Experiments and open issues on decision procedures theorem proving and software analysis

Maria Paola Bonacina
Dipartimento di Informatica
Universita` degli Studi di Verona

Outline

- First part: outside-in (work in progress)
  - From reasoning about SW to recent experiments with a FOL theorem prover in the theory of arrays
- Second part: inside-out (mostly ideas for the future)
  - Tailoring theorem proving and embedding it into software analysis tools

Outline of the first part

- Superposition-based satisfiability procedures for decidable theories
- A specific theory: arrays with extensionality
- A case study: three sets of synthetic benchmarks (parametric: empirical asymptotic behavior)
- Experiments comparing a superposition-based theorem prover and a validity checker

Outline of the second part

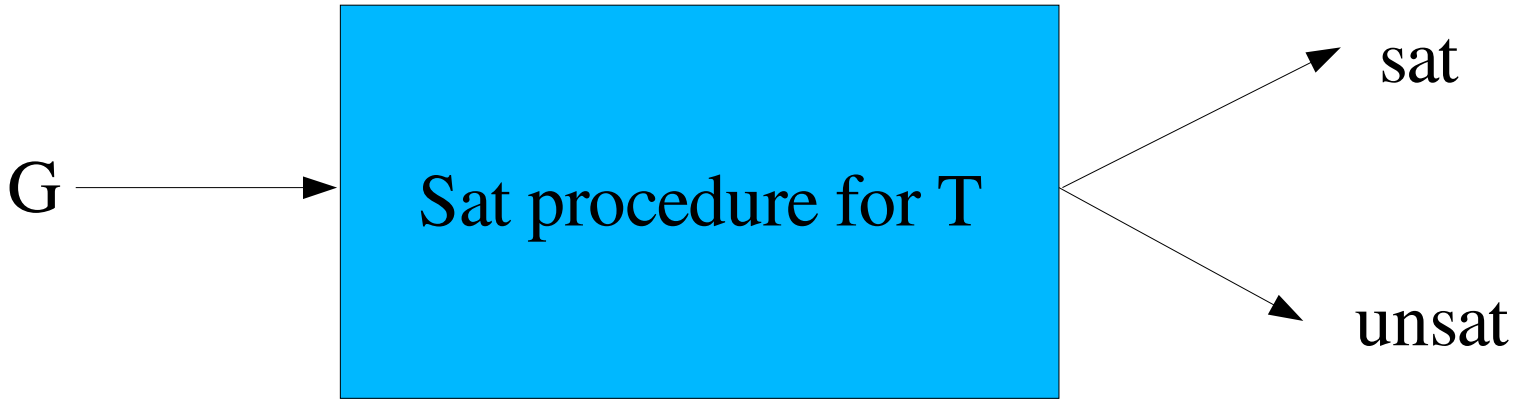
- From satisfiability procedures to decision procedures: current approaches
- From decision procedures to reasoning-based program analyzers
- Big picture: a few open issues in software analysis
- Discussion

Beginning the first part

- Reasoning about SW ... we all know why
- SW involves data types, e.g., integer, real, arrays, lists, sets, ...
- For some theories satisfiability is decidable (e.g., arrays)
- Satisfiability procedures

Satisfiability procedure

T : presentation of the background theory
(e.g., theory of arrays)
G : conjunction (set) of ground literals



G : set of arbitrary quantifier-free formulae (decision procedure)

Common approach

- Design
- Prove sound and complete
- Implement

  a satisfiability procedure for each decidable theory of interest.

Basic ingredients:

- Defined symbols (in T) and free symbols
- Congruence closure to handle equality and free symbols
- Build axioms of T into congruence closure algorithm

Examples

Theory of lists :

congruence closure with axioms built-in

[Nelson, Oppen JACM 1980]

Theory of arrays :

congruence closure with pre-processing with respect to axioms and partial equations (i.e., equalities that say that two arrays are equal except at certain indices)

[Stump, Barrett, Dill, Levitt LICS 2001]

Issues with the common approach

- Combination of theories / procedures
- Completeness proofs
- Implementation

First issue : combination

Most problems involve multiple theories:

combination of theories / procedures

Two congruence-closure based approaches:

[Nelson, Oppen ACM TOPLAS 1979] [Shostak JACM 1984]

that generated much scholarship:

[Cyrluk, Lincoln, Shankar CADE 1996]

[Harandi, Tinelli FroCoS 1998]

[Kapur RTA 2000]

[Ruess, Shankar LICS 2001]

[Barrett, Dill, Stump FroCoS 2002]

[Ganzinger CADE 2002]

Second issue : completeness proofs

Each new decision procedure needs its own proof of soundness and completeness:

Proofs for concrete procedures : complicated, ad hoc
[Shankar, Ruess LICS 2001]
[Stump, Barrett, Dill, Levitt LICS 2001]

Abstract frameworks : clarity, but gap wrt concrete procedures
[Bjorner PhD thesis 1998]
[Tiwari PhD thesis 2000]
[Bachmair, Tiwari, Vigneron JAR 2003]
[Ganzinger CADE 2002]

Third issue : implementation

Implement from scratch data structures and algorithms for each procedure in each context

(e.g., verification tool, proof assistant ...) :

- Correctness of implementation ?
- Flexibility ?
- SW reuse ?

Answer from a theorem-proving perspective

- Combination of theories : give union of the presentations in input to the prover
- Completeness proofs : use those given for known inference systems, no need of ad hoc proofs for each procedure
- Implementation : reuse code of existing provers

Termination ?

C = < I , P > : theorem-proving strategy

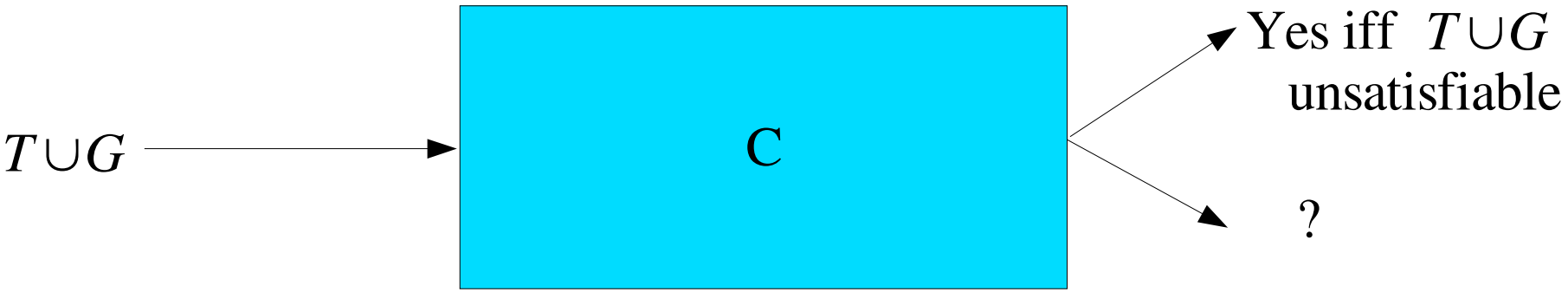
I : refutationally complete inference system with superposition/ paramodulation, (equational) factoring, simplification, subsumption ...

P : fair search plan

is a semi-decision procedure :

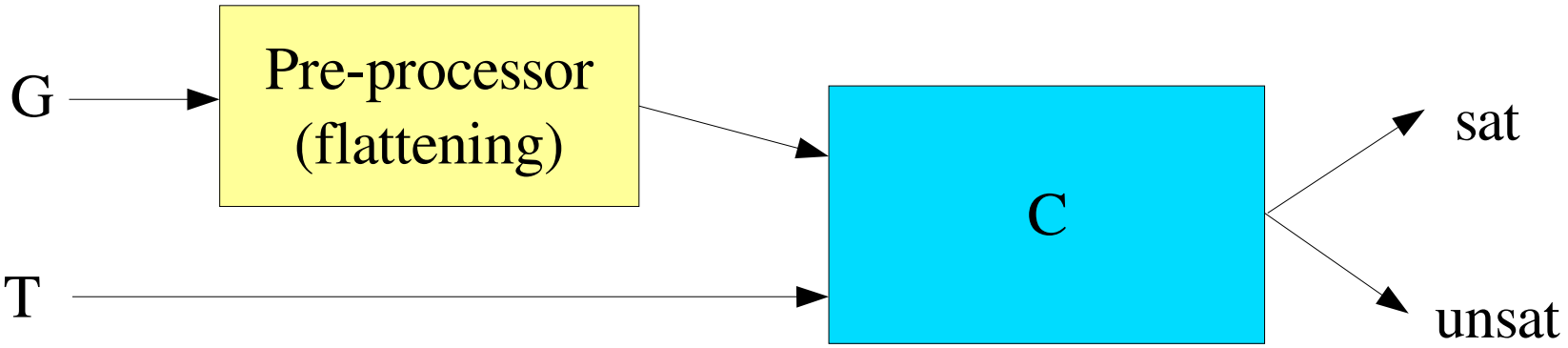
T : presentation of the theory (e.g., theory of arrays)

G : set of clauses (set of ground literals is a subcase)



Termination results

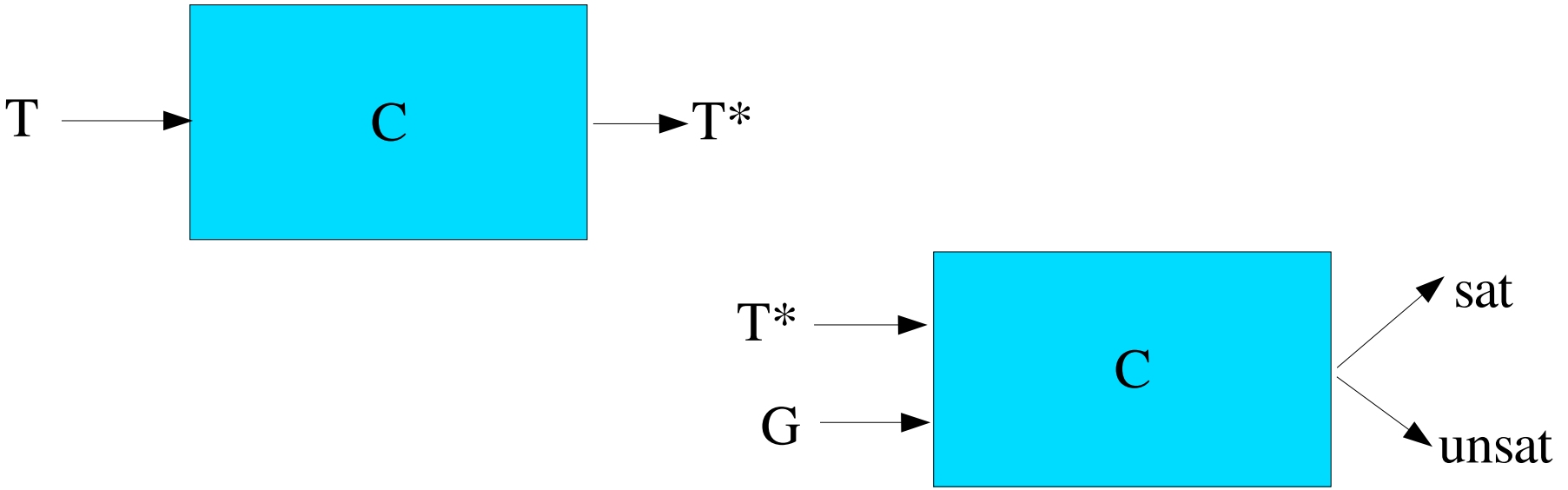
T : theory of arrays, lists, sets and combinations thereof
G : conjunction of ground literals
C = < I , P > : theorem-proving strategy



[Armando, Ranise, Rusinowitch CSL 2001]

Generalization : C can be a set of arbitrary quantifier-free formulae
[Ranise UNIF 2002]

Another way to put it



Pure equational : T* canonical rewrite system

Horn equational : T* saturated ground preserving

[Kounalis, Rusinowitch JSC 1991]

FOL special theories : e.g., T = T* for arrays

[Armando, Ranise, Rusinowitch IC 2003]

Theory of arrays : the signature

store : $ARRAY \times INDEX \times ELEMENT \longrightarrow ARRAY$

select : $ARRAY \times INDEX \longrightarrow ELEMENT$

The presentation (T_1)

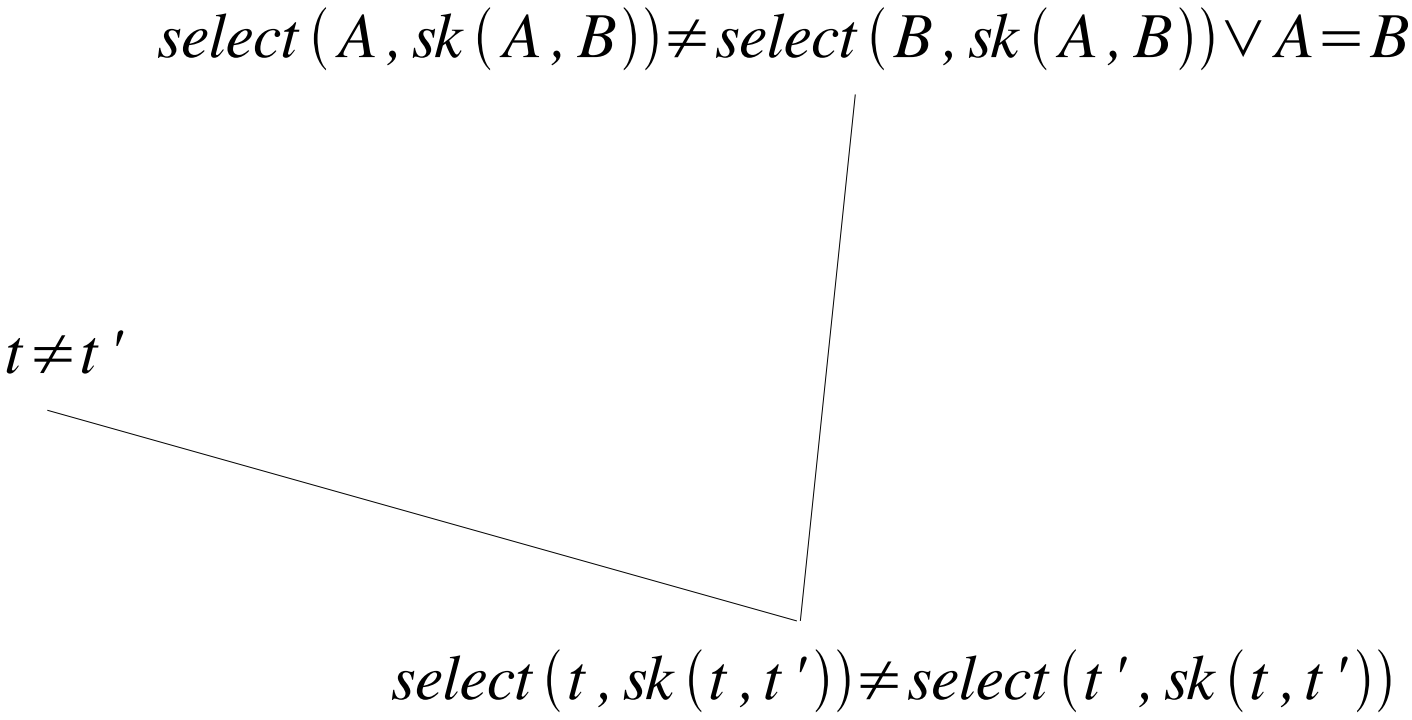
(1) $\forall A, I, E.\ select(store(A, I, E), I) = E$

(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



Proof of termination

Inference system : ordering-based
Expansion rules include superposition/paramodulation, reflection, equational factoring
Contraction rules include simplification and subsumption

Ordering : built out of precedence
store > select > a > e > i
for all constants a of sort ARRAY, e of sort ELEMENT and i of sort INDEX

Pre-processing: wrt extensionality + flattening

Proof : case analysis showing only finitely many clauses can be generated

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)

Usage of presentations

T_1 is saturated and application of C to T1 and G is guaranteed to terminate : C acts as a decision procedure

T_2 is not saturated (saturation does not halt) :
C applied to T_2 and G acts as semi-decision procedure

How about efficiency ?

A satisfiability procedure with T built into a congruence closure algorithm is expected to be always much faster than a superposition-based theorem prover with T in input!

Totally obvious ? Or worth investigating ?

- Synthetic benchmarks (allow one to assess scalability)
- Comparison : E prover and CVC validity checker (arrays built-in)

Three synthetic benchmarks

Storecomm(n) : Storing elements at distinct indices in an array is "commutative"

Swap(n) : Swapping the element at index i with the one at index j gives the same result as swapping the element at index j with the one at index i (generalized to n swap operations)

Storeinv(n) : If arrays A and B are equal after swapping elements of A with corresponding elements of B, A and B must have been equal to begin with.

Storecomm(n) : intuition

The instance for n = 2 :

$$i_1 \neq i_2 \Rightarrow$$
$$store(store(a, i_1, e_1), i_2, e_2) = store(store(a, i_2, e_2), i_1, e_1)$$

The relative order of store operations is immaterial.

Storecomm(n,p,q) : definition

n > 0

p, q : permutations of { 1, ... n }

D : set of 2-combinations over { 1, ... n }

Storecomm(n,p,q) is the formula

$$\bigwedge_{(l,m)\in D} i_l \neq i_m \Rightarrow (T_n(p) = T_n(q))$$

where

$$T_k(p) = a \quad if \quad k = 0$$

$$T_k(p) = store(T_{k-1}(p), i_{p(k)}, e_{p(k)}) \quad if \quad 1 \leq k \leq n$$

Storecomm(n) : definition

Let q be the identity permutation ι

Storecomm(n,p) = Storecomm(n, p, ι)

Storecomm(n) = { Storecomm(n,p) :
p is a permutation of {1, ... n} }

Storecomm(n) is a set of n! problems.

Two very recent results

Using the case analysis of the proof of termination we proved that for Storecomm(n)

- Equational Factoring and
- Paramodulation into negative unit clauses

can be disabled without losing refutational completeness.

Swap(n) : intuition

The instance for n = 2 :

$$swap(swap(a, i_0, i_1), i_2, i_1) = swap(swap(a, i_1, i_0), i_1, i_2)$$

where

$swap(a, i, j)$

stands for

$store(store(a, i, select(a, j)), j, select(a, i))$

Swap(n, c_1, c_2, p, q) : definition

c_1, c_2 : subsets of {1, ... n}

p, q : functions p, q : {1, ... n} $\longrightarrow$ {1, ... n}

Swap(n, c_1, c_2, p, q) is the equation

$$T_n(c_1, p, q) = T_n(c_2, p, q)$$

where

$$T_k(c, p, q) = a \quad if \quad k=0$$

$$T_k(c, p, q) = swap(T_{k-1}(c, p, q), i_{p(k)}, i_{q(k)}) \quad if \quad 1 \le k \le n \ \wedge \ k \in c$$

$$T_k(c, p, q) = swap(T_{k-1}(c, p, q), i_{q(k)}, i_{p(k)}) \quad if \quad 1 \le k \le n \ \wedge \ k \notin c$$

Swap(n) : definition

Swap(n) = { Swap(n, c_1, c_2, p, q) :

c_1, c_2 subsets of {1, ... n}

p, q functions from {1, ... n} to {1, ...n} }

Thus Swap(n) is a set of $2^{2n}n^{2n}$ problems.

Storeinv(n) : intuition

Case where a single index is involved :

$$store(a, i, select(b, i)) = store(b, i, select(a, i)) \Rightarrow a = b$$

Storeinv(n) : definition

$n \geq 0$

$$Storeinv(n) \quad = \quad \left\{ \; multiswap(a, b, n) \; \Rightarrow \; a = b \; \right\}$$

where

$$multiswap(a, b, k) \; = \; (a = b) \quad if \quad k = 0$$

$$multiswap(a, b, k) \quad =$$
$$store(a', i_k, select(b', i_k)) \; = \; store(b', i_k, select(a', i_k)) \quad if \quad k \geq 1$$

with $\quad (a' = b') \quad = \quad multiswap(a, b, k-1)$

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. at Stanford University]
[Aaron Stump at the Washington University in St. Louis]

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

Incorporates SAT solver for case analysis (first GRASP then Chaff)

Theory of arrays : congruence closure based algorithm with pre-processing with respect to axioms and partial equations (i.e., equalities that say that two arrays are equal except at certain indices)
[Stump, Barrett, Dill, Levitt LICS 2001]

Why CVC ?

We compare with CVC because it is the only system we are aware of that implements a complete decision procedure for the theory of arrays with extensionality:

neither ICS [Harald Ruess, personal communication, April 2003]
nor Simplify [Detlefs, Nelson, Saxe, TR HP Labs, 2003]
are complete for this theory.

The E theorem prover

[Stephan Schulz, TU-München, RISC Linz, IRST Trento]

Inference system I : ordering-based
Expansion rules include superposition/paramodulation, reflection, equational factoring
Contraction rules include simplification and subsumption

Search plans P : given-clause loop

- Only already-selected list kept inter-reduced
- Clause selection functions
- Term orderings : KBO, LPO
- Literal selection functions

Performance on Storecomm(n)

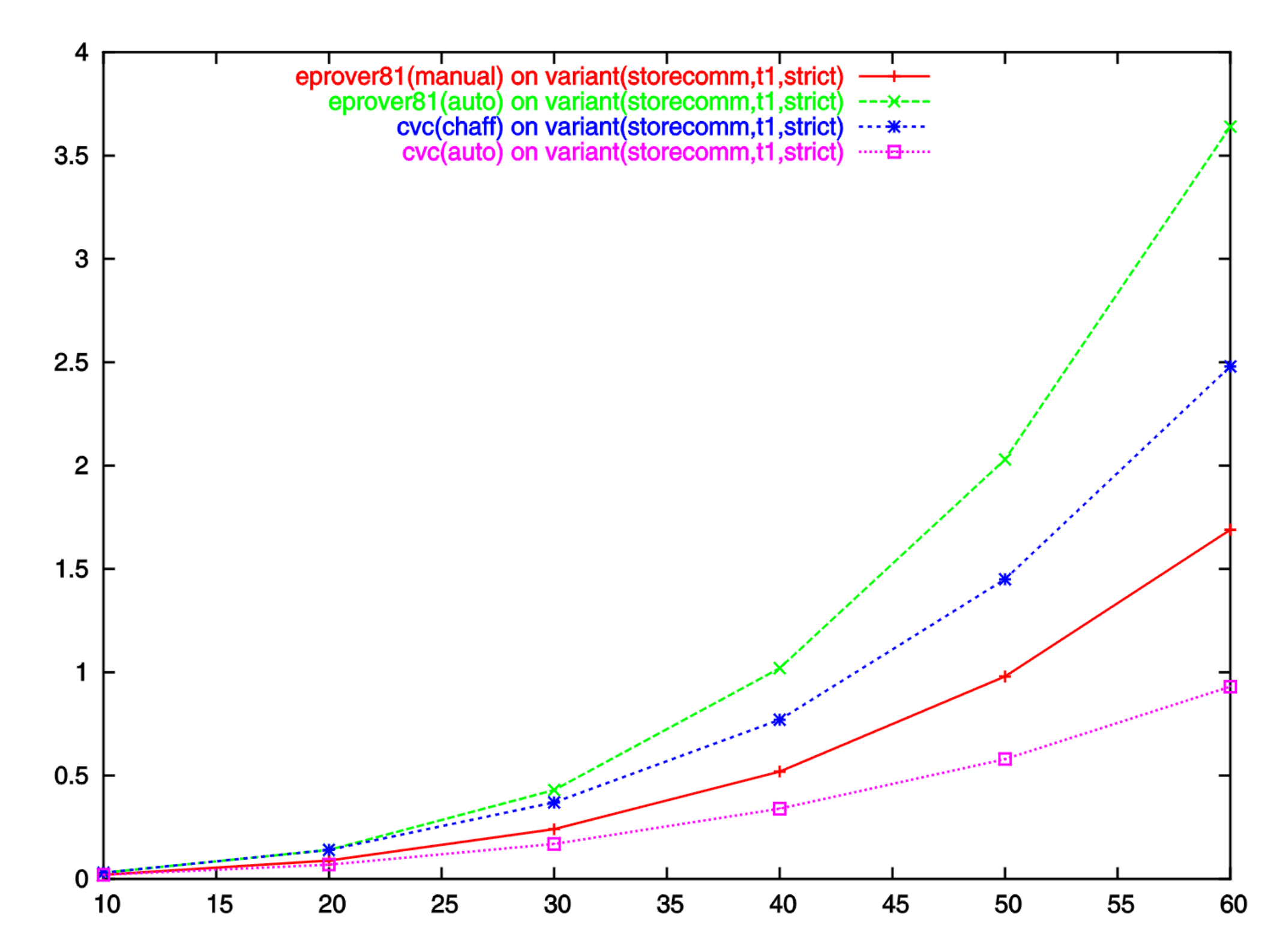
E-auto : automatic mode

E-manual : user-selected strategy with
Clause selection : (PreferGround, RefinedWeight)
Term ordering : KBO (all benchmarks, also in auto mode)
Precedence : store > select > constants

E takes presentation T_1 in input

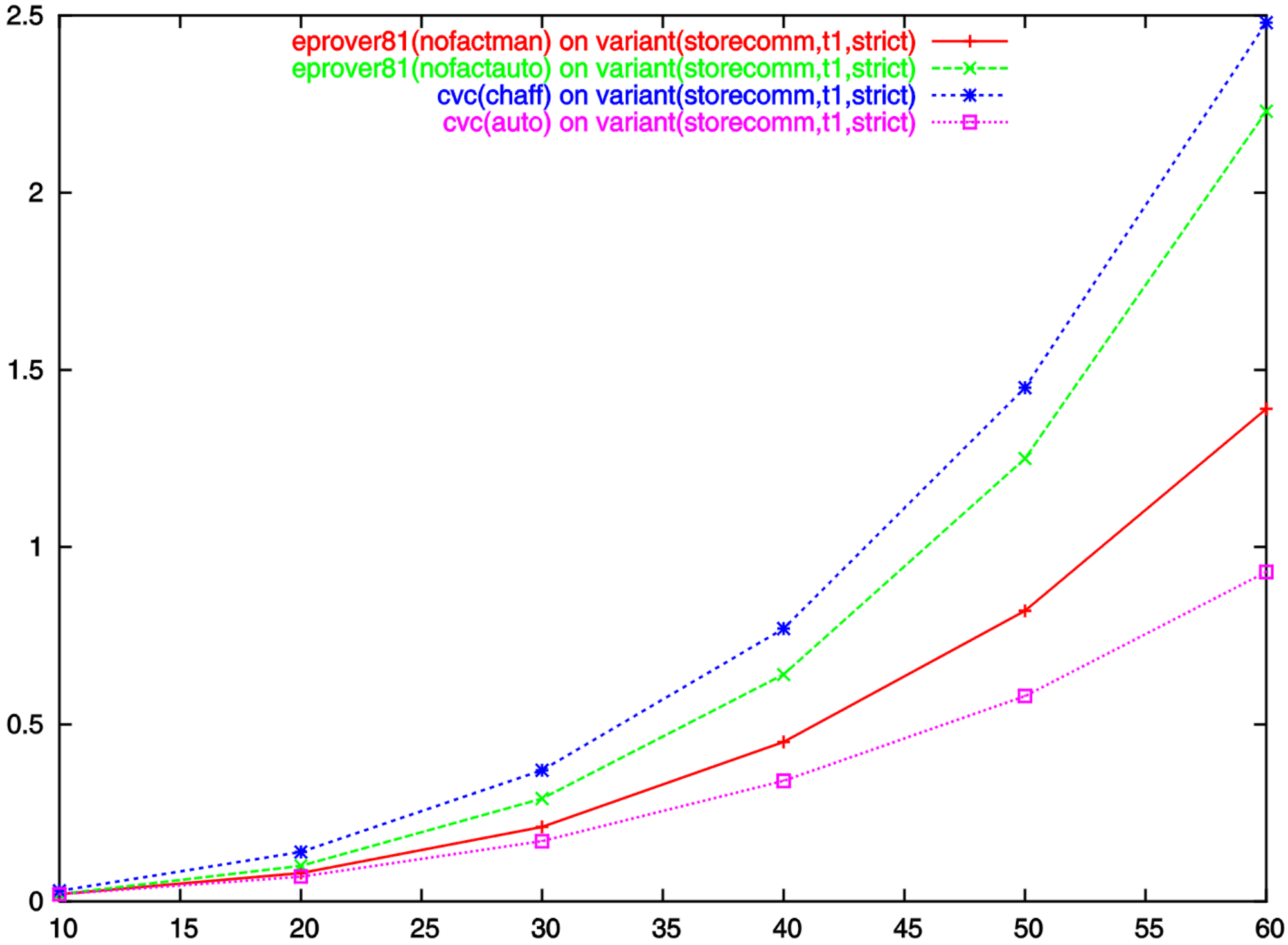
n ranges from 10 to 60

Performance (in sec) is the median over 5 random samples for each value of n


eprover81(manual) on variant(storecomm,t1,strict)
eprover81(auto) on variant(storecomm,t1,strict)
cvc(chaff) on variant(storecomm,t1,strict)
cvc(auto) on variant(storecomm,t1,strict)
4
3.5
3
2.5
2
1.5
1
0.5
0
10
15
20
25
30
35
40
45
50
55
60

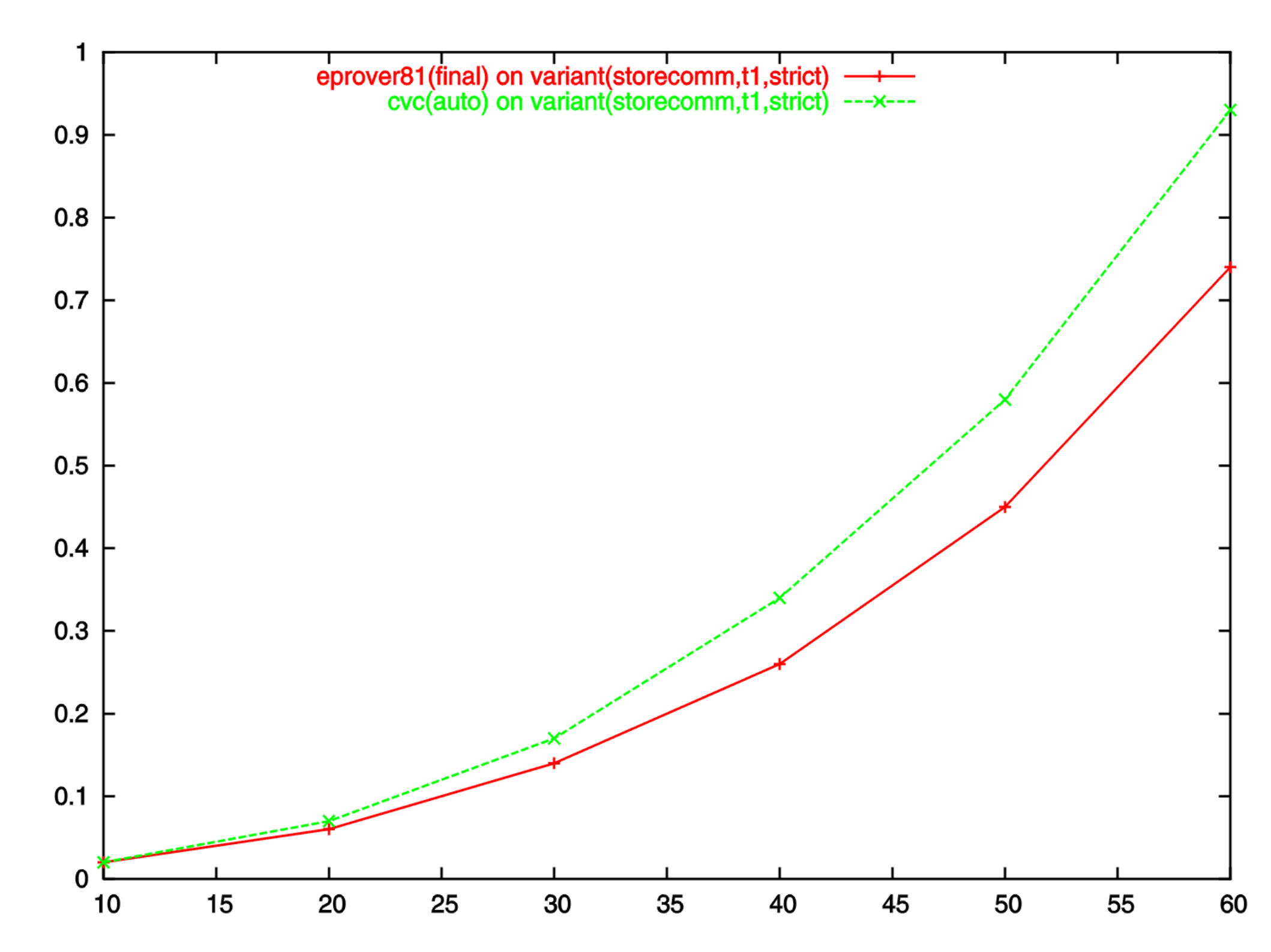
Tuning the prover I

The next slide shows the effect of disabling equational factoring.


eprover81(nofactman) on variant(storecomm,t1,strict)
eprover81(nofactauto) on variant(storecomm,t1,strict)
cvc(chaff) on variant(storecomm,t1,strict)
cvc(auto) on variant(storecomm,t1,strict)
2.5
2
1.5
1
0.5
0
10
15
20
25
30
35
40
45
50
55
60

Tuning the prover II

The next slide shows the effect of disabling also paramodulation into negative unit clauses and contraction of the given clause upon its selection (never used).


eprover81(final) on variant(storecomm,t1,strict)
cvc(auto) on variant(storecomm,t1,strict)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10
15
20
25
30
35
40
45
50
55
60

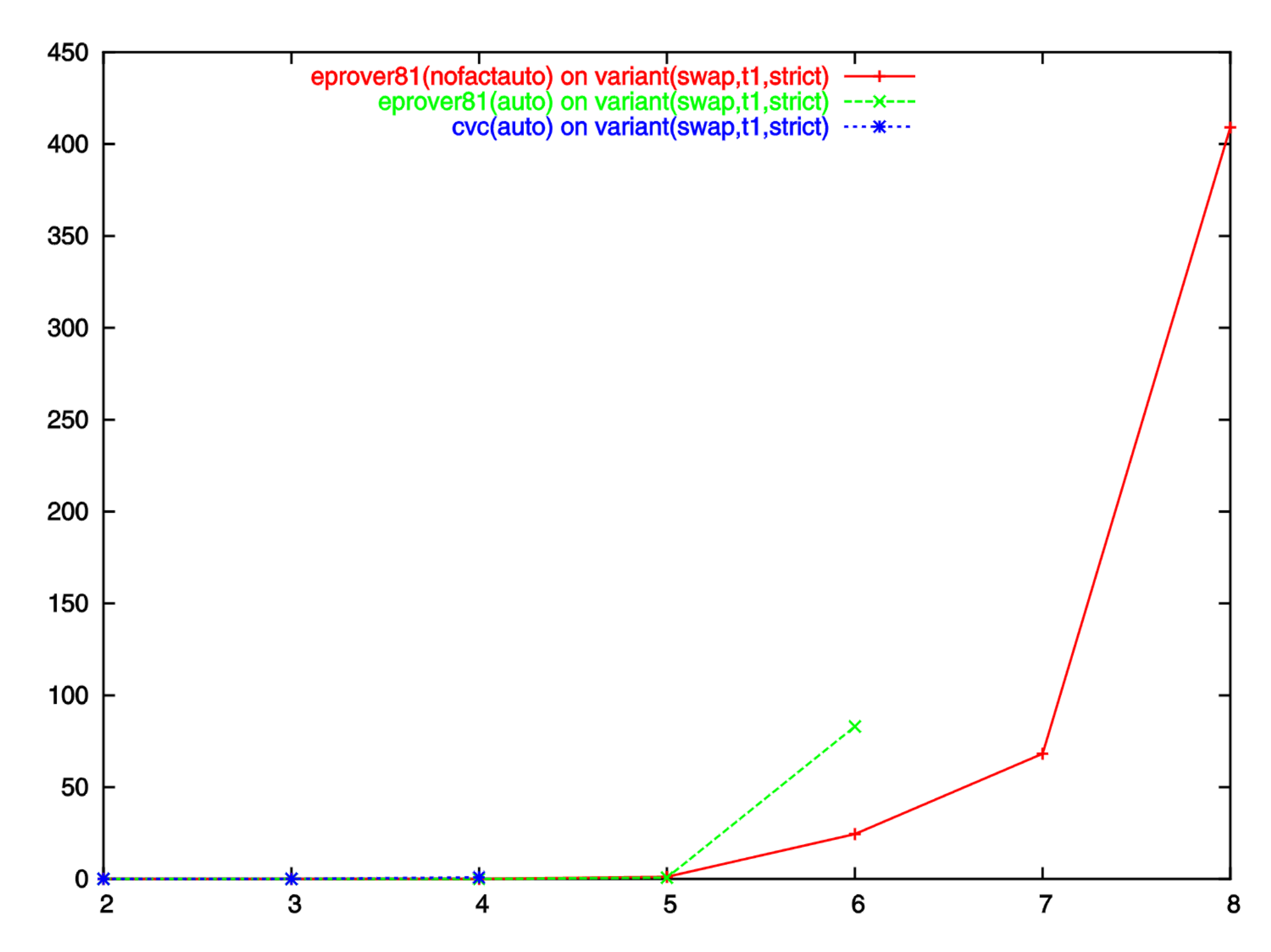
Performance on Swap(n)

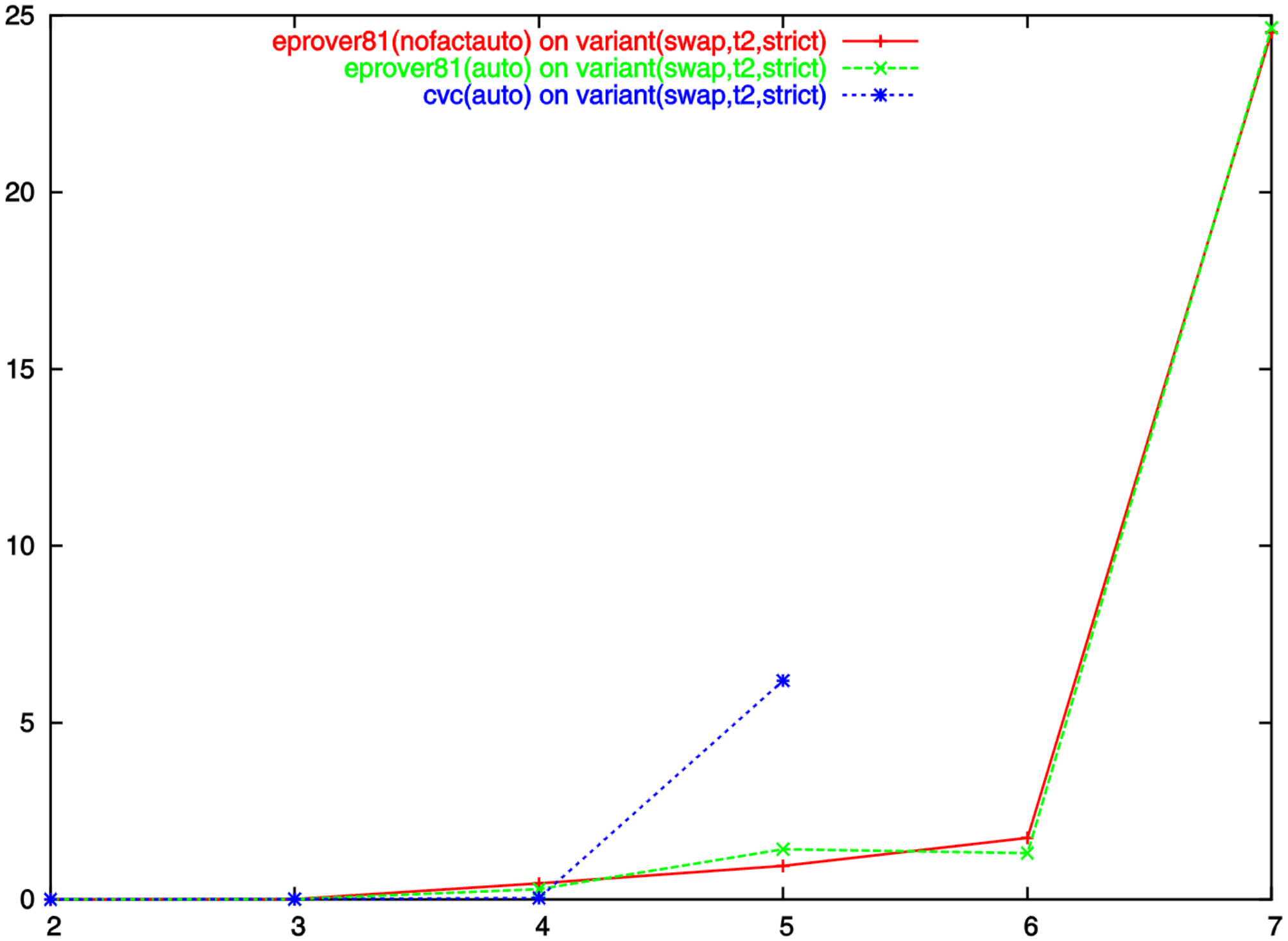
E-auto is sufficient

The reported performance (in sec) is the median over 5 random samples for each value of n

Next two slides :

- Performance with presentation T_1
- Performance with presentation T_2


eprover81(nofactauto) on variant(swap,t1,strict)
eprover81(auto) on variant(swap,t1,strict)
cvc(auto) on variant(swap,t1,strict)
450
400
350
300
250
200
150
100
50
0
2
3
4
5
6
7
8


eprover81(nofactauto) on variant(swap,t2,strict)
eprover81(auto) on variant(swap,t2,strict)
cvc(auto) on variant(swap,t2,strict)
25
20
15
10
5
0
2
3
4
5
6
7

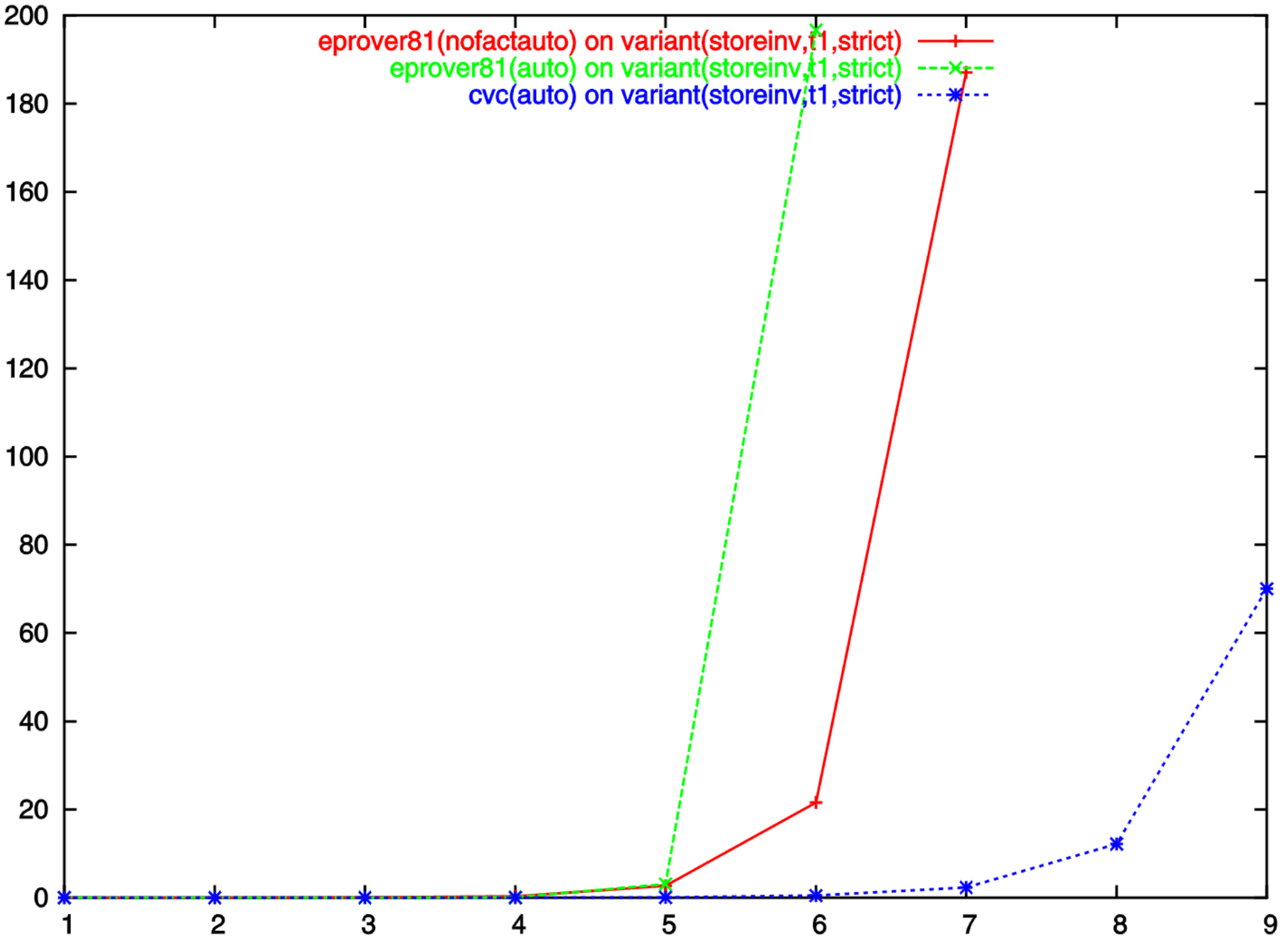
Performance on Storeinv(n)

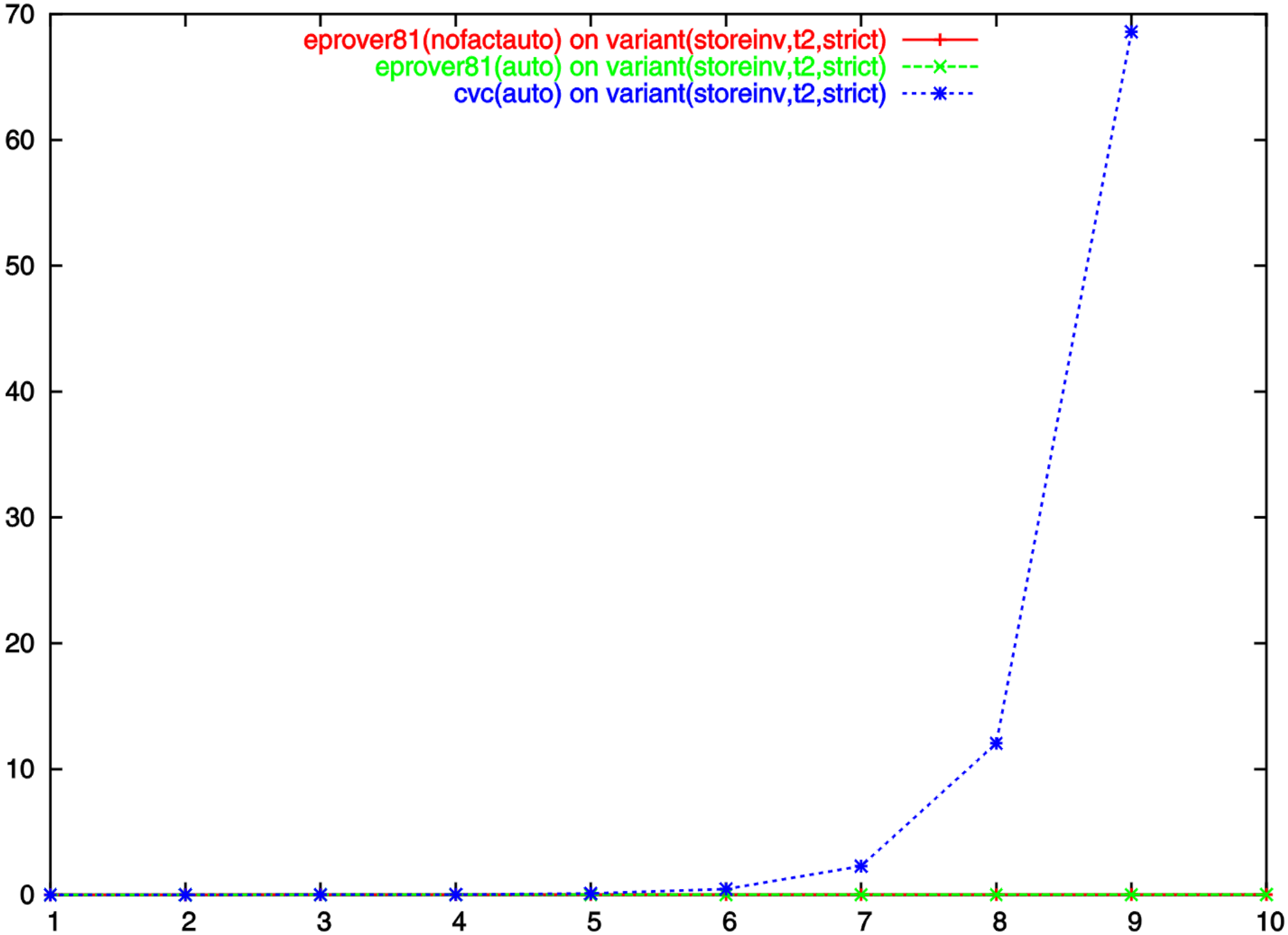
E-auto is sufficient.

Performance (in sec) is absolute, because Storeinv(n) contains only one problem: no sampling.

Next two slides :

- Performance with presentation T_1
- Performance with presentation T_2


eprover81(nofactauto) on variant(storeinv,t1,strict)
eprover81(auto) on variant(storeinv,t1,strict)
cvc(auto) on variant(storeinv,t1,strict)
200
180
160
140
120
100
80
60
40
20
0
1
2
3
4
5
6
7
8
9


eprover81(nofactauto) on variant(storeinv,t2,strict)
eprover81(auto) on variant(storeinv,t2,strict)
cvc(auto) on variant(storeinv,t2,strict)
70
60
50
40
30
20
10
0
1
2
3
4
5
6
7
8
9
10

Discussion of the experiments

- Against expectations, the general-purpose theorem prover is competitive with the specialized decision procedure.
- Nevertheless, we do not advocate using the theorem prover (too unwieldy) but carving better decision procedures out of the inference rules, search plans (and code!) of theorem provers (e.g., disabling equational factoring).

Continuing this work

- Try satisfiable inputs
- Try non-synthetic problems
- Automate the decision of disabling equational factoring
- Understand why Storeinv(n) is so easy for T_2
- Beyond arrays : other theories, combinations of theories

Related work

Proof of correctness of a basic Unix-style file system implementation

Proof checker (Athena) which integrates two paramodulation-based provers similar to E :
Vampire [Voronkov, Riazanov, U. Manchester] and
SPASS [Weidenbach et al., MPI Saarbrücken]
used for non-inductive reasoning about lists, arrays, etc., on the basis of their first-order axiomatizations

Full correctness proof (simulation relation between specification and implementation) needs (some) general-purpose deduction.

[Konstantine Arkoudas, Karen Zee, Viktor Kuncak and Martin Rinard MIT CSAIL TR 946, 2004]

From satisfiability procedures to decision procedures

Turn arbitrary quantifier-free formula F into DNF and use satisfiability procedure : not effective.

Use superposition-based inference system (termination proof extends from ground literals to ground clauses for arrays etc.) : not tested.

Integrate satisfiability procedure(s) with SAT solver to exploit its unmatched strength on the boolean structure of the formula.

Integration with SAT solver

Abstraction + iteration, e.g.:

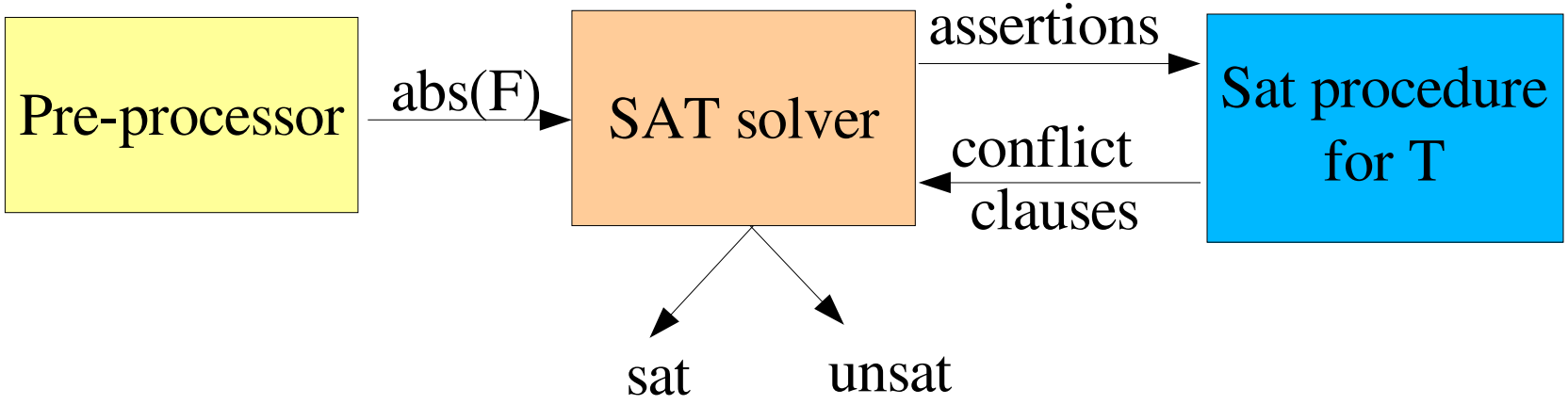
[Armando et al. ECP 1999 : TSAT] (temporal reasoning)

[Audemard et al. CADE 2002 : Math SAT] (mathematics)

[Barrett et al. CAV 2002 : CVC] (no quantifiers)

[de Moura et al. CADE 2002 : ICS] (no quantifiers)

[Deharbe, Ranise SEFM 2003 : haRVey] (with quantifiers) (*)



Plug-in a superposition-based procedure for the theory (*)

From decision procedures to program analysis

What is program analysis ?

Approaches to software quality:

- Process-based (historically dominant)
- Evidence-based (current trend, especially for safety)

  Evidence-based methodologies:

  - Testing (historically dominant)
  - Program analysis

Program analysis : all techniques (mostly semi-automated) to determine whether a program satisfies given properties (e.g., absence of certain bugs).

Program analysis

Although program analyzers do exists (e.g., the products by AbsInt or PolySpace), program analysis is very difficult in general.

Typical issues:

- Program class (e.g., no complex structures, no threads)
- Language class (e.g., no OOP)
- Too many false positives (say there's a bug and there is not)

Technologies for program analysis

- Annotations with pre- and post-conditions
- Modelling languages (e.g., UML, JML, Alloy)
- Static analysis: controlflow analysis, dataflow analysis, shape analysis
- Integration of CASE tools with interactive theorem provers (e.g., Coq, Isabelle, PVS) or automated but heuristic provers (e.g., Simplify)

Complementarities

For example, take again file systems:

Alloy (specification language with its model finder) has been used to check structural properties of file systems for debugging, but is not meant to show full functional correctness as in the more theorem-proving oriented approach of Athena with Spass or Vampire.

Common issue: more automation

Contrast with hardware analysis by model checking.

Fundamental difference :

- Modelling hardware circuits : finite state systems
- Modelling software systems : requires infinite domains

Software model checking : model checking + theorem proving
as in the abstract-check-refine paradigm

Abstract-check-refine paradigm

Build abstraction B of program P (e.g., boolean program, linear program)
Check B (model checking) :
  if success (i.e., no bug), exit (P also bug free)
  if failure, see if error trace in B is also in P :
      if yes, bug found in P
       else Refine B (theorem proving) and repeat.

[Ball, Rajamani SPIN 2000 Bepop]
[Ball, Rajamani SPIN 2001 SLAM] (linear programs)
[Henzinger et al. POPL 2002 BLAST] (non-recursive C programs)
[Armando et al. TR DIST UniGE 2004 eureka] (linear programs with external ground decision procedure for linear arithmetic + ICS)

Open issues

Theorem proving used in current approaches to SW model checking is

- either generic (no specialized decision procedures)
- or incomplete (false positives), even unsound (false negatives)
- or not fully automated.

Other issues:

- Expressivity (check what you intend)
- Flexibility (sufficient theory support)
- Feed-back (e.g., counter-models for non-valid properties)

Discussion

Fully automated program analyzers capable of handling programs with

- Rich data structures
- General loops
- Tight interplay between data and control

call for

- Integration of existing technologies/systems (CASE, ATP, SAT, AMB ...)
- Combination of expertises (modelling, reasoning ...)

Joint work with

Alessandro Armando
(DIST, Universita` degli Studi di Genova)

Stefano Ferrari
(my student at the Universita` degli Studi di Verona)

Silvio Ranise
(INRIA Lorraine, Nancy)

Supported in part by MIUR PRIN project no. 2003-097383