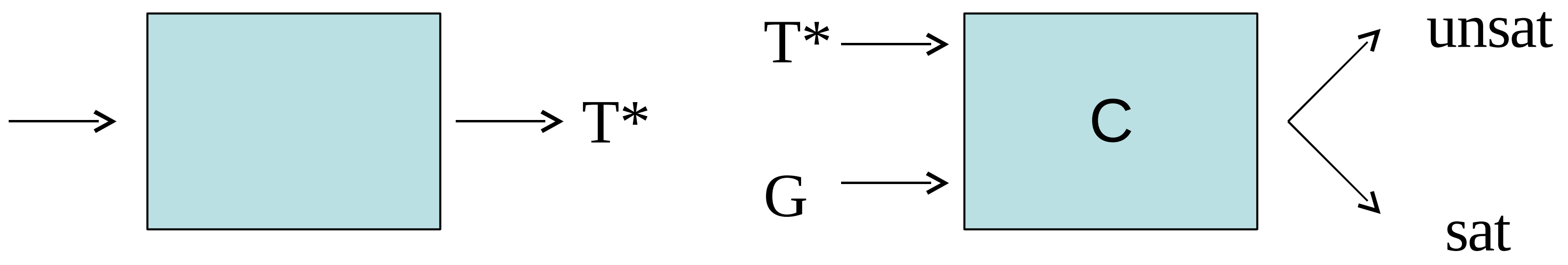
Termination results

[Armando, Ranise, Rusinowitch, CSL 2001]

T: theory of arrays, lists, sets and combinations thereof



Another way to put it



Pure equational: T* canonical rewrite system

Horn equational: T* saturated ground-preserving
[Kounalis & Rusinowitch, CADE 1988]

FO special theories: e.g., T = T* for arrays [ARR, CSL 2001]

Beyond conjunctions of literals

[Silvio Ranise, UNIF 2002]

Extension to arbitrary quantifier-free formulae including connectives if_then_else_ and let_in_ that are very useful in verification problems :

- transformation to clauses such that at most one literal is an equality of ground first-order flat terms and the other literals are propositional variables
- extension of termination results

How about efficiency ?

A satisfiability procedure with T built-in is expected to be always much faster than a theorem prover with T in input !

Totally obvious ? Or worth investigating ?

- theory of arrays
- synthetic benchmarks (allow to assess scalability by experimental asymptotic analysis)
- comparison of E prover and CVC validity checker with theory of arrays built-in

Theory of arrays: the signature

$$\text{store} : \text{array} \times \text{index} \times \text{element} \to \text{array}$$

$$\text{select} : \text{array} \times \text{index} \to \text{element}$$

The presentation (T_1)

(1) $\forall$A, I, E. select (store (A, I, E), I) = E

(2) $\forall$A, I, J, E. I $\neq$ J $\Rightarrow$
select (store (A, I, E), J) = select (A, J)

(3) Extensionality: $\forall$A, B.
$\forall$I. select (A, I) = select (B, I)
$\Rightarrow$
A = B

Pre-processing extensionality

$$select\ (A, sk\ (A, B)) \neq select\ (B, sk\ (A, B)) \lor A = B$$

$$t \neq t'$$

$$select\ (t, sk\ (t, t')) \neq select\ (t', sk\ (t, t'))$$

Another presentation (T_2)

Keep (1) and (2) and replace extensionality (3) by:

(4) $\forall A, I.\ store(A, I, select(A, I)) = A$

(5) $\forall A, I, E, F.$
$store(store(A, I, E), I, F) = store(A, I, F)$

(6) $\forall A, I, J, E.\ I \neq J \Rightarrow$
$store(store(A, I, E), J, F) = store(store(A, J, F), I, E)$

T_1 entails (4) (5) (6)

Use of presentations

- T_1 is saturated and application of C to

$T_1 \cup G$ is guaranteed to terminate [ARR2001]:
C acts as decision procedure

- T_2 is not saturated (saturation does not halt):

C applied to $T_2 \cup G$ acts as semi-decision procedure

Two sets of synthetic benchmarks

in array theory

storecomm(N): intuition

Storing values at distinct places in an array is “ commutative”

storecomm(N) : definition

$k_1 \ldots k_N$: N indices

D : set of 2-combinations over { 1 … N }

Indices must be distinct:

$$\bigwedge_{(p, q) \in D} k_p \neq k_q$$

$i_1 \ldots i_N$, $j_1 \ldots j_N$: two distinct permutations of 1 … N

$$\text{store}\,(\ldots(\,\text{store}\,(\,a, k_{i_1}, e_{i_1}\,), \ldots k_{i_N}, e_{i_N}\,)\ldots)$$

$$=$$

$$\text{store}\,(\ldots(\,\text{store}\,(\,a, k_{j_1}, e_{j_1}\,), \ldots k_{j_N}, e_{j_N}\,)\ldots)$$

storecomm(N) : schema

$$\bigwedge_{(p,\, q) \in D} k_p \neq k_q$$

$$\Rightarrow$$

$$\text{store}(\ldots(\text{store}(a, k_{i1}, e_{i1}), \ldots k_{iN}, e_{iN}) \ldots)$$

$$=$$

$$\text{store}(\ldots(\text{store}(a, k_{j1}, e_{j1}), \ldots k_{jN}, e_{jN}) \ldots)$$

storecomm(N) : instances

Each choice of permutations generates a different instance:

N! permutations of the indices

The number of instances is the number of 2-combinations of N! permutations:

$$N! \, (N! - 1) / 2$$

swap(N): intuition

Swapping pairs of elements in an array
in two different orders yields the same array

swap(N) : definition

Recursively:

Base case: N = 2 elements:

L_2 = store (store (a, i_1, select (a, i_0)), i_0, select (a, i_1))

R_2 = store (store (a, i_0, select (a, i_1)), i_1, select (a, i_0))

$$L_2 = R_2$$

Recursive case: N = k+2 elements:

L_{k+2} = store (store (L_k, i_{k+1}, select (L_k, i_k)), i_k, select (L_k, i_{k+1}))

R_{k+2} = store (store (R_k, i_k , select (R_k, i_{k+1})), i_{k+1}, select (R_k, i_k))

$$L_{k+2} = R_{k+2}$$

swap(N) : instances

N elements, N/2 pairs to exchange

N! permutations of the elements

C_i : number of i-combinations over the set of N/2 pairs

number of ways of picking i pairs for exchange

$$\Sigma_i C_i = 2^{(N/2)} - 1$$

Number of instances: $1/2 \times N! \times (2^{(N/2)} - 1)$

Experiments

with E and CVC

Set up of the experiments

- Two tools: CVC validity checker and E theorem prover
- E: auto mode and user-selected strategy
- Comparison of asymptotic behavior of E and CVC as N grows

The CVC validity checker

[Aaron Stump, David L. Dill et al., Stanford U.]

Combines procedures à la Nelson-Oppen
(e.g., lists, arrays, records, real arithmetics …).

Has SAT solver: first GRASP then Chaff

Theory of arrays: ad hoc algorithm based on congruence closure with pre-processing wrt. axioms of T_1 and elimination of " store" via partial equations

Why a SAT solver ?

To handle arbitrary quantifier-free formulae :

cooperation of SAT solver and decision procedure(s)

Map first-order formula φ to propositional abstraction abs (φ)

abs (φ) unsatisfiable : φ unsatisfiable

abs (φ) satisfiable : model α yields

either model of φ (checked by decision procedure)

or minimal conflict clause to feed back to SAT solver

Interaction is incremental

The E theorem prover

[Stephan Schulz, TU-Muenchen]

Inference system I : o-superposition/paramodulation, reflection, o-factoring, simplification, subsumption

Search plans Σ :

- given-clause loop with clause selection functions and only "already-selected" list inter-reduced
- term orderings: KBO and LPO
- literal selection functions

Strategies in experiments

- E-auto: automatic mode
- E-SOS: { problem in form $T \cup G$ }

  Clause selection: (SimulateSOS,RefinedWeight)

  Term ordering: LPO
- Precedence: select > store > sk > constants

First set of experiments on storecomm(N)

E takes presentation T_1 in input

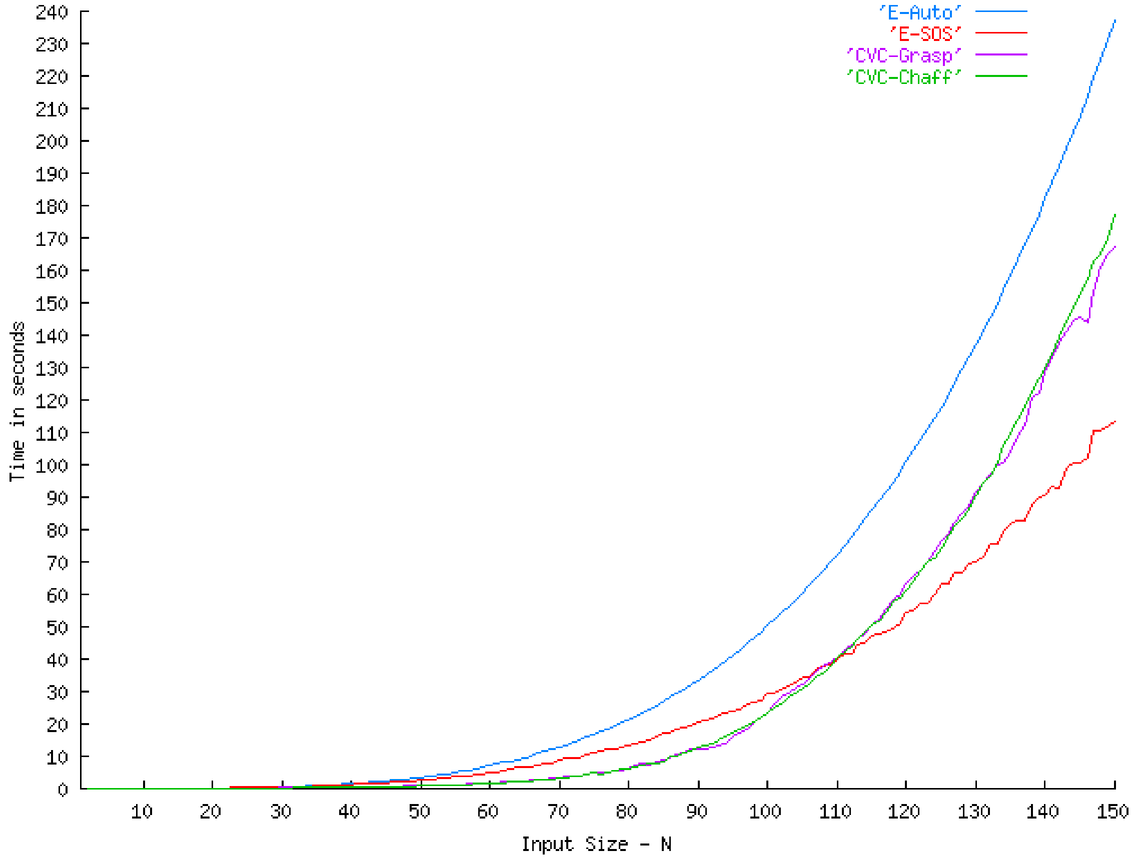
N ranges from 2 to 150

Sample 10 permutations: 45 instances for each value of N
Non-uniform sampling (favors permutations with local changes)

Performance for N is average over all generated instances for value N

Versioni : E 0.62
CVC/GRASP Fall 2001, CVC/CHAFF January 2002

First set : storecomm(N)



Second set of experiments on storecomm(N)

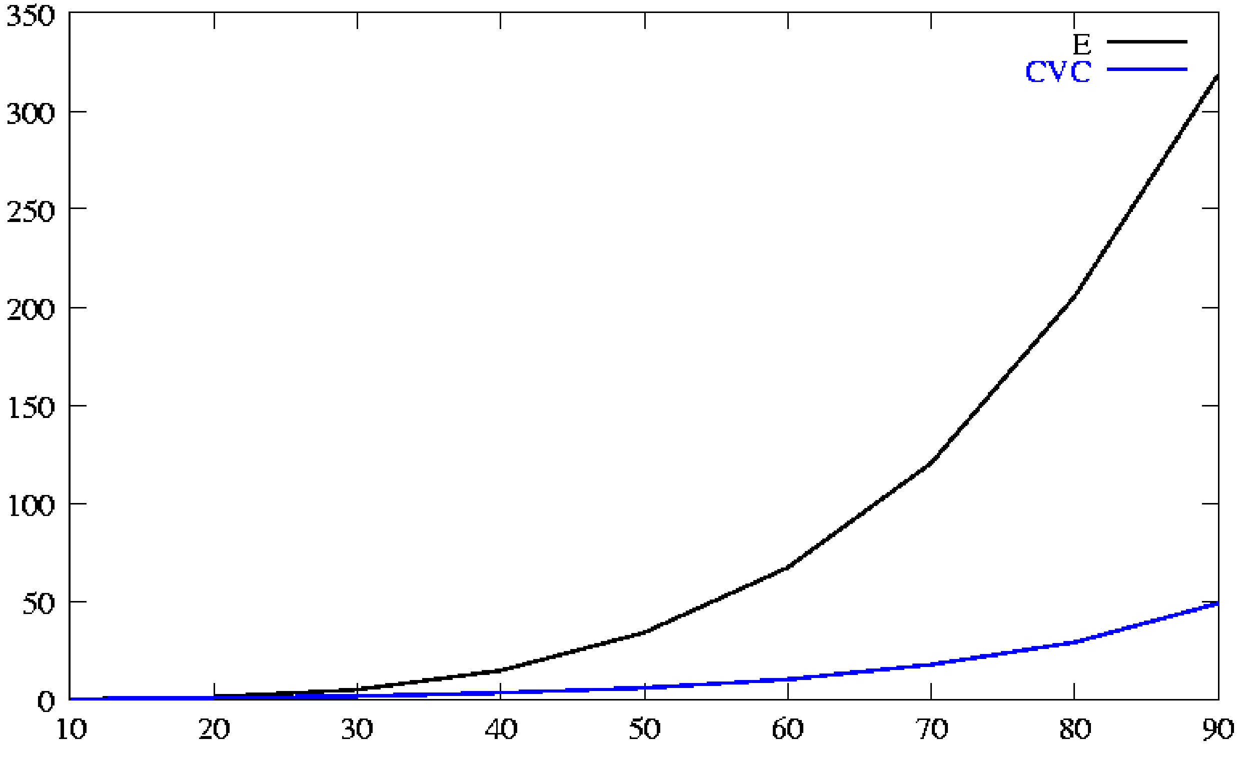
E takes presentation T_1 in input

N ranges from 2 to 90
For each value of N pick one instance at random :
no averages

Only E-auto, E-SOS do not help

Versioni : E 0.62
CVC/CHAFF October 2002

Second set : storecomm(N)



First set of experiments on swap (N)

Sample up to 16 permutations and 20 instances for each value of N

Non-uniform sampling (favors permutations with local changes)

Performance for N is average over all generated instances for value N

CVC: does up to N = 10, runs out of memory on any instance of swap(12)

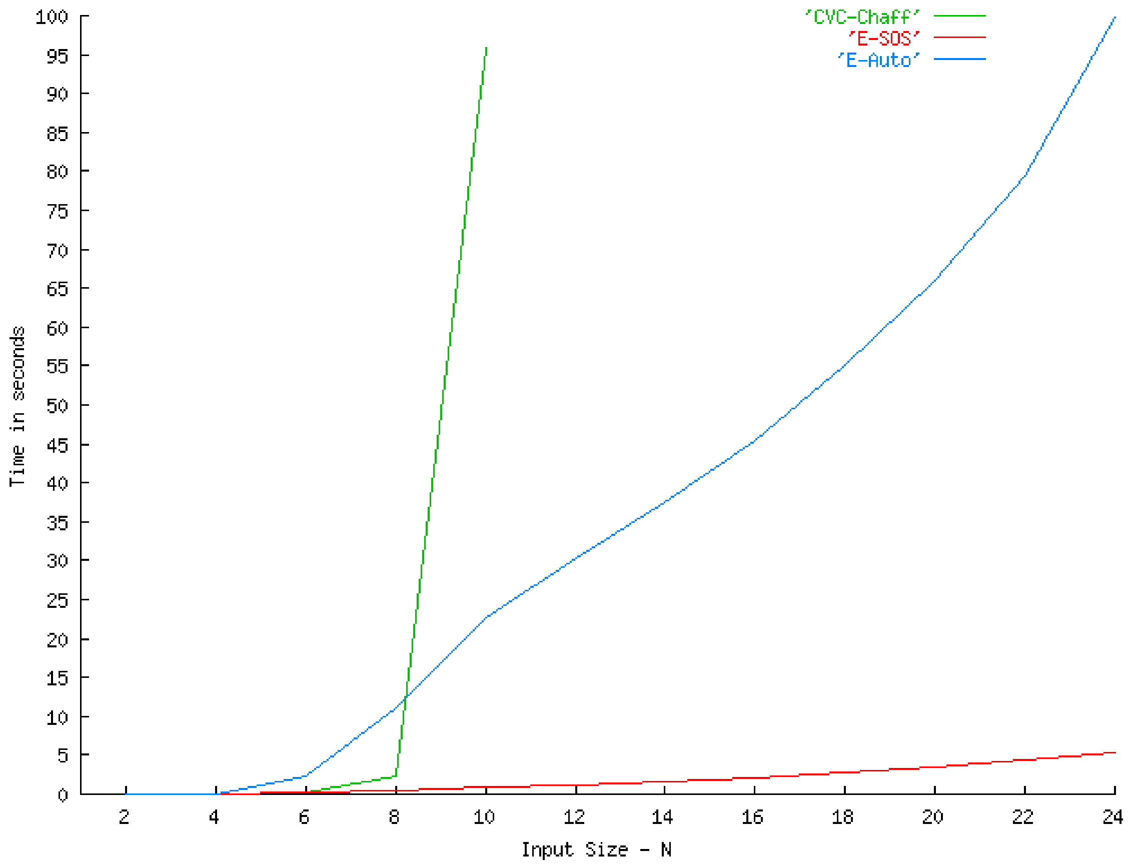
E with presentation T_1: same as above and slower

E with presentation T_2: succeeds also for $N \geq 12$

Versioni : E 0.62

CVC/GRASP Fall 2001, CVC/CHAFF January 2002

First set : swap(N)



Second set of experiments on swap(N)

E takes presentation T_1 in input

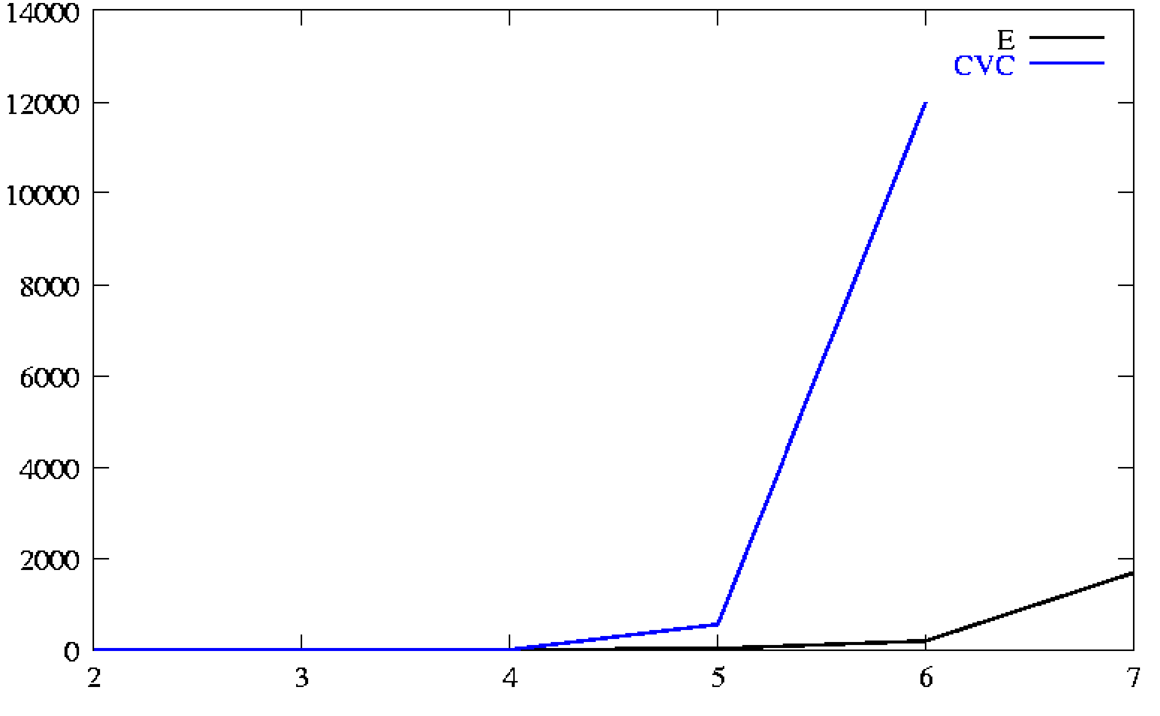
For each value of N pick one instance at random : no averages

Only E-auto, E-SOS don' t help

CVC : does only up to N = 6, E goes beyond

Versioni : E 0.62
CVC/CHAFF October 2002

Second set : swap(N)



Discussion

- Need more experiments: other synthetic benchmarks, other theories, combination of theories, real-world problems
- Understand role of flattening better
- Other provers, e.g., w. more inter-reduction
- Termination results for other theories?
- Complexity of concrete strategies on specific theories

Discussion

- Deduction may help build better decision procedures
- Integration of automated theorem proving and automated model building
- ATP needs more work on auto mode and search plans (search, not blind saturation)
- Proof assistants incorporate satisfiability procedures: integration of ATP/AMB in proof assistants