

An Experimental Evaluation of Ground Decision Procedures

Leonardo de Moura

(joint work with Harald Rueß)

`demoura@csl.sri.com`

Computer Science Laboratory

SRI International

Menlo Park, CA

Overview

- ▶ Motivation
- ▶ Ground Decision Procedures
- ▶ Systems
- ▶ Benchmark Suites
- ▶ Setup
- ▶ Experimental Results
- ▶ Observations & Research Questions

Motivation

- ▶ Improve ICS
- ▶ Understand the weaknesses and strengths of each approach
- ▶ SAT community approach
 - Optimize based on real examples from different domains.

Ground Decision Procedures

- ▶ **Decision procedures** are an enabling technology for:
 - ▶ Automated deduction in hardware and software verification
 - ▶ Planning
 - ▶ Bounded Model Checking
 - ▶ Search Problems
- ▶ **Example:**

$$z = f(x - y) \wedge x = z + y \rightarrow -y = -(x - f(f(z))) .$$

- ▶ **Goal:** propositional formulas with thousands of variables and hundreds of thousands of literals.

Propositional combinations of constraints

- ▶ **Given**: a procedure for deciding satisfiability of conjunctions of constraints in \mathcal{T} .
- ▶ **Problem**: decide propositional combinations of constraints in \mathcal{T} .
- ▶ **Naive Solution**: transform the formula into **DNF** (too expensive).

Propositional combinations of constraints

- ▶ Extensions of Binary Decision Diagrams (**BDDs**).
- ▶ Eager Approach
 - ▶ Equisatisfiable translation to propositional logic.
 - ▶ **Two Steps**: Translator + (most efficient) SAT solver
- ▶ Lazy Approach
 - ▶ SAT solver (search) + Constraint Solver
 - ▶ Replace constraints by fresh propositional variables.
 - ▶ SAT solver checks if the formula is satisfiable.
 - ▶ Constraint solver validates the (partial) assignment.
 - ▶ Assignment is not valid \Rightarrow **lemma** is generated.

Systems

- ▶ Stanford Validity Checker (SVC)
- ▶ The Cooperating Validity Checker (CVC)
 - ▶ CVC 1.0a
 - ▶ CVC Lite 1.0.1
- ▶ The Integrated Canonizer and Solver (ICS 2.0)
- ▶ UCLID
- ▶ Math-SAT
- ▶ Simplify

Systems (cont.)

	UF	Arith.	Int.	Approach
SVC	×	LA		Shostak - FM - BDD
CVC	×	LA		Shostak - FM - Lazy
ICS	×	LA		Shostak - Simplex - Lazy
UCLID	×	SL	×	Eager
Math-SAT		LA		BF & Simplex - Lazy
Simplify	×	LA		NO - Simplex

UF: uninterpreted functions

LA: linear arithmetic

SL: separation logic ($x - y \leq c$)

BF: Bellman-Ford

NO: Nelson-Oppen

Benchmark suites

- ▶ **The Math-SAT benchmark suite**: timed automata verification problems.
- ▶ **The UCLID benchmark suite**: processor and cache coherence protocol verification.
- ▶ **The SAL benchmark suite**: timed automata and linear hybrid systems, and test-case generation for embedded controllers.
- ▶ **The SEP benchmark suite**: symbolic simulation of hardware designs, timed automata systems, and scheduling problems.

Benchmark suites (cont.)

Benchmark suite	Sat	Unsat	Unsolved
Math-Sat	37	224	19
UCLID	0	36	2
SAL	21	167	29
SEP	9	8	0

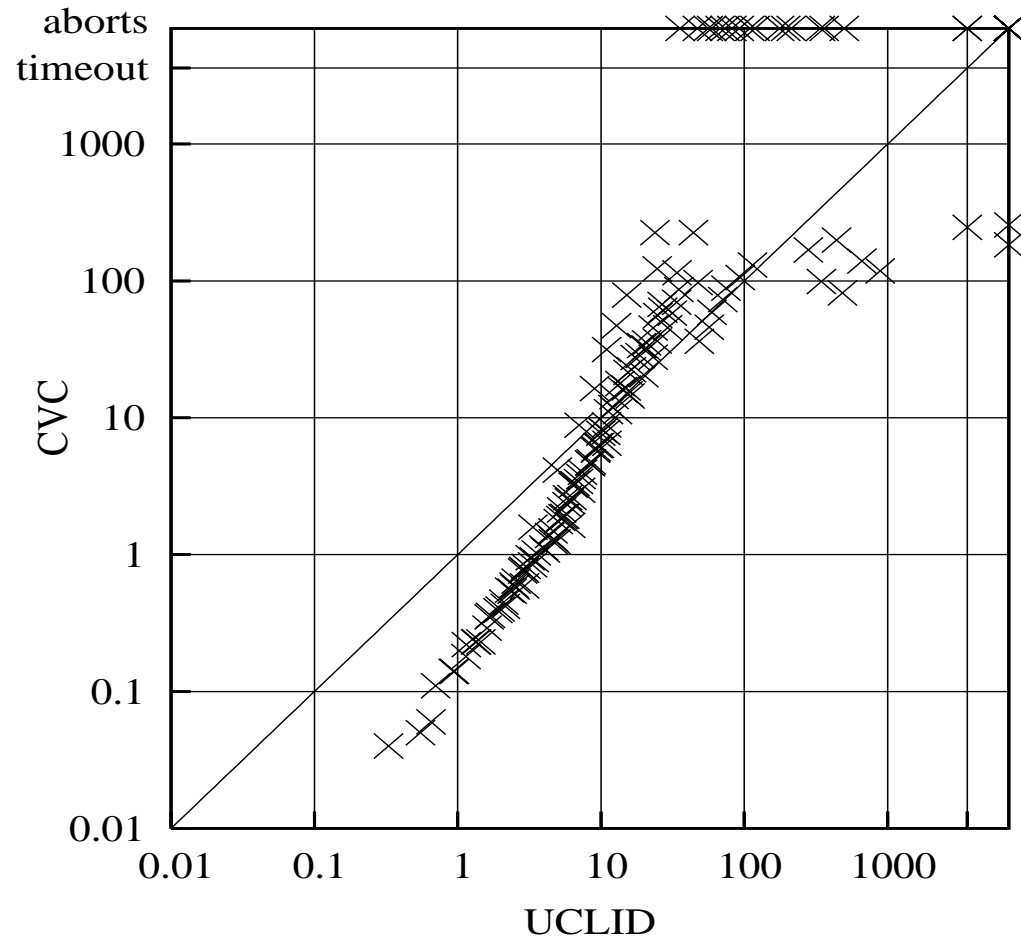
Setup

- ▶ Configuration: 1GHz Pentium III processor 256Kb of cache and 512Mb of RAM, running Red Hat Linux 7.3
- ▶ Timeout: 3600 secs
- ▶ Data Area: 450 Mb
- ▶ Stack Area: 40 Mb
- ▶ It required more than **30 CPU days** to run all experiments.

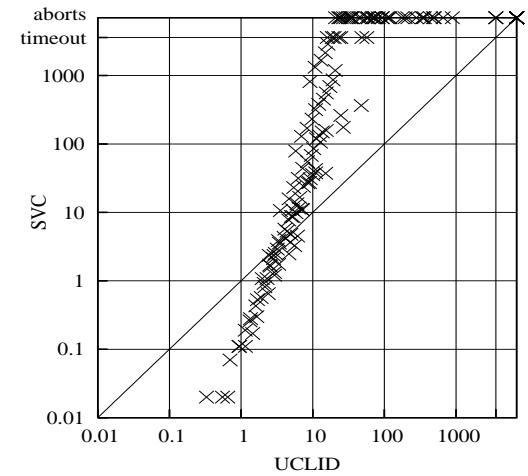
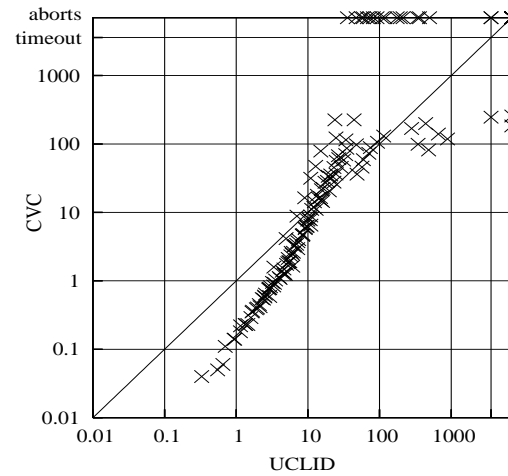
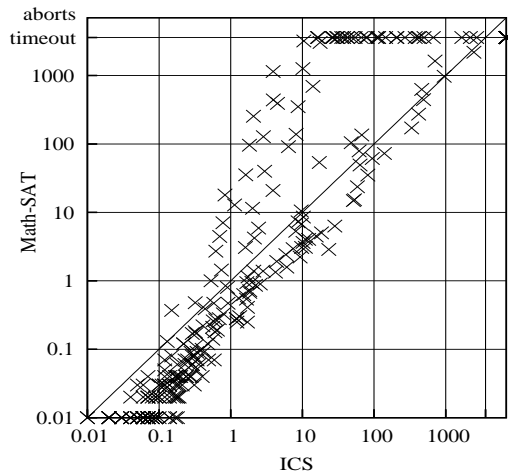
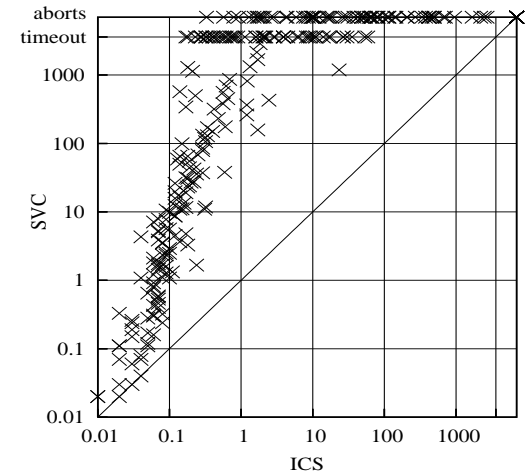
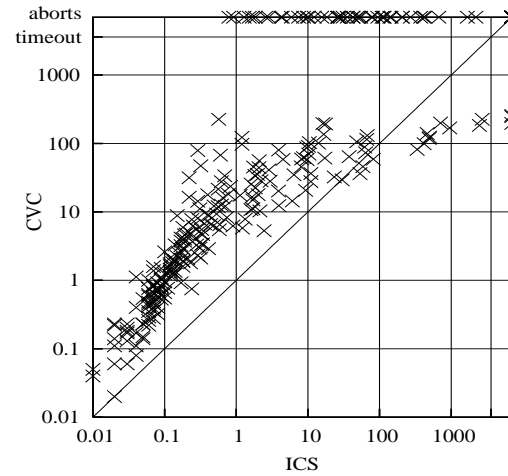
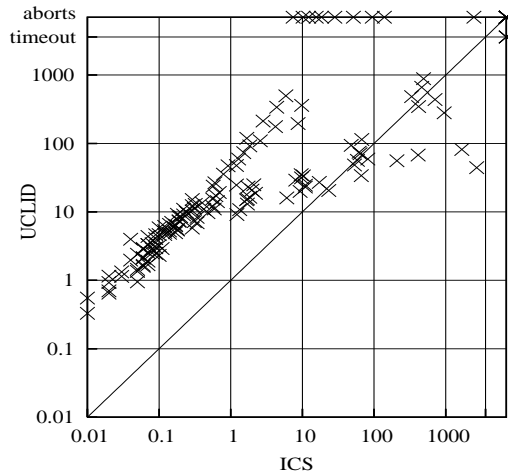
Setup (cont.)

ICS	<code>ics <i>problem-name</i>.ics</code>
UCLID	<code>uclid <i>problem-name</i>.ucl sat 0 zchaff</code>
CVC	<code>cvc +sat < <i>problem-name</i>.cvc</code>
CVC Lite	<code>cvcl +sat fast < <i>problem-name</i>.cvc</code>
SVC	<code>svc <i>problem-name</i>.svc</code>
Simplify	<code>Simplify <i>problem-name</i>.smp</code>
Math-SAT	<code>mathsat_linux <i>problem-name</i>.ms -bj math -heuristic SatzHeur</code>

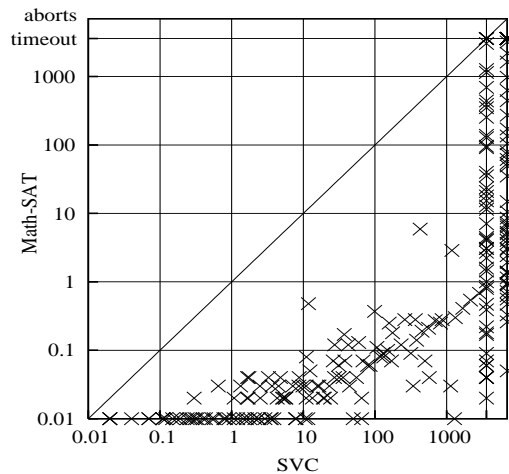
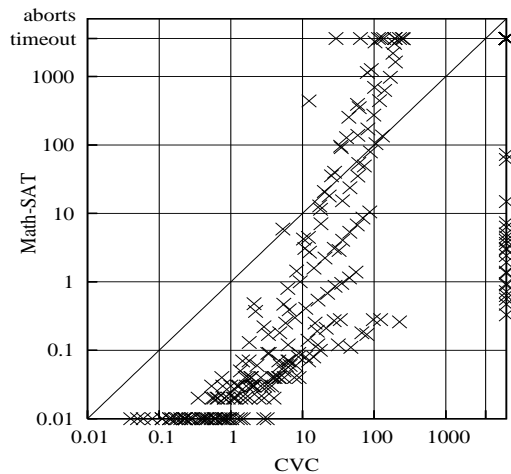
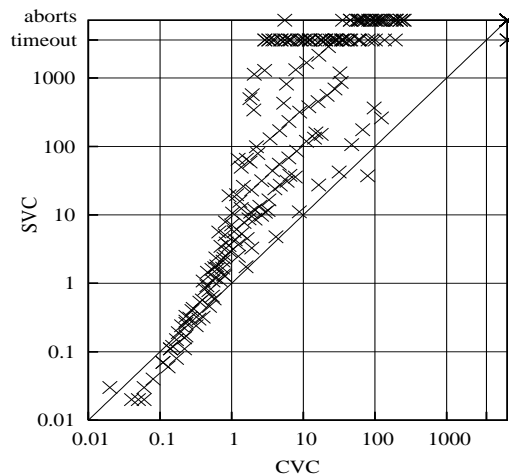
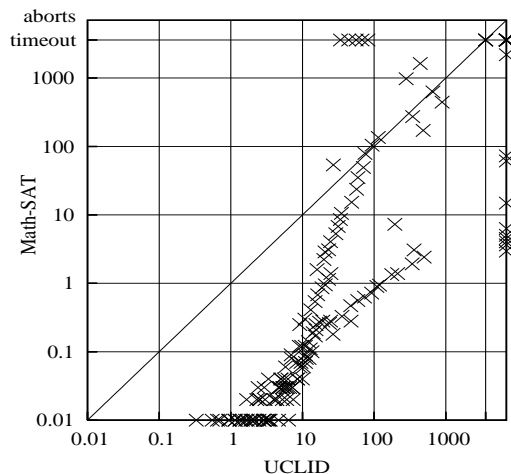
Scatter Graphs



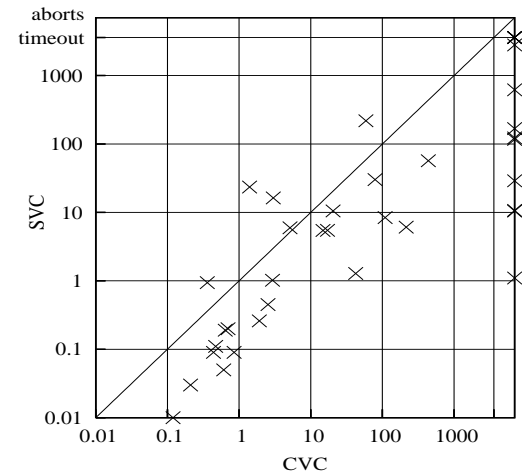
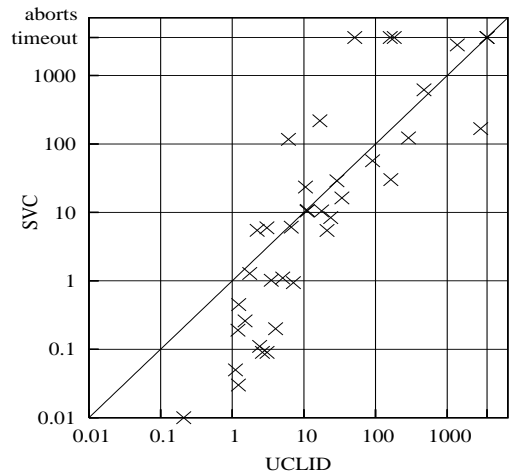
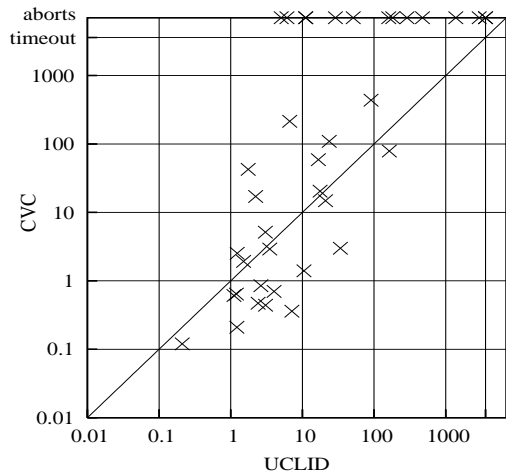
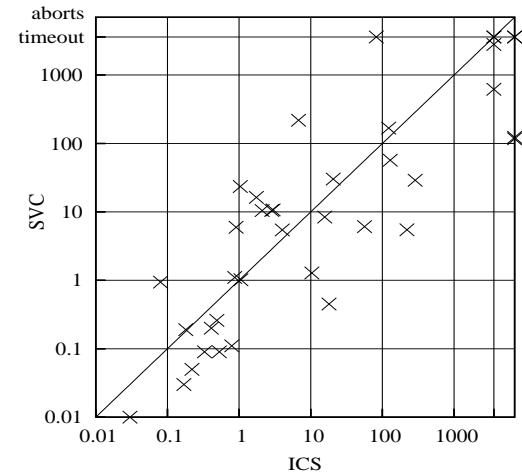
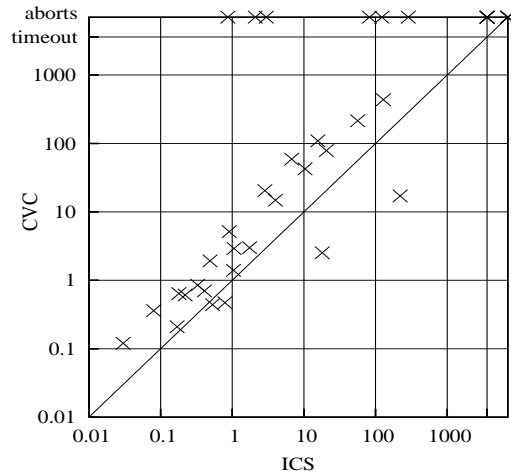
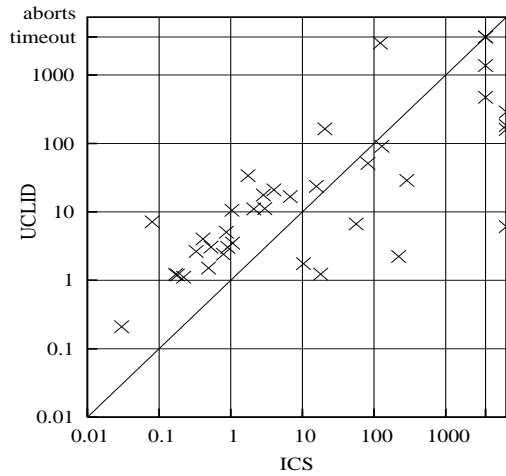
Results: Math-SAT benchmarks



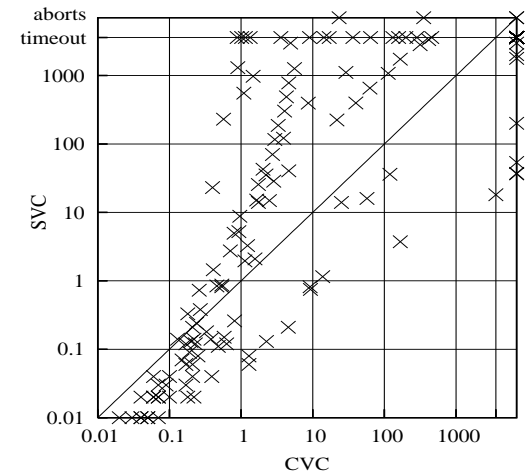
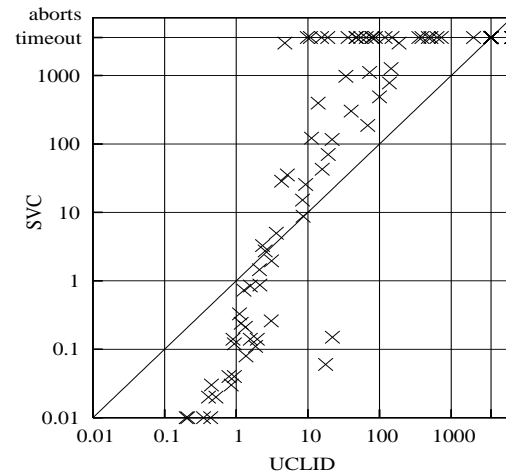
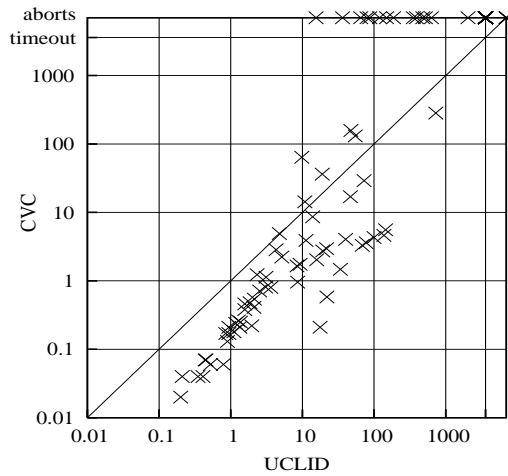
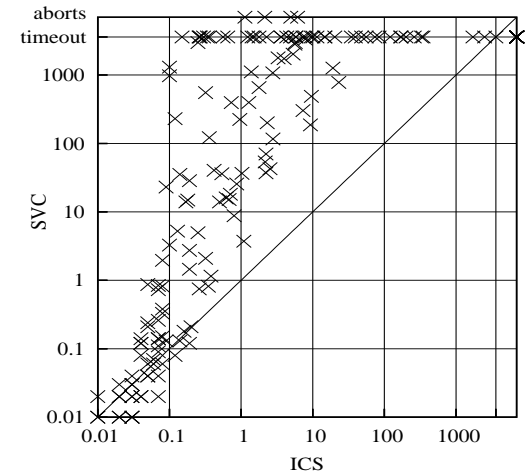
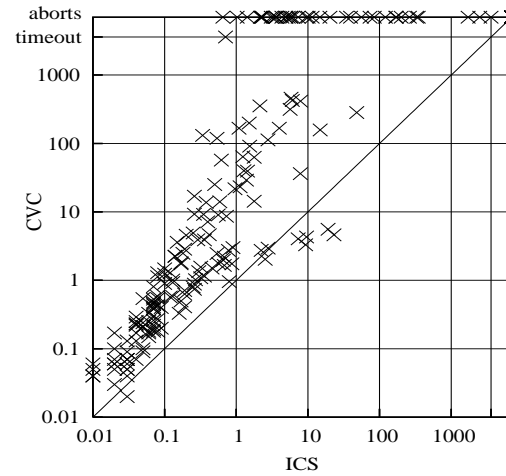
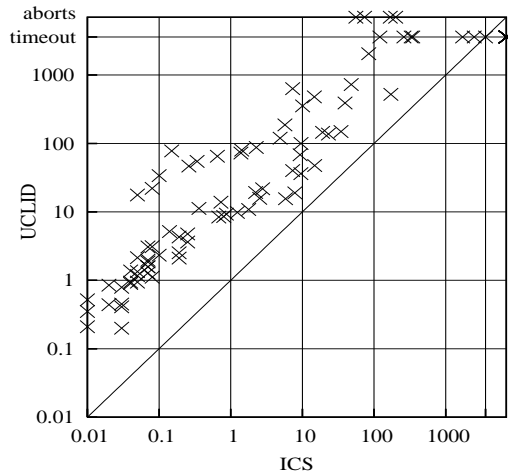
Results: Math-SAT benchmarks (cont.)



Results: UCLID benchmarks



Results: SAL benchmarks



Observations & Research Questions

- ▶ Insufficient constraint propagation in lazy integrations.
- ▶ Arithmetical constraints in the eager approach.
- ▶ Blind search problem in lazy solvers.
- ▶ Memory Usage.
- ▶ Loss of Structural Information.

Insufficient constraint propagation

- ▶ In most lazy systems, the constraint solver only **propagates** (detects) **inconsistencies**.
- ▶ Example:

$$\begin{aligned} &F_1[x = y] \wedge F_2[y = z] \wedge \dots \wedge \\ &(x = z \rightarrow p_2) \wedge (x = z \rightarrow p_3) \wedge \dots \\ &(\neg p_1 \vee \neg p_2 \vee \neg p_3) \wedge \dots \end{aligned}$$

- ▶ Assume: $x = y$, $y = z$ and p_1 are assigned to true.
- ▶ Then: $x = z$ could be assigned to true by constraint propagation (transitivity).
- ▶ Then: p_2 and p_3 are implied by $x = z$.
- ▶ Then: **Inconsistency** is detected.

Insufficient constraint propagation

- ▶ Exception: **Simplify** propagates equalities implied by transitivity, symmetry, and congruence.
- ▶ Tradeoff between precision and performance.

Arith. constraints in the eager approach

- ▶ UCLID usually performs worse in arithmetic-intensive problems.
- ▶ The performance is heavily affected by the size of constants.
- ▶ In several examples the whole memory was consumed in the translation to propositional logic.
- ▶ Extreme example: the problem `abz5-900` was easily solved by ICS and Simplify using less than 40Mb.

Blind search problem in lazy solvers

- ▶ Lazy systems introduce a fresh boolean variable for each distinct constraint.
- ▶ Consider: A problem where almost all atoms are constraints in \mathcal{T} and each constraint occurs only once.
- ▶ Lazy approach produces an **underconstrained** formula which contains several propositional variables which occurs only once.
- ▶ The beginning of the search is chaotic and arbitrary (**blind search**).
- ▶ Our hypothesis is corroborated by the experimental results.

Memory Usage

- ▶ Math-SAT and Simplify are the systems using the least amount of memory.
- ▶ CVC often aborts by running out of memory instead of running up its time limits.
- ▶ A similar phenomenon can be observed with ICS.
- ▶ We have traced this deficiency back to imprecise generation of **lemmas** for pruning Boolean assignments.
- ▶ Another problem in systems such as ICS and CVC is the maintenance of information to perform backtracking. Math-SAT is a non-backtrackable system, so it does not suffer from this problem.

Loss of Structural Information

- ▶ The performance of most solvers heavily depends on the way problems are encoded.
- ▶ Example: the repetitive application of the transformation

$$F[t] \implies F[x] \wedge x = t$$

transforms many easy problems into very hard problems for UCLID, CVC and CVC Lite.

Conclusions

- ▶ The performance study exposes specific weaknesses for each of the tools under consideration.
- ▶ Handling of arithmetic needs to be improved for eager systems.
- ▶ Lazy systems can be considerably improved by:
 - ▶ A tighter integration of constraint propagation.
 - ▶ Specialized search heuristics.
 - ▶ Generation of more precise lemmas (Justifying Equality, PDPAR'04).

Conclusions (cont.)

▶ A main impediment for future improvements in the field of GDPs, however, is not necessarily a lack of new algorithms in the field, but rather the limited availability of meaningful benchmarks.

▶ All benchmarks, translators, and results are available online at

<http://www.csl.sri.com/~demoura/gdp-benchmarks.html>.