TSAT++: SAT-Based Solver for Separation Logic

Marco Maratea (STAR-Lab)

j.w.w. A. Armando (AI-Lab)
C. Castellini (AI-Lab)
E. Giunchiglia (STAR-Lab)

Mechanised Reasoning Group
Dipartimento di Informatica, Sistemistica e Telematica
Università di Genova

Motivation

Decision procedures able to decide quantifier-free first-order theories are becoming increasingly important in Artificial Intelligence (AI) and Formal Verification (FV) areas.

Several properties of hardware, timed automata, and software can be modeled in quantifier-free first-order theories as well as planning and scheduling problems.

Separation Logic (SL) is one of such decidable quantifier-free first-order theories that allows boolean combination of difference constraints.

Due to the boost that SAT solvers had in the last years, the SAT-Based approach to decide these formulas has become a suitable and efficient way.

Why Separation Logic?

SL seems to be a good compromise between efficiency and expressivity.

It combines propositional atoms with a restricted form of linear arithmetic via the standard boolean connectives.

Many available benchmarks, both random, hand-made and real-life (following the usual SAT Competition division in categories) are in SL and a lot of properties of systems and planning/scheduling constraints can be encoded in this logic; i.e., bounded reachability of timed automata, existence of path in digital circuits with bounded delay and feasibility of scheduling problems.

SL: Definitions (1)

Fix a domain of interpretation D for the arithmetic variables (the set of real or the set of integer numbers).

An SL-atom is either a propositional variable or an SL-expression x - y $\leq$ c ($<, >, \geq, =, \neq$ can be (easily) recast in $\leq$), where x and y range on D and c is a numeric constant.

An SL-expression is also called difference constraint.

An SL-literal is an SL-atom or its negation.
An SL-clause is a finite disjunction of SL-literals.
An SL-formula is a finite conjunction of SL-clauses.

SL: Definitions (2)

We are restricting our attention on SL-formula in Conjunctive Normal Form (CNF): This is not a big problem as long as there are efficient algorithms for transforming a non-CNF in a CNF formula.

Deciding the satisfiability of an SL-formula (Is there an SL-assignment to propositional atoms and arithmetic variables such that the SL-formula is true?) is an NP-complete problem.

TSAT++: Yet another SAT-Based solver for SL?

Even TSAT++ follows the well-known (SAT-Based) lazy approach , it introduces (some of them new) ideas/optimization techniques like:

1. formula preprocessing
2. SAT solver partial assignments check (early pruning)
3. detection of the "best" witness of inconsistency
4. propositional assignment reduction (via prime implicants generation)

Using these techniques and combining them, TSAT++ can reach state-of-the-art results in a very wide range of domains of benchmarks arising from the AI and FV communities.

Agenda

- TSAT++'s overview
- TSAT++'s optimization techniques
- Experimental analysis
- Conclusions and future work

TSAT++'s architecture

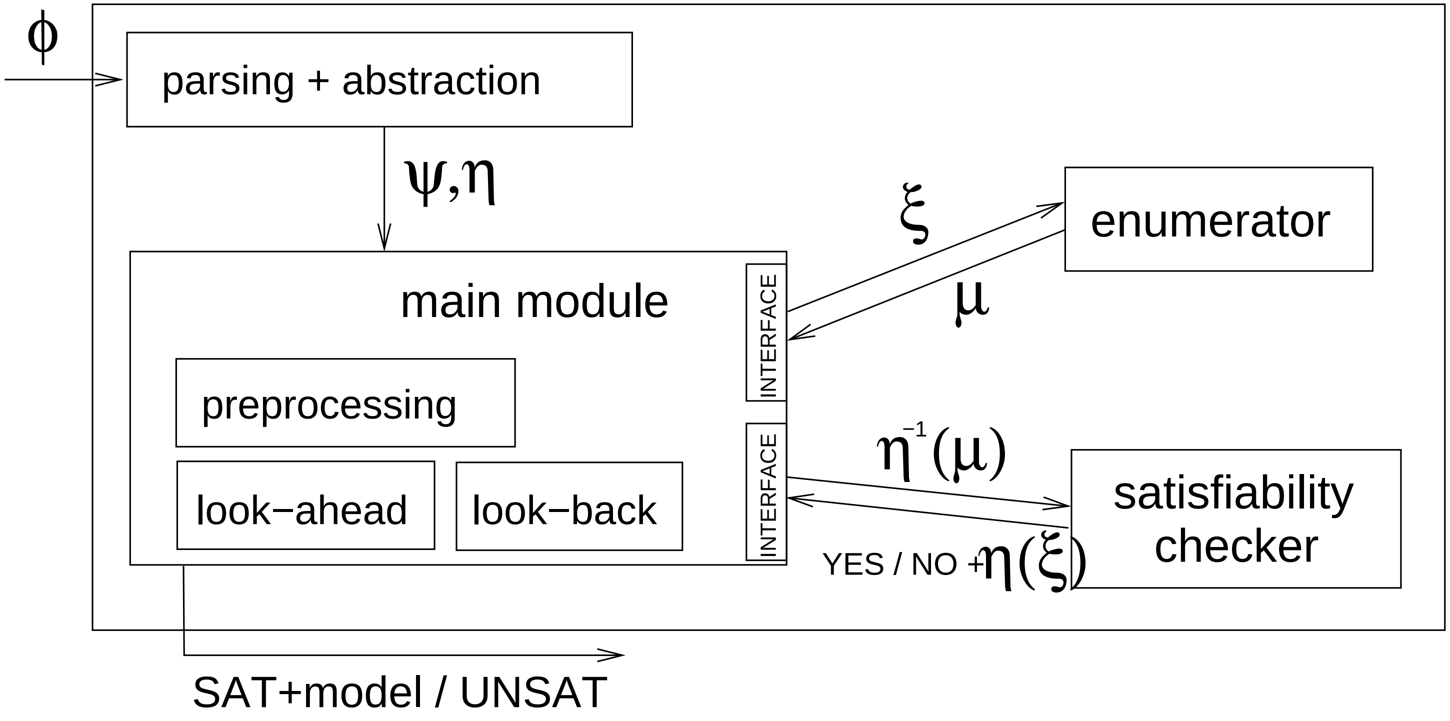

Figure 1: High-level view of TSAT++.

TSAT++'s approach

TSAT++ is implemented under the ACR (abstract-check-refine) paradigm:

1. first the SL-formula ϕ is abstracted to a boolean formula ψ;
2. the boolean models μs are checked for arithmetic consistency;
3. the boolean formula is refined using ξ in case of arithmetic inconsistency (and back to step. 2).

TSAT++ employs SIMO for the propositional part (ENUMERATOR), and a modified version of the Bellman-Ford (BF) algorithm for the arithmetic part (SATISFIABILITYCHECKER).

TSAT++'s basic algorithm

Given an SL-formula ϕ,

```
function TSAT++-Solve($\phi$)
    $< \psi, \eta >$ = Abstract&Map($\phi$)
    ENUMERATOR.LoadFormula($\psi$)
    while (($\mu$ = ENUMERATOR.Solve($\psi$)) != NULL)
        if (($\xi$ = SATISFIABILITYCHECKER.ConsistencyCheck($\mu$, $\eta$)) == NULL)
            return SAT
        ENUMERATOR.BacktrackWithReason($\xi$)
    return UNSAT
```

Working example

$$\{\, p_{12},\ \neg V\_Predecode_1 = I_R{}^R \}$$
$$\{\, \neg p_{12},\ V\_Predecode_1 = I_R{}^R \}$$
$$\{\, p_{14},\ \neg V\_Predecode_1 = I_R{}^R + 1 \}$$
$$\{\, \neg p_{14},\ V\_Predecode_1 = I_R{}^R + 1 \}$$
$$\{\, p_{15},\ p_{16} \}$$
$$\{\, p_{15}, \neg p_{16},\ \neg p_{17} \}$$
$$\{\, p_{19},\ \neg en\_ICLASS' + 2 = en\_ICLASS \}$$
$$\{\, \neg p_{19},\ en\_ICLASS' + 2 = en\_ICLASS \}$$

from the design of the bounded model checking problem for the memory unit of the Motorola Elf microprocessor

Working example: Boolean formula and map

ψ	η
$\{\ p_{12},\ \neg\mathbf{p}_{100}\}$	
$\{\ \neg p_{12},\ \mathbf{p}_{100}\}$	
$\{\ p_{14},\ \neg\mathbf{p}_{101}\}$	
$\{\ \neg p_{14},\ \mathbf{p}_{101}\}$	$\mathbf{p}_{100} \mapsto V\_P_{r}edecode_1 = {I_R}^R$
$\{\ p_{15},\ p_{16}\}$	$\mathbf{p}_{101} \mapsto V\_P_{r}edecode_1 = {I_R}^R + 1$
$\{\ p_{15}, \neg p_{16},\ \neg p_{17}\}$	$\mathbf{p}_{102} \mapsto en\_ICLASS' + 2 = en\_ICLASS$
$\{\ p_{19},\ \neg\mathbf{p}_{102}\}$	
$\{\ \neg p_{19},\ \mathbf{p}_{102}\}$	

Optimization 1: Preprocessing

One drawback of the generate-and-test approach is that (exponentialy) many trivialy inconsistent valuations can be generated and then discarded (e.g., with $x - y \leq 3$ assigned to true and $x - y \leq 5$ to false)

To reduce this drawback, in TSAT++ for each pair c_1, c_2 of difference constraints in the same variables and occurring in ϕ, the consistency of all possible pairs of literals built out of them, i.e., $\{c_1, c_2\}$, $\{\neg c_1, c_2\}$, $\{c_1, \neg c_2\}$, and $\{\neg c_1, \neg c_2\}$, is checked.

Assuming, e.g., $\{c_1, c_2\}$ is inconsistent, the clause $\{\neg c_1, \neg c_2\}$ is added to ϕ before the search starts. (in our ex., we would add the clause $\{\neg x - y \leq 3, x - y \leq 5\}$).

This dramaticaly speeds-up the search, especialy on randomly generated problems. In fact, e.g., as soon as $x - y \leq 3$ is assigned to true, $x - y \leq 5$ gets also assigned to true by unit propagation.

TSAT++'s algorithm with opt 1.

Given an SL-formula ϕ,

```
function TSAT++-Solve($\phi$)
    $< \psi, \eta >$ = Abstract&Map($\phi$)
    $\psi$= Preprocessing($\psi, \eta$)
    ENUMERATOR.LoadFormula($\psi$)
    while (($\mu$ = ENUMERATOR.Solve($\psi$)) != NULL)
        if (($\xi$ = SATISFIABILITYCHECKER.ConsistencyCheck($\mu$, $\eta$)) == NULL)
            return SAT
        ENUMERATOR.BacktrackWithReason($\xi$)
    return UNSAT
```

Working example (1)

The technique is not directly applicable in our example

Working example (1)

The technique is not directly applicable in our example, but consider the straightforward translation:

$$\{\ p_{12},\ V\_P_{r}edecode_1 < I_R{}^R,\ V\_P_{r}edecode_1 > I_{RR}\}$$
$$\{\ \neg p_{12},\ V\_P_{r}edecode_1 \leq I_R{}^R\}\ \{\ \neg p_{12},\ V\_P_{r}edecode_1 \geq I_R{}^R\}$$
$$\{\ p_{14},\ V\_P_{r}edecode_1 < I_R{}^R + 1,\ V\_P_{r}edecode_1 > I_R{}^R + 1\}$$
$$\{\ \neg p_{14},\ V\_P_{r}edecode_1 \leq I_R{}^R + 1\}\ \{\ \neg p_{14},\ V\_P_{r}edecode_1 \geq I_R{}^R + 1\}$$
$$\{\ p_{15},\ p_{16}\}$$
$$\{\ p_{15}, \neg p_{16},\ \neg p_{17}\}$$
$$\{\ p_{19},\ en\_ICLASS' + 2 < en\_ICLASS,\ en\_ICLASS' + 2 > en\_ICLASS\}$$
$$\{\ \neg p_{19},\ en\_ICLASS' + 2 \leq en\_ICLASS\}$$
$$\{\ \neg p_{19},\ en\_ICLASS' + 2 \geq en\_ICLASS\}$$

Working example (2)

All the arithmetic constraints are “translated” to difference constraints and the difference constraints involving the same arithmetic variables are tested for consistency.

For example, $V\_Predecode_1 < I_R{}^R$ is inconsistent with $V\_Predecode_1 \geq I_R{}^R$, $V\_Predecode_1 > I_R{}^R + 1$ and $V\_Predecode_1 \geq I_R{}^R + 1$, and the related propositional clauses

$$
\begin{array}{l}
\{\neg V\_Predecode_1 < I_R{}^R,\ \neg V\_Predecode_1 \geq I_R{}^R\} \\
\{\neg V\_Predecode_1 < I_R{}^R,\ \neg V\_Predecode_1 > I_R{}^R + 1\} \\
\{\neg V\_Predecode_1 < I_R{}^R,\ \neg V\_Predecode_1 \geq I_R{}^R + 1\}
\end{array}
$$

are added to ψ (obviously, each difference constraint must be substituted with the correspondent propositional atom).

Working example (3)

Nevertheless, in the experimental part of the talk, we will see that there are some well-known benchmarks from the AI community, called DTP, with the following structure:

$$\begin{array}{l}
x_3 - x_1 \leq 68 \;\vee\; x_1 - x3 \leq -15 \;\wedge \\
x_0 - x_3 \leq 42 \;\vee\; x_0 - x_1 \leq 80 \;\wedge \\
x_0 - x_4 \leq -42 \;\vee\; x_2 - x_4 \leq 34 \;\wedge \\
x_3 - x_2 \leq -60 \;\vee\; x_0 - x_2 \leq 15 \;\wedge \\
x_0 - x_3 \leq -60 \;\vee\; x_0 - x_4 \leq -4
\end{array}$$

This is also an instance of SL!
(even if not very likely to obtain :-)

Optimization 2: Early pruning

In CSP, this technique has been shown to be very effective on randomly generated problems.

The SAT solver must have the feature of returning also partial (but consistent) boolean model μ_p (even they do not satisfy yet ψ), other than boolean model μ satisfying ψ.

If μ_p leads to an arithmetic inconsistency, there is no need to go on this branch, and the procedure can backtrack.

If μ_p does not lead to an arithmetic inconsistency, we have to go on this branch; if μ_p was satisfying, we are done.

TSAT++'s algorithm with opt 2.

Given an SL-formula ϕ,

```
function TSAT++-Solve($\phi$)
    $< \psi, \eta >$ = Abstract&Map($\phi$)
    ENUMERATOR.LoadFormula($\psi$)
    while (($\mu_p$ = ENUMERATOR.Solve($\psi$)) != NULL)
        if (($\xi$ = SATISFIABILITYCHECKER.ConsistencyCheck($\mu_p$, $\eta$)) == NULL)
            if (IsSatisfying($\mu_p$))
                return SAT
        else
            ENUMERATOR.BacktrackWithReason($\xi$)
    return UNSAT
```

Working example

ψ	η
$\{\ p_{12},\ \neg\mathbf{p_{100}}\}$	
$\{\ \neg p_{12},\ \mathbf{p_{100}}\}$	
$\{\ p_{14},\ \neg\mathbf{p_{101}}\}$	
$\{\ \neg p_{14},\ \mathbf{p_{101}}\}$	$\mathbf{p_{100}} \quad \mapsto \quad V\_P_r{}^{edecode_1} = I_R{}^R$
$\{\ p_{15},\ p_{16}\}$	$\mathbf{p_{101}} \quad \mapsto \quad V\_P_r{}^{edecode_1} = I_R{}^R + 1$
$\{\ p_{15}, \neg p_{16},\ \neg p_{17}\}$	$\mathbf{p_{102}} \quad \mapsto \quad en\_ICLASS' + 2 = en\_ICLASS$
$\{\ p_{19},\ \neg\mathbf{p_{102}}\}$	
$\{\ \neg p_{19},\ \mathbf{p_{102}}\}$	

Consider $\mu_p = \{p_{12}, \mathbf{p_{100}}, p_{14}, \mathbf{p_{101}}\}$. Clearly, $\{\mathbf{p_{100}}, \mathbf{p_{101}}\}$ is an inconsistent set. There is no need to go on this branch, we can backtrack.

Optimization 3: Detecting reason (1)

Slightly modifying the BF algorithm, after the detection of a negative cycle, we are able to extract the "best" reason ξ under given condition looking among the available cycles.

The following three options are currently implemented in TSAT++:

- plain: pick the first reason non-deterministicaly
- shortest: pick the minimal reason under cardinality
- shallowest: pick the minimal reason under the order induced by the propositional assignment (stack)

TSAT++'s algorithm with opt 3.

Given an SL-formula ϕ,

```
function TSAT++-Solve($\phi$)
    $< \psi, \eta >$ = Abstract&Map($\phi$)
    ENUMERATOR.LoadFormula($\psi$)
    while (($\mu$ = ENUMERATOR.Solve($\psi$)) != NULL)
        if (($\xi$ = SATISFIABILITYCHECKER.ConsistencyCheck($\mu$, $\eta$)) == NULL)
                return SAT
        ENUMERATOR.BacktrackWithReason($\xi$)
    return UNSAT
```

Working example

ψ	η
$\{\ p_{12},\ \neg\mathbf{p_{100}}\}$	
$\{\ \neg p_{12},\ \mathbf{p_{100}}\}$	
$\{\ p_{14},\ \neg\mathbf{p_{101}}\}$	
$\{\ \neg p_{14},\ \mathbf{p_{101}}\}$	$\mathbf{p_{100}} \quad \mapsto \quad V\_P_{r}^{edecode_1} = I_R{}^R$
$\{\ p_{15},\ p_{16}\}$	$\mathbf{p_{101}} \quad \mapsto \quad V\_P_{r}^{edecode_1} = I_R{}^R + 1$
$\{\ p_{15}, \neg p_{16},\ \neg p_{17}\}$	$\mathbf{p_{102}} \quad \mapsto \quad en\_ICLASS' + 2 = en\_ICLASS$
$\{\ p_{19},\ \neg\mathbf{p_{102}}\}$	
$\{\ \neg p_{19},\ \mathbf{p_{102}}\}$	

Consider the satisfying $\mu := \{p_{12}, \mathbf{p_{100}}, p_{14}, \mathbf{p_{101}}, p_{15}, p_{16}, \neg p_{17}, p_{19}, \mathbf{p_{102}}\}$.
This is not a “good” model, because the set $\{\mathbf{p_{100}}, \mathbf{p_{101}}\}$ is inconsistent.
In this case, we will refine the formula with the clause $\{\neg\mathbf{p_{100}}, \neg\mathbf{p_{101}}\}$

Opt. 3: Generalizing reason detection (2.1)

The example before is naive. In general, given an inconsistent assignment $\mu = \{l_1, \ldots, l_n\}$, and the set of all possible sources of inconsistency $\{\xi_1, \ldots, \xi_j\}$,

- $\xi_k \in \{\xi_1, \ldots, \xi_j\}$ is the shortest reason iff $\forall \xi_i \in \{\xi_1, \ldots, \xi_j\}, 0 < i \leq j$, $|\xi_i| \geq |\xi_k|$
- $\xi_k \in \{\xi_1, \ldots, \xi_j\}$ is the shallowest reason iff is the minimal under lexicographical order.

Opt. 3: Generalizing reason detection (2.2)

Example

Consider $\mu = \{l_1, \ldots, l_{50}\}$, with the index denoting the position the literal has been assigned in the propositional stack, and $\xi_1 := \{l_{10}, l_{17}, l_{35}\}$, $\xi_2 := \{l_{15}, l_{45}\}$ and $\xi_3 := \{l_{20}, l_{30}, l_{40}\}$.

Clearly, the shortest reason is ξ_2, but the shallowest is ξ_1 because $35 < 40 < 45$.

Notice that, if we had a further reason $\xi_4 := \{l_7, l_{10}, l_{12}, l_{35}\}$, it would have been chosen as shallowest, since the literal in it that was assigned *second*-last has a lower index than that in ξ_1 $(12$ versus $17)$.

Optimization 3: Detecting reason (3)

In the analysis, we have used the “shortest” option because it seems to lead to slight better results.

Nevertheless, the differences between reasons are not considerable, because there are very few available negative cycles for each failure.

This is due to the single-source nature of the BF.

Optimization 4: Assignment reduction

Given μ propositional satisfying ψ, it may be the case that some of the literals in μ may be not necessary to satisfy ψ. This is in particular true when SAT solvers use lazy data structure like watched literals.

We compute a *prime implicant* μ_r of the formula ψ such that $\mu_r \subseteq \mu$.

We call the above procedure *reduction*, and it may be useful because

– if μ leads to a satisfying SL-assignment ,so is μ_r, and we are done;
– if μ does not lead to a satisfying SL-assignment, it may nevertheless be the case that μ_r does, and we can still interrupt the search because we are done;
– if μ and μ_r do not lead to a satisfying SL-assignment, checking the consistency of μ instead of μ_r can cause exponentially many more consistency checks.

TSAT++'s algorithm with opt 4.

Given an SL-formula ϕ,

```
function TSAT++-Solve($\phi$)
    $< \psi, \eta >$ = Abstract&Map($\phi$)
    ENUMERATOR.LoadFormula($\psi$)
    while (($\mu$ = ENUMERATOR.Solve($\psi$)) != NULL)
        $\mu_r$ = reduction($\mu$)
        if (($\xi$ = SATISFIABILITYCHECKER.ConsistencyCheck($\mu_r$, $\eta$)) == NULL)
                return SAT
        ENUMERATOR.BacktrackWithReason($\xi$)
    return UNSAT
```

Working example

ψ	η
$\{\ p_{12},\ \neg\mathbf{p}_{100}\}$	
$\{\ \neg p_{12},\ \mathbf{p}_{100}\}$	
$\{\ p_{14},\ \neg\mathbf{p}_{101}\}$	
$\{\ \neg p_{14},\ \mathbf{p}_{101}\}$	$\mathbf{p}_{100} \mapsto V\_Predecode_1 = I_R{}^R$
$\{\ p_{15},\ p_{16}\}$	$\mathbf{p}_{101} \mapsto V\_Predecode_1 = I_R{}^R + 1$
$\{\ p_{15}, \neg p_{16},\ \neg p_{17}\}$	$\mathbf{p}_{102} \mapsto en\_ICLASS' + 2 = en\_ICLASS$
$\{\ p_{19},\ \neg\mathbf{p}_{102}\}$	
$\{\ \neg p_{19},\ \mathbf{p}_{102}\}$	

Consider the satisfying assignment $\mu := \{\neg p_{12}, \neg\mathbf{p}_{100}, p_{14}, \mathbf{p}_{101}, p_{15}, p_{16}, \neg p_{17}, p_{19}, \mathbf{p}$ μ is reduced to $\mu_r := \{\neg p_{12}, \neg\mathbf{p}_{100}, p_{14}, \mathbf{p}_{101}, p_{15}, p_{19}, \mathbf{p}_{102}\}$ by noting that p_{16} and p_{17} does not contribute to the satisfiability of ϕ.

Another example: Degenerate instance

In the case above, the number of binary constraints processed by the satisfiability checker remains the same, since model reduction has excluded from μ propositional atoms only; but this is not the case in general.

Further, consider the following instance:

$$\begin{array}{l}
\{\ a,\ x1 - x2 \leq 4,\ x3 - x4 > 6,\ \ldots\ \} \\
\{\ a,\ x1 - x2 > 4,\ x3 - x4 \leq 6,\ \ldots\ \} \\
\{\ a,\ \ldots\ \} \\
\{\ \ldots,\ \ldots\ \} \\
\{\ a,\ \ldots\ \}
\end{array}$$

with a can also be a constraint, and there are an exponential number of inconsistencies among all the constraints but a.

Another example: Degenerate instance (2)

Checking the satisfiability of μ without reduction can cause an exponential number of checks before leading to the solution.

The reduction of *any* μ containing a (note that most of the SAT heuristic would assign $a := true$) will lead to $\mu_r := \{a\}$, which is immediately satisfiable.

Experimental settings

For all the solvers:
TIME : 1000 sec
MEM : 512MB
"–" : segmentation fault

In TSAT++:
– j : preprocessing
– p : prime implicant generation (assignment reduction)
– 2 : shortest reason

SEP-m : SEP with internal conjunction matrix off (suggested by O. Strichman)

Disjunctive Temporal Problem (DTPs)

These are well-known random problems from the AI community, with applications in scheduling and management of temporal databases.

DTPs are randomly generated by fixing the number k of expressions x - y $\leq$ c per SL-clause, the number n of arithmetic variables, a positive integer L such that all the constants are taken in $[-L, L]$. Then:

1. the number of clauses m is increased in order to range from satisfiable to unsatisfiable instances from 2*n to 14*n step n,
2. for each tuple of values of the parameters, 100 instances are generated and then given to the solvers, and
3. the median of the CPU time is plotted against the m/n ratio.

DTP instance

$\#$
$\#k = 2, L = 100, n = 5, m = 10$
$\#$

$$
\begin{array}{lllll}
x_3 - x_1 \leq & 68 & \vee & x_1 - x3 \leq & -15 \quad \wedge \\
x_0 - x_3 \leq & 42 & \vee & x_0 - x_1 \leq & 80 \quad \wedge \\
x_0 - x_4 \leq & -42 & \vee & x_2 - x_4 \leq & 34 \quad \wedge \\
x_3 - x_2 \leq & -60 & \vee & x_0 - x_2 \leq & 15 \quad \wedge \\
x_0 - x_3 \leq & -60 & \vee & x_0 - x_4 \leq & -4 \quad \wedge \\
x_4 - x_3 \leq & 21 & \vee & x_1 - x_4 \leq & -18 \quad \wedge \\
x_2 - x_4 \leq & 50 & \vee & x_1 - x_4 \leq & -20 \quad \wedge \\
x_4 - x_0 \leq & -38 & \vee & x_3 - x_4 \leq & 50 \quad \wedge \\
x_0 - x_1 \leq & 52 & \vee & x_1 - x_0 \leq & 27 \quad \wedge \\
x_0 - x_1 \leq & 88 & \vee & x_0 - x_3 \leq & 82
\end{array}
$$

`TSAT++`'s performances (1.1): DTPs

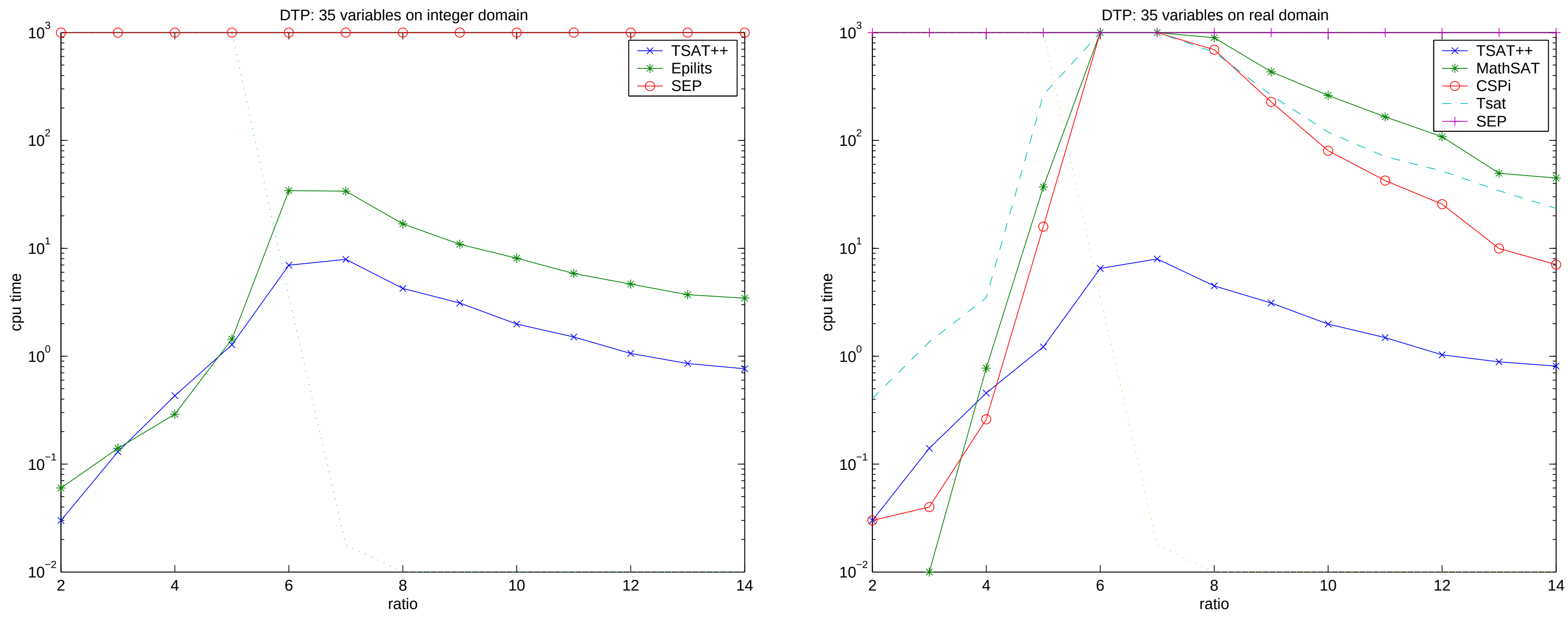

Figure 2: Evaluation on the DTP on 35 variables. Integer domain (left) and real domain (right). Setting: $k = 2$, $L = 100$.

TSAT++'s scalability (1.2): DTPs

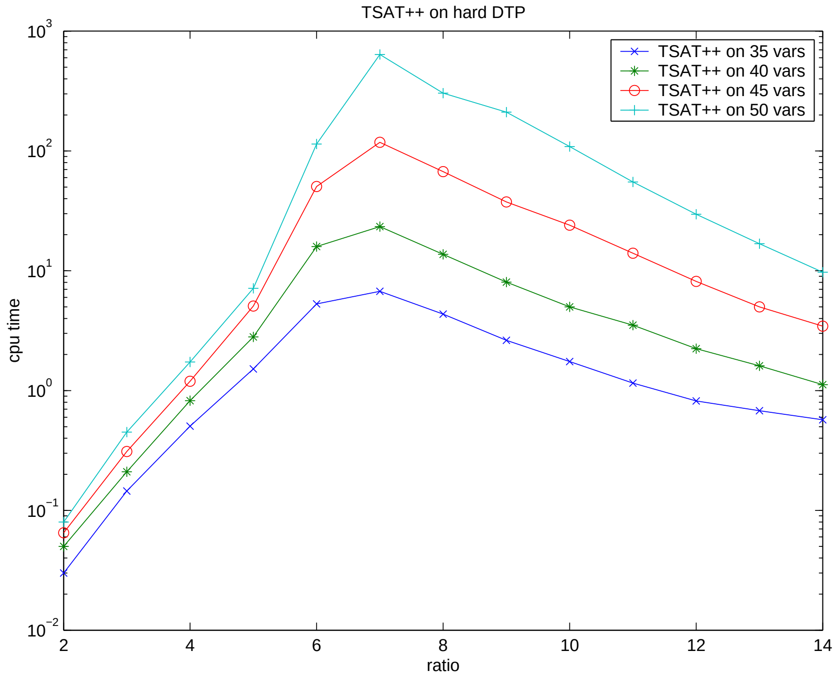

Figure 3: TSAT++ scalability on 35, 40, 45, 50 variables. Real domain. Setting: $k = 2$, $L = 100$.

Post-office problem: Description

"Consider a post-office with N desks, each desk serving a customer every T seconds. Every new customer chooses the desk with shorter queue and, when more than one queue has minimal length, the minimal queue with the minimum index. It is not possible to change a queue after entering it. We want to prove that, although all desks have the same serving time T and customers are "smart", one customer can get into the annoying situation of finding himself in queue after one person, whilst all other queue are empty, and having to wait for a non-instantaneous period in this situation"

TSAT++'s performance (2): Post-office problems

Instance	SAT?	TSAT++ jp2	MathSAT	SEP
P04-6-P04	NO	0.07	0.36	16.02
P04-7-P04	NO	0.11	0.36	134.21
P04-11-P04	NO	1.01	2.13	TIME
P04-12-P04	YES	0.58	0.91	TIME
P05-10-P05	NO	2.41	5.32	–
P05-11-P05	NO	3.44	9.23	–
P05-12-P05	NO	4.79	22.06	–
P05-13-P05	NO	8.88	54.17	–
P05-14-P05	YES	2.99	11.36	–

Diamonds problems

Given a parameter D (number of diamonds), these problems are characterized by an exponentially large (2^D) number of boolean models μ, some of which correspond to satisfying SL-assignments; hard instances with a unique "good" propositional satisfying assignment can be generated.

A second parameter, S (related to the number of edge in each diamond), is used to make μ larger, further increasing the difficulty.

Variables range over the reals.

Diamond problem graphical representation

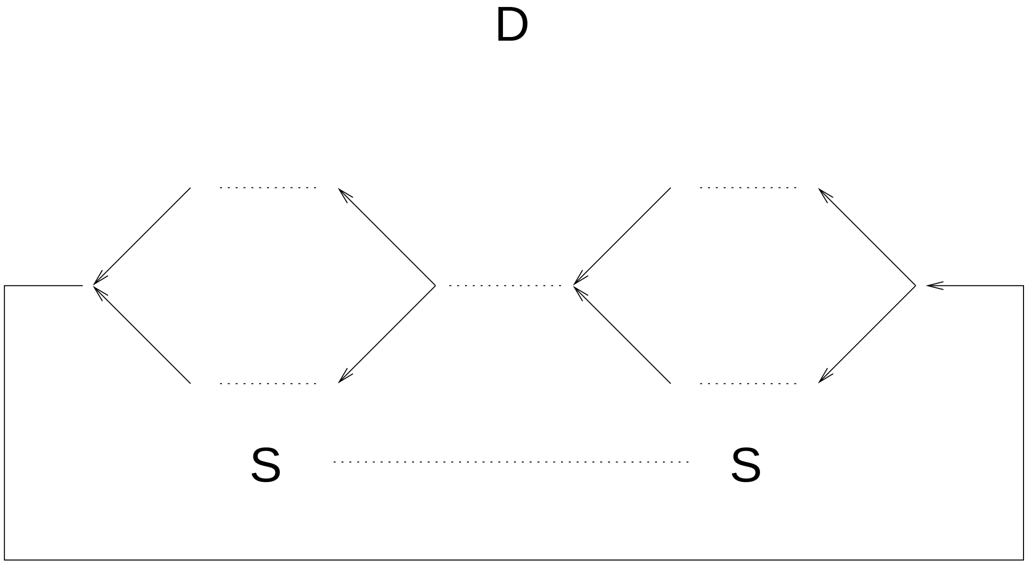

Figure 4: Diamond graphical representation.

`TSAT++`'s performance (3): Diamonds problems

Instance			**Lazy**				**Eager**	
D	S	u?	TSAT++ p2	M.SAT	ICS	CVC	SEP	SEP-m
250	5	NO	0.08	5.40	0.05	MEM	52.20	0.95
250	5	YES	0.21	TIME	150.02	3.26	0.77	288.30
500	5	NO	0.29	21.22	0.11	MEM	742.99	5.92
500	5	YES	1.05	TIME	MEM	6.99	4.85	TIME
1000	5	NO	1.07	–	0.28	MEM	TIME	27.52
1000	5	YES	6.45	–	MEM	15.68	22.53	TIME
2000	5	NO	3.76	–	0.82	MEM	–	–
2000	5	YES	29.90	–	MEM	37.53	–	–

Real-world problems from UCLID: Description

These very same benchmarks are publicly available at the UCLID homepage, where they are formulated in CLU (Counter arithmetic with Lambda expressions and Uninterpreted functions) logic.

The benchmarks were kindly translated in SL (as DAG) by Sanjit Seshia.

The instances include problems about cache coherence protocol, load-store unit and out-of-order execution unit.

TSAT++'s performance (4): Real-world problems from UCLID

Instance	**Lazy**		**Eager**
	TSAT++ p2	ICS	SEP
cache.inv10	0.11	5.29	–
cache.inv12	75.08	53.83	–
dlx1c	TIME	MEM	–
elf.rf8	0.74	2.68	MEM
elf.rf9	13.92	39.24	TIME
ooo.rf7	7.42	16.26	MEM
ooo.rf8	231.80	265.16	TIME
q2.14	230.69	479.65	–

Conclusions

TSAT++ is a SAT-Based solver for the boolean combination of difference constraints that follows the lazy approach.

It introduces various ideas/optimization techniques; most of them are theory-independent.

Using and combining these techniques, TSAT++ is faster (sometimes significantly) than other state-of-the-art solvers in different domains arising from AI and FV communities.

References

The material presented here and information about TSAT++ can be found at:

http://www.ai.dist.unige.it/Tsat/

Alessandro Armando, Claudio Castellini, Enrico Giunchiglia, Marco Maratea
A SAT-based Decision Procedure for the Boolean Combination of Difference Constraints.
Accepted to SAT 2004. 10-13 May, Vancouver, Canada.

Armando Armando, Claudio Castellini, Enrico Giunchiglia, Massimo Idini, Marco Maratea
TSAT++: An Open Reasoning Platform for Satisfiability Modulo Theory.
Accepted to PDPAR 2004. 5 July 2004, Cork, Ireland.

Some recent results on TSAT++ (1)

More detailed evaluation on SAT heuristics and early pruning periodicity.

SAT heuristics:

- basic Chaff heuristic (VSIDS)
- basic Chaff heuristic with the chosen variables assigned by default to *false*
- SATZ-like heuristic, more (quadratic) reasoning done at each node (Unit heuristic)

Early pruning periodicity:

- early pruning can be too expensive sometimes; the ConsistencyCheck is not performed at each node, but with a given periodicity n.

Some recent results on TSAT++ (2): DTP

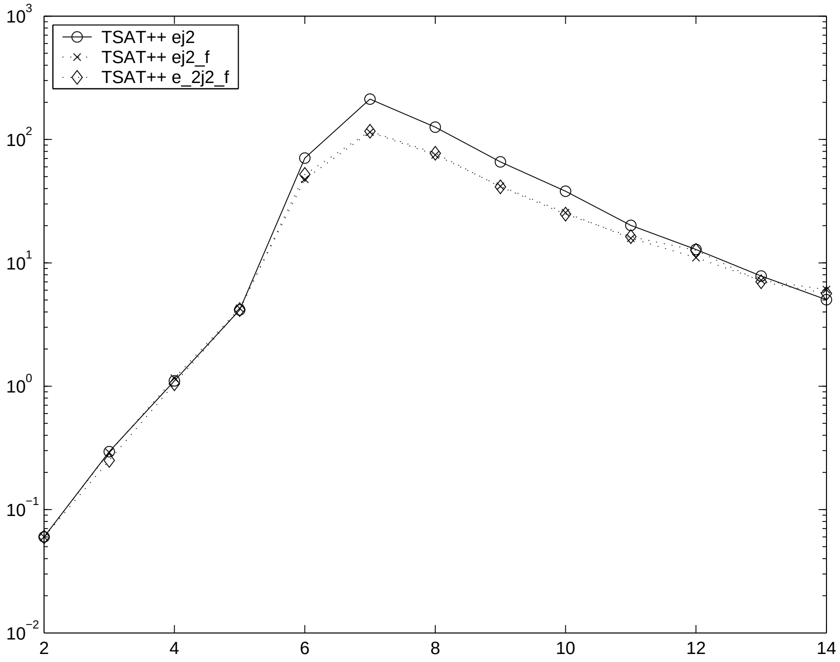

Figure 5: TSAT++ performances on DTPs with 45 vars; further analysis.

Some recent results on TSAT++ : Post-office

Instance	SAT?	TSAT jp2	TSAT$_u$	TSAT jp2$_u$	MathSAT	SEP
P04-6-P04	NO	0.07	0.01	0.07	0.36	16.02
P04-7-P04	NO	0.11	0.02	0.09	0.36	134.21
P04-11-P04	NO	1.01	1.08	0.76	2.13	TIME
P04-12-P04	YES	0.58	1.71	0.52	0.91	TIME
P05-10-P05	NO	2.41	0.05	0.71	5.32	–
P05-11-P05	NO	3.44	0.3	0.8	9.23	–
P05-12-P05	NO	4.79	2.27	1.62	22.06	–
P05-13-P05	NO	8.88	6.52	5.65	54.17	–
P05-14-P05	YES	2.99	11.15	7.01	11.36	–

Recent results . . .

. . . seem to point out that

- TSAT++ is not optimized for performances in each problem domain, and . . .

- some “basic” settings are far from being obvious and/or optimal,

- tuning other “basic” parameters can probably enhance performances ulteriorly (SAT heuristic periodicity, SAT solver forgetting clauses policy, . . .)

Considerations & Events

A considerably amount of time has been devoted to the “translation” among input formats.

This slide is also a “call” for a common input format. a

The SMT (Satisfiability Modulo Theory) initiative (by Silvio Ranise and Cesare Tinelli) is pushing in this direction with a proposal for a common language.

The upcoming competition (coordinate by Clark Barrett and Aaron Stump) hopefully will accelerate this process.

PDPAR05, July 2005 (Program Co-Chairs Alessandro Armando and Alessandro Cimatti) probably will be held in conjunction with CAV05.

SAT2005 Special Issues on the Journal of Automated Reasoning (JAR) (edited by Enrico Giunchiglia and Toby Walsh)

Ongoing work (1)

From the point of view of the basic research, extending TSAT++'s theory with

- (full) linear arithmetic,
- arrays

Ongoing work (2)

On the "applications" side, using TSAT++ as an effective back-end solver for:

- Software Model Checking
  Alessandro Armando, Claudio Castellini and Jacopo Mantovani
  Software Model Checking using Linear Programs.
  Accepted to ICFEM 2004

- Planning/Scheduling

  "Activity A1 lasts for 10 units of time at most": $e_1 - s_1 \leq 10$

  "Activity A1 should start before activity A2 finishes": $s_1 \leq e_2$

  "Activity A1 should start before activity A2 finishes, otherwise A3 should start when A2 finishes": $s_1 \leq e_2 \vee s_3 = e_2$