Deciding satisfiability problems by rewrite-based deduction: Experiments in the theory of arrays

Maria Paola Bonacina, Dept. of Computer Science, U. Iowa, USA
Soon: 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

- Introduction
- Background on satisfiability procedures and rewrite-based deduction
- Synthetic benchmarks in the theory of arrays
- Experimental results with E and CVC
- Discussion

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.

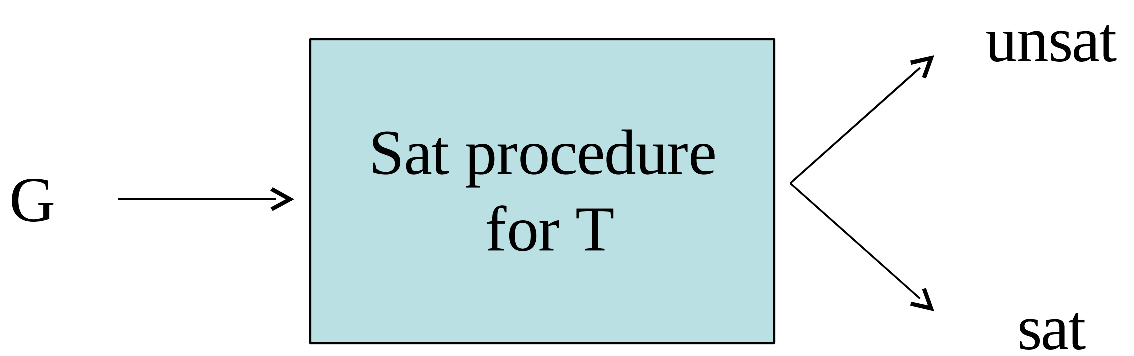
Satisfiability procedures

T : background theory, possibly with intended interpretation

φ : quantifier-free formula

φ' : DNF ($\neg\varphi$)

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, ..]
- Abstract frameworks [e.g., Tiwari] or proofs for concrete procedures [e.g., Shankar, Stump]
- Implement from scratch data structures and algorithms for each procedure: correctness of implementation? SW reuse?

Relation to term rewriting :

These theories involve equality:

- Ground completion and congruence closure to decide quantifier-free theory of equality
- Unification theory, reasoning “modulo” to work with a background theory
- Normalization: key notion in satisfiability procedures
- Completion-based, or, more generally, 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, Σ > : theorem-proving strategy

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

Σ: fair search plan

is a semi-decision procedure:

$T \cup G$

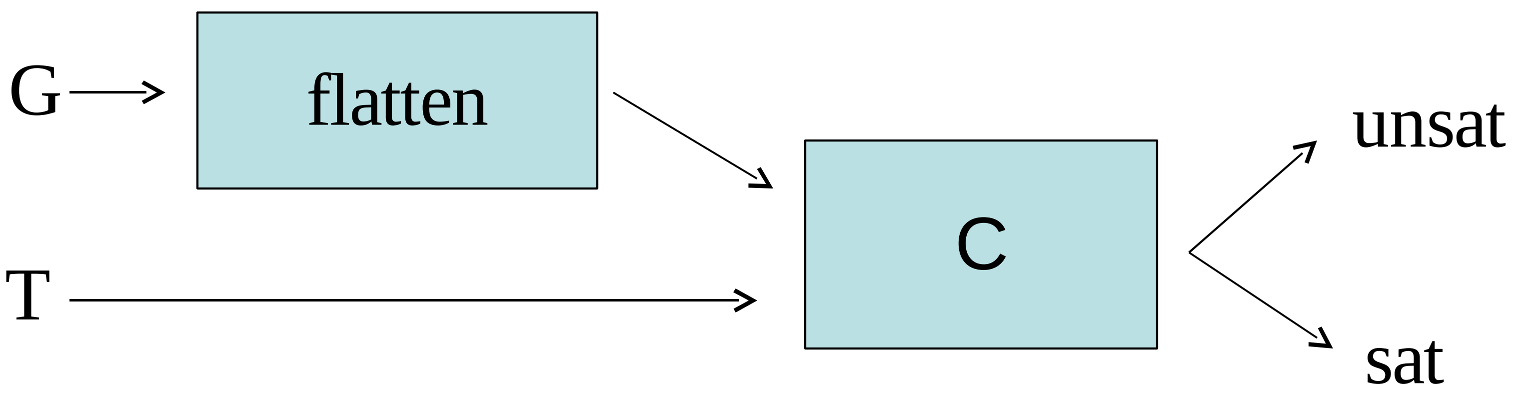
Yes, iff $T \cup G$ is unsatisfiable

?

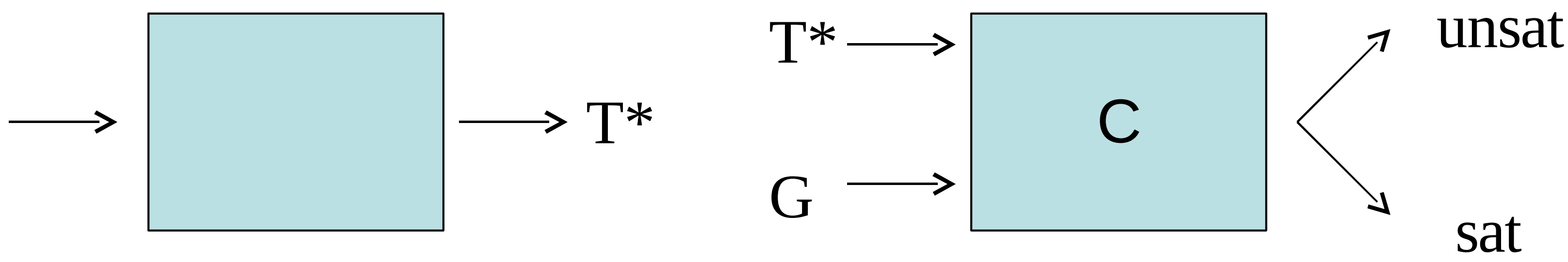
Termination results :

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]

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 !

May not be obvious:

- 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

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

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

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

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

$$t \neq t'$$

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

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

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

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

$$=$$

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

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:

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

Sample 10 permutations: 45 instances for each value of N

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

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

Number of instances: $1/2 \times N! \times (2^{(N/2)} - 1)$
Sample up to 16 permutations and 20 instances for each value of N.

Experiments

Set up of the experiments

- Two tools: CVC validity checker and E theorem prover
- E: auto mode and user-selected strategy
- Performance for N is average over all generated instances for value N
- 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 T1 and elimination of “store” via partial equations

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)

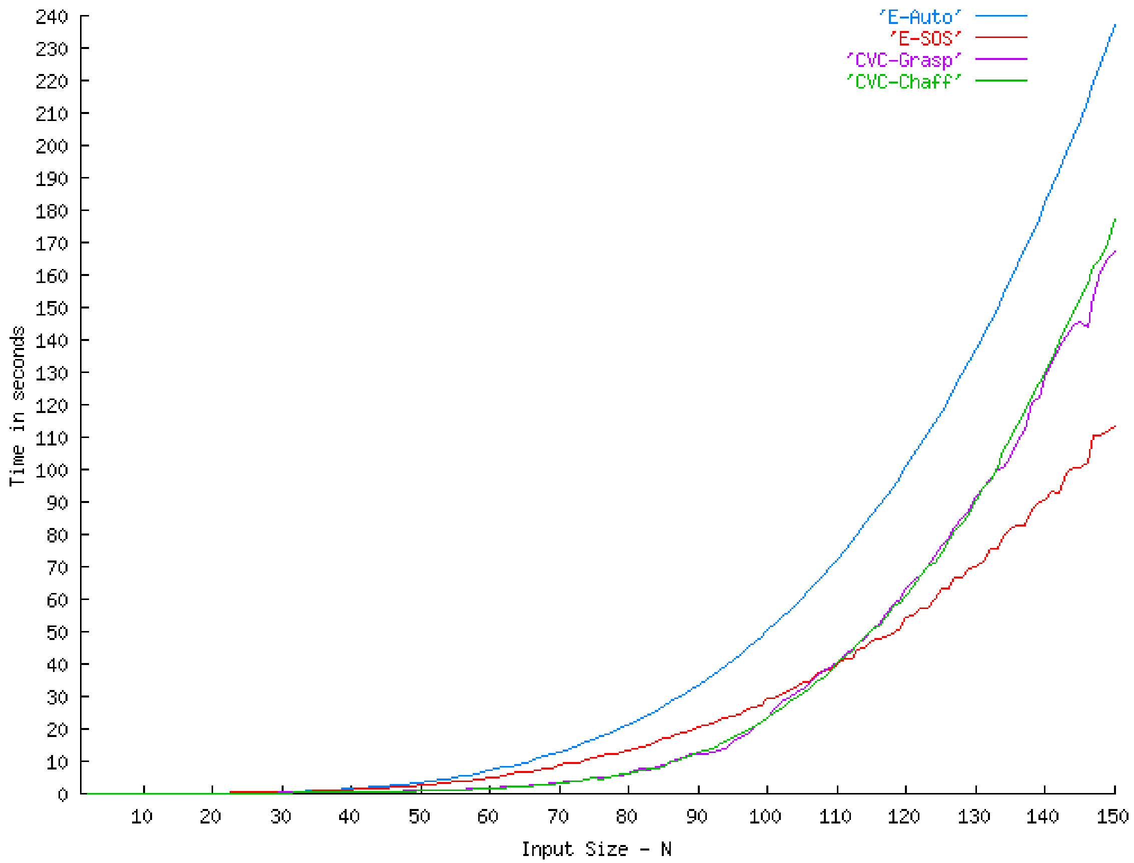
  Term ordering: LPO
- Precedence: select > store > sk > constants

Running CVC and E on storecomm(N)

N ranges from 2 to 150

E takes presentation T1 in input

Behavior on storecomm(N)



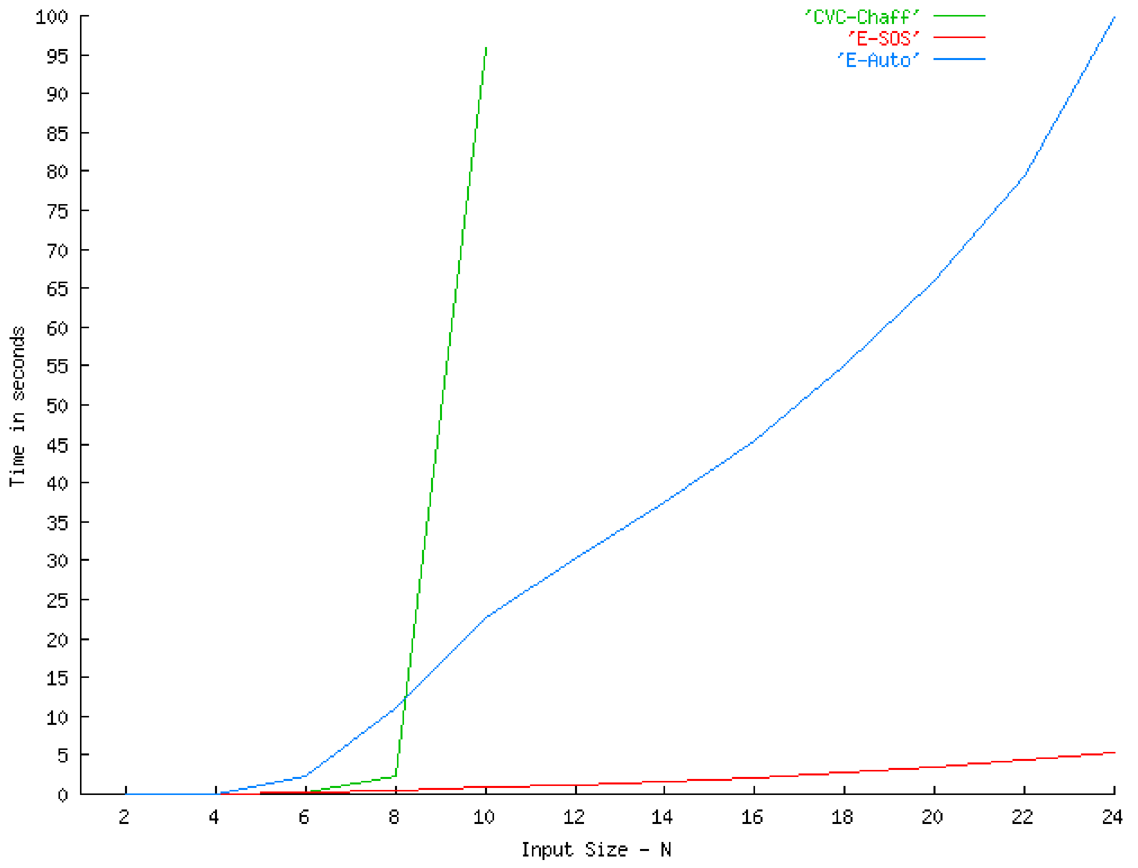
Running CVC and E on swap(N)

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 $N \geq 12$

Behavior on 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

- Theorem proving may help build better satisfiability procedures
- Theorem proving needs more work on auto mode and search plans (search, not blind saturation)
- Proof assistants incorporate satisfiability procedures: integration of automated theorem proving in proof assistants