Implication Graph Compression inside the SMT Solver iSAT3*

Karsten Scheibler
Albert Ludwigs Universität
Freiburg
scheibler@informatik.uni-freiburg.de

Bernd Becker
Albert Ludwigs Universität
Freiburg
becker@informatik.uni-freiburg.de

Abstract

The iSAT algorithm aims at solving boolean combinations of linear and non-linear arithmetic constraint formulas (including transcendental functions), and thus is suitable to verify safety properties of systems consisting of both, linear and non-linear behaviour. The iSAT algorithm tightly integrates interval constraint propagation into the conflict-driven clause-learning framework. During the solving process, this may result in a huge implication graph. This paper presents a method to compress the implication graph on-the-fly. Experiments demonstrate that this method is able to reduce the overall memory footprint up to an order of magnitude.

1. Introduction

Computer-aided verification techniques have been intensively studied over the past decade and are becoming increasingly accepted by the industry. Applications range from hardware verification of digital circuits to the analysis of embedded systems. Nowadays, embedded systems are a hybrid of digital components and additional analog parts (e.g. a digital processor controlling a robotic arm and reacting to analog sensor inputs). An increasing number of applications in particular in the verification area are applying SAT Modulo Theory solvers for finding errors or proving the absence of errors in such systems. When modeling hybrid systems, boolean combinations of non-linear arithmetic functions (including transcendental functions like $\sin$, $\cos$, and $\exp$) arise naturally. That is one main reason the iSAT algorithm has been invented in [FHT+07, Her11]. This algorithm can be used for determining the satisfiability of formulas containing arbitrary boolean combinations of linear and non-linear arithmetic constraints.

*This work has been partially supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (DFG, SFB/TR 14 AVACS, http://www.avacs.org/) and by the Cluster of Excellence BrainLinks-BrainTools (DFG, grant number EXC 1086, http://www.brainlinks-braintools.uni-freiburg.de/)

The iSAT algorithm extends the conflict-driven clause-learning (CDCL) framework – successfully used in propositional SAT solving – by seamlessly integrating interval constraint propagation (ICP) into it. Therefore, deductions made by ICP are also recorded in the implication graph used by CDCL to allow efficient backtracking and conflict clause creation. Depending on the input formula ICP may produce long deduction sequences resulting in a huge implication graph. In this paper we present an approach to reduce the size of the implication graph considerably and thus decrease the overall memory footprint.

The paper is structured as follows. After introducing the iSAT algorithm in Section 2, we give a motivating example in Section 3, before describing our method to compress the implication graph in Section 4. In Section 5 we discuss the experimental results and conclude with a summary in Section 6.

2. Preliminaries

2.1. CDCL, Resolution, SMT

Given a propositional formula and asking the question if there exists an assignment to its variables rendering the formula `true` is also known as the *satisfiability problem*. Programs for solving this kind of problem are called SAT solvers. A very simple approach to tackle the satisfiability problem for a given formula is to try every possible assignment to its variables systematically. For a formula containing n variables this would require to try up to 2^n different variable assignments. With $n > 100$ this method won't give an answer in reasonable time.

Fifty years ago DPLL [DLL62] was proposed. This method exploits that every boolean formula can be rewritten to an equi-satisfiable *conjunctive normal form* (CNF) using the Tseitin-transformation [Tse68]. A CNF consists of a conjunction of clauses with each clause being a disjunction of literals and a literal being a boolean variable or its negation. To satisfy a formula in CNF every clause in the formula has to be satisfied. As a consequence, if an assignment to a subset of the variables already results in an unsatisfied clause (a so-called *conflict*) every assignment containing this subset is known to not satisfy the formula. This results in a huge reduction of variable assignments to be tested.

Additionally *boolean constraint propagation* (BCP) is used to detect implied assignments. Everytime a clause with n literals contains $n - 1$ literals being already assigned to `false`, the remaining literal has to be `true` in order to retain a chance to satisfy the formula. Such a clause implying a literal is called an *implication clause*. Moreover, today's SAT solvers add conflict clauses to the formula to prune the search space even further – so-called *conflict-driven clause learning* (CDCL) [SS96]. To allow backtracking and to ease the creation of a conflict clause an *implication graph* is used. It stores all currently assigned literals and keeps a link to the implication clause in case a literal was implied.

For the creation of conflict clauses *resolution* [Rob65] is used. To resolve two clauses C_1 and C_2 regarding a variable v both clauses have to contain v – but in opposite polarities. For example the clauses $C_1 = (a \vee \neg b \vee c)$ and $C_2 = (\neg d \vee e \vee \neg c)$ could be resolved regarding c yielding the resolvend $R = (a \vee \neg b \vee \neg d \vee e)$. The resolvend is always satisfied if the two originating clauses are satisfied. When adding a further conjunct to a CNF one may only exclude solutions but may not add new ones, therefore $C_1 \wedge C_2$ is equivalent to $C_1 \wedge C_2 \wedge R$.

SAT Modulo Theory (SMT) lifts the CDCL working principle to a higher level. In SMT every literal may represent a theory atom, e.g. $(x + y < 10)$. The SAT solver now works on the boolean abstraction of the underlying problem and assigns `true` or `false` to the theory atoms. If the SAT solver finds a satisfying assignment for the boolean abstraction, a theory solver is used to check if the conjunction of theory atoms satisfying the clauses is indeed satisfiable. If this is not the case the boolean abstraction is refined with a conflict clause which forbids the conflicting theory atoms. This scheme is also abbreviated as DPLL(T) or CDCL(T) – with T being the theory used.

2.2. The iSAT algorithm, iSAT3

The iSAT algorithm [FHT+07, Her11] uses *interval constraint propagation* (ICP, see e.g. [BG06]) to check the consistency of theory atoms – but there is no straight separation between the theory and SAT solver part. Instead ICP is tightly integrated into the CDCL framework. This deep integration has the advantage of sharing the common core of the search algorithms between the propositional and the theory-related part of the solver. So strictly speaking the iSAT algorithm goes beyond CDCL(ICP).

Theory atoms in the iSAT algorithm may contain linear and non-linear arithmetic involving transcendental functions. Examples for theory atoms are: $(x^2 + y^2 = z^2)$, $(|v - w| < \min(v, w))$ or $(\sqrt[3]{x} + \sin y < e^z)$. The iSAT algorithm supports boolean, integer- and real-valued variables Additionally every variable is bounded.

Before solving a given formula each theory atom is decomposed by a Tseitin-like transformation into *simple bounds* and unit clauses containing *primitive constraints*. During this process fresh auxiliary variables are introduced. A simple bound imposes a lower or an upper bound to a variable. Such a bound could be strict, e.g. $(x < 4)$ or non-strict, e.g. $(x \leq 4)$. Since each bound is represented as a floating point number, we use outward rounding (this means we round down for lower bounds and round up for upper bounds), to get a safe interval enclosure. Having a fixed set of primitive constraints makes it easier to apply ICP later on. A primitive constraint contains besides an unary or binary operator up to three associated variables. The following primitive constraints are supported:

unary primitive constraints	$h = -x$	
	$h = \|x\|$	
	$h = \sin x$	
	$h = \cos x$	
	$h = \exp x$	
	$h = x^n$	($n \geq 0$ integer constant)
	$h = \sqrt[n]{x}$	($n \geq 1$ integer constant)
binary primitive constraints	$h = x + y$	
	$h = x - y$	
	$h = x \cdot y$	
	$h = \min(x, y)$	
	$h = \max(x, y)$	

During the search process a so-called contractor is used to narrow the intervals occuring in each primitive constraint. Let's illustrate this with a small example. Assume the primitive constraint $(x = y + z)$ is given together with the following intervals for the contained variables:

$x \in [1,9], y \in [1,3]$ and $z \in [4,10]$. To contract the interval of x, the intervals of y and z are added. This yields the interval $[5,13]$, which is then intersected with the original interval of x and results in $[5,9]$ as the new interval for x. To contract the interval of y the primitive constraint is redirected to $y = x - z$. The resulting interval does not provide a stronger lower or upper bound, so the interval for y stays unchanged. After redirecting the primitive constraint to $z = x - y$ the interval for z is contracted to $[4,8]$. Contractors for other primitive constraints work in a similar way.

The three basic elements of the CDCL framework – (1) propagate, (2) resolve conflicts, (3) decide – are also present in the iSAT algorithm, but are extended for the operation on integer- and real-valued intervals in addition to boolean variables. Deciding an integer- or real-valued variable corresponds to splitting its interval and selecting the lower or upper half. Furthermore, in the propagation phase ICP is executed in addition to BCP. During ICP it is checked if every primitive constraint is still consistent with the current interval valuation of the variables. An interval valuation $\rho : \mathit{Var} \to \mathbb{I}_{\mathbb{R}}$ is a mapping from a set of variables *Var* to a set of intervals $\mathbb{I}_{\mathbb{R}}$. A unit clause containing a primitive constraint c is *inconsistent* under an interval valuation ρ, iff no values in the intervals $\rho(x)$ of the variables x in c can satisfy c, i.e.

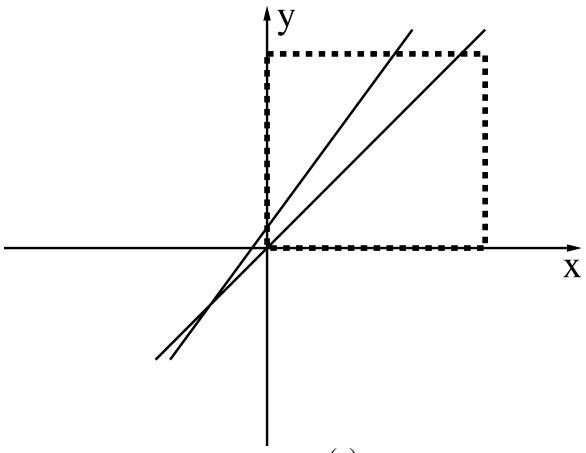
$$
\begin{array}{lcll}
\neg\exists v \in \rho(x) & : & v \sim r \quad \text{if } c = (x \sim r), & (r \in \mathbb{Q}) \\
\neg\exists v \in \rho(x), \neg\exists v' \in \rho(\circ y) & : & v \sim v' \quad \text{if } c = (x = \circ y), & (\circ \in \{-, \mathrm{abs}, \sin, \cos, \exp, \sqrt[n]{\cdot}, \cdot^n\}) \\
\neg\exists v \in \rho(x), \neg\exists v' \in \rho(y \circ z) & : & v \sim v' \quad \text{if } c = (x = y \circ z) & (\circ \in \{+, -, *\})
\end{array}
$$

where $\sim \in \{<, \leq, \geq, >\}$. Otherwise c is *consistent* under ρ.

If the primitive constraint is consistent and a new bound was deduced it is checked if the newly deduced bound has a negligible progress compared to the existing bound. If this is the case the new bound is ignored to prevent infinite propagation sequences and thus to guarantee termination. Furthermore, the iSAT algorithm may terminate with an inconclusive answer, because in general equations like $x = y \cdot z$ can only be satisfied by point intervals. However, reaching such point intervals by ICP cannot be guaranteed for real-valued variables. In such cases iSAT will return a so called *candidate solution*.

As mentioned, the iSAT algorithm uses interval splitting to decide integer- and real-valued variables. To enforce termination of the algorithm, interval splits are only performed if the considered interval is larger the so-called *minimal splitting width*. Additionally, every newly deduced (non-conflicting) bound has to be larger or equal to the *minimum progress* – otherwise the deduced bound is discarded. These two parameters may be specified by the user.

In this paper we focus on the third implementation of the iSAT algorithm – which we name iSAT3 [SKB13]. All three implementations (with HySAT [HEFT08] and iSAT [EKKT08] being the first two) share the same major core principles of tightly integrating ICP into the CDCL framework. However, while the core solvers of HySAT and iSAT operate on simple bounds, the core of iSAT3 uses literals and additionally utilizes an *abstract syntax graph* (ASG) for more advanced formula preprocessing. At first glance it looks like a minor design decision to choose between simple bounds and literals – because every simple bound can be seen as a literal and every literal can be expressed as a simple bound – but as the results in [SKB13] show it is beneficial to decide for literals, because it makes it easier to apply techniques known from propositional SAT solving like conflict clause minimization [SB09], assumptions [ES03] and further techniques [PD07, CHS09].

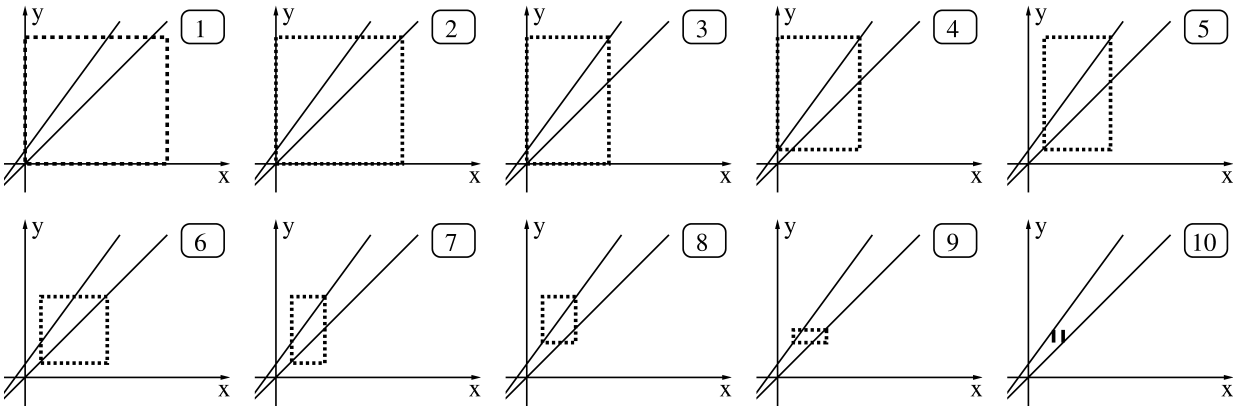


(a)

```
DECL
float [0,1000000] x,y;

EXPR
y = 2.00001*x + 0.25;
y = 2*x;
```

(b)



(c)

Figure 1: A small benchmark with two linear theory atoms listed in (b). There is no intersection within the given initial intervals $x, y \in [0, 1000000]$ as illustrated in (a). ICP will continuously shrink the intervals (like shown in (c) 1-9) until it finally deduces contradicting bounds for a variable in (c) 10. The pictures in (a) and (c) are only a rough illustration of what is happening, because for this example ICP needs in fact millions of deductions until the conflict is discovered.

3. Motivating example

We illustrate the inner workings of iSAT3 with the help of a small example shown in Figure 1(b). To keep it simple, we omitted any boolean structure or further theory atoms. Within the benchmark the variables x and y are declared – both within the initial interval $[0, 1000000]$. Furthermore, two linear theory atoms are specified. Every theory atom describes a line in the $\mathbb{R}^2$ space. These two lines are very close, but they have no intersection inside the initial intervals as illustrated in Figure 1(a). So this benchmark is unsatisfiable. With gaussian elimination this could be easily determined. ICP is also able to detect the unsatisfiability, but it needs a long deduction sequence to do so – exactly this is the purpose of this artificial example. Now one could get the impression that ICP is a weak procedure if it needs that many deductions. However, the beauty of ICP lies in its simple elegance and the ability to handle linear, non-linear and transcendental functions all within the same framework – while gaussian elimination is tailored for handling only quadratic systems of linear equations.

1	$(y$	$=$	$2.00001 * x + 0.25)$			$\wedge$	$(y$	$=$	$2 * x)$		
2	$(0$	$=$	$x + (-\frac{100000}{200001} * y) + (\frac{100000}{200001} * 0.25))$			$\wedge$	$(0$	$=$	$x + (-0.5 * y))$		
3	$(-\frac{25000}{200001}$	$=$	$x + (-\frac{100000}{200001} * y))$			$\wedge$	$(0$	$=$	$x + (-0.5 * y))$		
4a	$(t2 = x + t1) \wedge (t1 = c1 * y) \wedge$						$(t4 = x + t3) \wedge (t3 = c2 * y) \wedge$				
4b	$(c1 \geq \lfloor -\frac{100000}{200001} \rfloor) \wedge (c1 \leq \lceil -\frac{100000}{200001} \rceil) \wedge$					$\bigwedge$	$(c2 \geq -0.5) \wedge (c2 \leq -0.5) \wedge$				
4c	$(t2 \geq \lfloor -\frac{25000}{200001} \rfloor) \wedge (t2 \leq \lceil -\frac{25000}{200001} \rceil)$						$(t4 \geq 0) \wedge (t4 \leq 0)$				
5a	$(t2 = x + t1) \wedge (t1 = c1 * y) \wedge (t4 = x + t3) \wedge (t3 = c2 * y) \wedge$										
5b	$(c1 \geq \lfloor -\frac{100000}{200001} \rfloor) \wedge (c1 \leq \lceil -\frac{100000}{200001} \rceil) \wedge (c2 \geq -0.5) \wedge (c2 \leq -0.5) \wedge$										
5c	$(\neg b1 \vee (t2 \geq \lfloor -\frac{25000}{200001} \rfloor)) \wedge (\neg b1 \vee (t2 \leq \lceil -\frac{25000}{200001} \rceil)) \wedge (b1 \vee \neg(t2 \geq \lfloor -\frac{25000}{200001} \rfloor) \vee \neg(t2 \leq \lceil -\frac{25000}{200001} \rceil)) \wedge$										
5d	$(\neg b2 \vee (t4 \geq 0)) \wedge (\neg b2 \vee (t4 \leq 0)) \wedge (b2 \vee \neg(t4 \geq 0) \vee \neg(t4 \leq 0)) \wedge$										
5e	$(\neg b3 \vee b1) \wedge (\neg b3 \vee b2) \wedge (b3 \vee \neg b1 \vee \neg b2) \wedge (b3)$										

Figure 2: The translation of the original theory atoms to CNF.

										deduced	current
1	$t4_{lb}$	$=$	$x_{lb} + t3_{lb}$	$=$	0	$+$	-500000	$=$	-500000	-500000	0
2	$t4_{ub}$	$=$	$x_{ub} + t3_{ub}$	$=$	1000000	$+$	0	$=$	1000000	1000000	500000
3	x_{lb}	$=$	$t4_{lb} - t3_{ub}$	$=$	-500000	$-$	0	$=$	-500000	-500000	0
4	x_{ub}	$=$	$t4_{ub} - t3_{lb}$	$=$	0	$-$	-500000	$=$	500000	1000000	500000
5	$t3_{lb}$	$=$	$t4_{lb} - x_{ub}$	$=$	-500000	$-$	0	$=$	-500000	-500000	-500000
6	$t3_{ub}$	$=$	$t4_{ub} - x_{lb}$	$=$	1000000	$-$	1000000	$=$	0	0	0

Figure 3: ICP for the primitive constraint $(t4 = x + t3)$. Here ICP was able to deduce a new stronger upper bound for x in line 4.

Before solving a benchmark, iSAT3 normalizes it and converts it into a CNF. With the help of the auxiliary variables $t1, t2, t3, t4, c1, c2, b1, b2$ and $b3$, the theory atoms are decomposed into primitive constraints and simple bounds. Figure 2 summarizes what happens during formula preprocessing and Tseitin-transformation. Lines 1-3 show how the theory atoms are rewritten. Lines 5a-5e list the complete CNF – but for sake of simplicity we will use the CNF shown in lines 4a-4c, because unit propagation will directly assign $b1 = b2 = b3 =$`true`. The conjuncts with a darker gray background are primitive constraints, the ones with a lighter gray background are simple bounds. A simple bound containing a number like $\lfloor 0.1 \rfloor$ represents the largest floating pointing number smaller or equal to 0.1 – analogously does $\lceil 0.1 \rceil$ stand for the smallest floating point number larger or equal to 0.1 As a last step we add the initial bounds as unit clauses to the formula – if there is no bound information contained already for the associated variable. This results in the following formula:

$$(t2 = x + t1) \wedge (t1 = c1 * y) \wedge (t4 = x + t3) \wedge (t3 = c2 * y) \wedge (c1 \geq \lfloor -\frac{100000}{200001} \rfloor) \wedge (c1 \leq \lceil -\frac{100000}{200001} \rceil) \wedge$$
$$(c2 \geq -0.5) \wedge (c2 \leq -0.5) \wedge (t2 \geq \lfloor -\frac{25000}{200001} \rfloor) \wedge (t2 \leq \lceil -\frac{25000}{200001} \rceil) \wedge (t4 \geq 0) \wedge (t4 \leq 0) \wedge (x \geq 0) \wedge (x \leq 1000000) \wedge$$
$$(y \geq 0) \wedge (y \leq 1000000) \wedge (t1 \geq \lfloor -1000000 \frac{100000}{200001} \rfloor) \wedge (t1 \leq 0) \wedge (t3 \geq -500000) \wedge (t3 \leq 0)$$

Assume this formula is now passed to the solver core of iSAT3. Since every simple bound is interpreted as a literal, BCP will process all unit clauses containing the initial bounds and marks every variable for which a new bound was set. Then ICP is executed and looks at all primitive constraints containing a variable marked during the BCP stage. Regarding our example assume x

was marked first and ICP looks at the primitive constraint $(t4 = x + t3)$. Based on the current lower and upper bounds of $t4, x$ and $t3$ ICP will now try to deduce stronger lower and upper bounds for all three variables. As Figure 3 shows, only the value for x_{ub} deduced in line 4 is an improvement compared to the current x_{ub}. A new literal is created to represent this simple bound. Every assignment to a literal results in a new entry in the implication graph. While a decision has no associated implication clause, each deduced literal needs to have a clause implying it. Regarding x_{ub}, the reasons for this deduction are the values of $t4_{ub}$ and $t3_{lb}$. This results in the implication shown in Figure 4 line 1 – lines 2-4 show how it is possible to reformulate this implication as a clause. Later on, such clauses will contain the newly deduced bound as first literal as can be seen in Figure 5. ICP then continues to deduce new values for $x, y, t1$ and $t3$. Figure 1(c) illustrates this. From a geometrical point of view ICP constructs wrapping boxes around each theory atom and calculates the intersection of those boxes. This is done as long as new non-conflicting bounds can be deduced. Figure 5 shows a short excerpt of the deductions made by ICP in iSAT3 for this example. As the numbers in Figure 5 suggest, the intervals of x and y are slowly shrinking. In fact ICP will need millions of deductions to finally discover the conflict. With millions of entries one can imagine that the implication graph will occupy a huge amount of memory. This problem will now be addressed by the method we present in the next section.

1	$((t4 \leq 0)$	$\wedge$	$(t3 \geq -500000))$	$\Rightarrow$	$(x \leq 500000)$
2	$\neg((t4 \leq 0)$	$\wedge$	$(t3 \geq -500000))$	$\vee$	$(x \leq 500000)$
3	$\neg(t4 \leq 0)$	$\vee$	$\neg(t3 \geq -500000)$	$\vee$	$(x \leq 500000)$
4	$(t4 > 0)$	$\vee$	$(t3 < -500000)$	$\vee$	$(x \leq 500000)$

Figure 4: Creation of the implication clause for the deduced new upper bound for x.

4. Implication Graph Compression

As Figure 5 shows newly deduced bounds (e.g. the new upper bound for x deduced in line 2) themselves occur as reasons later on (for example in line 4). This is due to the fact that for all ICP operations the current bounds are used – resulting in the associated literals to occur in the implication clauses. The basic idea to compress the implication graph is to throw away all *intermediate bounds* and to adapt the implication clauses of the remaining bounds accordingly. In this context an intermediate bound is a bound which is only needed to deduce further stronger bounds. Regarding our motivating example almost all bounds are intermediate bounds. In fact for this example it would be enough to create an implication clause containing the initial bounds as reasons and implying a contradicting bound for one variable.

The question is now, how the implication clauses should be adapted. For a given implication clause we need to replace all contained intermediate bounds with their own reasons – as long as all reasons do not contain intermediate bounds themselves. Strictly speaking this is exactly what resolution does. Regarding Figure 5 line 4, we would resolve with the clause in line 2 regarding the literal $(x \leq 499997.37501312\ldots)$ and its negation $(x > 499997.37501312\ldots)$, resulting in the new implication clause:

$$(t3 \geq -499997.37501312\ldots) \vee (t4 < 0) \vee (t2 > -0.12499937\ldots) \vee (t1 < -499997.50001249\ldots)$$

This clause states that $(t3 \geq -499997.37501312\ldots)$ can be deduced because of bounds which are no intermediate bounds. As there is no other clause using $(x \leq 499997.37501312\ldots)$ as a reason,

1	$(x \leq 500000)$	$\vee$	$(t4 > 0)$	$\vee$	$(t3 < -500000)$
2	$(x \leq 499997.37501312\ldots)$	$\vee$	$(t2 > -0.12499937\ldots)$	$\vee$	$(t1 < -499997.50001249\ldots)$
3	$(t1 \leq -0.12499937\ldots)$	$\vee$	$(t2 > -0.12499937\ldots)$	$\vee$	$(x < 0)$
4	$(t3 \geq -499997.37501312\ldots)$	$\vee$	$(t4 < 0)$	$\vee$	$(x > 499997.37501312\ldots)$
5	$(y \geq 0.24999999\ldots)$	$\vee$	$(t1 > -0.12499937\ldots)$		
6	$(t3 \leq -0.12499999\ldots)$	$\vee$	$(y < 0.24999999\ldots)$		
7	$(y \leq 999994.75002625\ldots)$	$\vee$	$(t3 < -499997.37501312\ldots)$		
8	$(x \geq 0.12499999\ldots)$	$\vee$	$(t4 < 0)$	$\vee$	$(t3 > -0.12499999\ldots)$
9	$(t1 \geq -499994.87503874\ldots)$	$\vee$	$(y > 999994.75002625\ldots)$		
10	$(x \leq 499994.75003937\ldots)$	$\vee$	$(t2 > -0.12499937\ldots)$	$\vee$	$(t1 < -499994.87503874\ldots)$
11	$(t1 \leq -0.24999937\ldots)$	$\vee$	$(t2 > -0.12499937\ldots)$	$\vee$	$(x < 0.12499999\ldots)$
12	$(y \geq 0.50000124\ldots)$	$\vee$	$(t1 > -0.24999937\ldots)$		
13	$(t3 \geq -499994.75003937\ldots)$	$\vee$	$(t4 < 0)$	$\vee$	$(x > 499994.75003937\ldots)$
14	$(t3 \leq -0.25000062\ldots)$	$\vee$	$(y < 0.50000124\ldots)$		
15	$(y \leq 999989.50007874\ldots)$	$\vee$	$(t3 < -499994.75003937\ldots)$		
16	$(t1 \geq -499992.25007812\ldots)$	$\vee$	$(y > 999989.50007874\ldots)$		
17	$(x \geq 0.25000062\ldots)$	$\vee$	$(t4 < 0)$	$\vee$	$(t3 > -0.25000062\ldots)$
18	$(x \leq 499992.12507874\ldots)$	$\vee$	$(t2 > -0.12499937\ldots)$	$\vee$	$(t1 < -499992.25007812\ldots)$
19	$(t1 \leq -0.37500000\ldots)$	$\vee$	$(t2 > -0.12499937\ldots)$	$\vee$	$(x < 0.25000062\ldots)$
20	$(t3 \geq -499992.12507874\ldots)$	$\vee$	$(t4 < 0)$	$\vee$	$(x > 499992.12507874\ldots)$
...			...		

Figure 5: An excerpt of the deductions done by ICP regarding the benchmark from Figure 1(b). Floating point numbers with ... are truncated for better readability

the bound itself is now obsolete and can be removed from the implication graph. Of course the new bound for $t3$ occurs itself as a reason for another implication (Figure 5 line 7). Here again, we do resolution and resolve regarding $(t3 \geq -499997.37501312\ldots)$ and its negation. We get an implication clause with no intermediate bounds and because $(t3 \geq -499997.37501312\ldots)$ is not used in another place we can remove it.

So the idea is to do resolution between the implication clauses – but instead of doing this as a post-processing step after the implication graph already exists, we want to do this on-the-fly and therefore prevent the growth of implication graph right from the start. To do so, we delay the addition of newly deduced bounds to the implication graph – instead we store them temporarily together with their implication clauses. The pseudocode of the algorithm is shown in Figure 6. In line 2 ICP is executed and returns an implication clause. In this clause the first literal is always the implied literal. A new clause is created in line 3 and used in lines 4-15 to resolve intermediate bounds. In line 7 the associated variable of a simple bound is determined, e.g. x for the simple bound $(x \leq 500000)$. In line 9 it is checked if a variable has already a temporary implication clause for its lower or upper bound. If yes, we can do resolution between this temporary implication clause and the current implication clause in line 11. Here we exploit that the implied literal is always stored in the first position of an implication clause. Then in line 20 and 21 we store the new bound together with its implication clause. These values are written to the implication graph in line 25 and 28 if (1) BCP is able to deduce implied literals inbetween, (2) or no further deductions are possible in the current decision level.

If the sheer amount of the intermediate bounds is the problem, then it could seem tempting to favor CDCL(ICP) over the tight integration of ICP into CDCL – because in such a scenario the implication graph would never contain intermediate bounds. On the other hand if we would move ICP into a separate theory solver which is only used for consistency checks of the theory atoms

```
1  on_the_fly_compression() {
2      clause = do_icp();
3      c = new clause;
4      append(c, clause[0]);
5      for (i = 1; i < length(clause); i++) {
6          literal = clause[i];
7          variable = get_associated_variable(literal);
8          is_lb = is_lower_bound(literal);
9          if (tmp_clauses[variable, !is_lb] != NULL) {
10             t = tmp_clauses[variable, !is_lb];
11             for (j = 1; j < length(t); j++) append(c, t[j]);
12         } else {
13             append(c, clause[i]);
14         }
15     }
16     clause = c;
17     variable = get_associated_variable(clause[0]);
18     is_lb = is_lower_bound(clause[0]);
19     variables[is_lb] = variables[is_lb] ∪ variable
20     tmp_bounds[variable, is_lb] = clause[0];
21     tmp_clauses[variable, is_lb] = clause;
22 }
23 update_impl_graph() {
24     foreach (v in variables[0]) {
25         append_to_impl_graph(tmp_bounds[v, 0], tmp_clauses[v, 0]);
26     }
27     foreach (v in variables[1]) {
28         append_to_impl_graph(tmp_bounds[v, 1], tmp_clauses[v, 1]);
29     }
29     clear(variables, tmp_bounds, tmp_clauses);
31 }
```

Figure 6: Pseudocode of the two core functions needed for implication graph compression. The function `on_the_fly_compression()` resolves the implication clauses on-the-fly and stores the current bounds. If those bounds and the implication clauses have to be stored in the implication graph the function `update_impl_graph()` will be used.

satisfying a clause, we would lose the ability to trigger boolean propagations as early as possible. With our approach we get both, (1) we are still able to quickly alternate between ICP and BCP and (2) we have a compact implication graph.

If we now apply the algorithm of Figure 6 to the benchmark from Figure 1(b) the overall memory footprint drops from 620 MB down to 6.5 MB. This is not surprising, because the benchmark was designed to trigger very long ICP sequences. Therefore, the next section will answer the more interesting question, how the approach will perform on a set of standard benchmarks.

5. Experimental Results

Table 1 shows the experimental results over a set of standard benchmarks for the iSAT algorithm. We selected those benchmarks with a considerable amount of ICP to measure the effectiveness of our method. In the first column the names of the benchmarks are listed. Columns 2 and 3 contain

Benchmark	without IGC		with IGC		Change	
	Runtime (s)	Memory (MB)	Runtime (s)	Memory (MB)	Runtime	Memory
minigolf	15.05	526.77	7.43	59.02	-50.63%	-88.80%
train_system1	82.18	1063.75	117.86	486.09	+43.42%	-54.30%
train_system2	15.28	967.59	10.46	66.45	-31.54%	-93.13%
train_system3	825.46	7824.00	818.55	1489.50	-0.84%	-80.96%
train_system3*	-	MEMOUT	2539.81	1858.54	-	-
train_distance_ctrl1	8.11	578.41	10.79	89.62	+33.05%	-84.51%
train_distance_ctrl2	13.83	486.88	21.16	115.16	+53.00%	-76.35%
train_distance_ctrl3	15.81	521.56	25.82	198.25	+63.31%	-61.99%
etcs_train_system	200.08	2364.00	194.75	361.73	-2.66%	-84.70%
renault_clio	1.54	154.52	2.44	208.34	+58.44%	+34.83%
collision_avoidance1	73.69	670.46	43.61	16.73	-40.82%	-97.50%
collision_avoidance2	51.51	1004.32	37.01	16.32	-28.15%	-98.38%
hysat_6_bus	407.15	696.92	157.08	247.80	-61.42%	-64.44%
bouncing_ball_euler	2.36	72.58	3.40	59.97	+44.07%	-17.37%
arbiter	58.38	105.62	105.79	169.95	+81.21%	+60.91%
$\sum$	1770.43	17037.38	1556.15	3584.93	-12.10%	-78.96%

Table 1: Comparison between iSAT3 with and without implication graph compression. The runtime (User time) and memory consumption (Maximum resident set size) was measured with `time -v`.

the runtime and memory consumption for iSAT3 without implication graph compression, while Columns 4 and 5 show the values for iSAT3 with implication graph compression. In Columns 6 and 7 the change of runtime and memory consumption is recorded.

The results show that our method is able to reduce the memory consumption considerably – for some benchmarks even by more than an order of magnitude. As expected the runtime keeps roughly unchanged summed up over all benchmarks. Some benchmarks are solved faster, while others need more runtime. This is due to the fact that the decision heuristics is influenced indirectly. The solver core of iSAT3 uses the Variable State Independent Decaying Sum (VSIDS) heuristics when a decision has to be made. VSIDS calculates a so-called activity for every variable. This activity depends on how often a variable is seen during the conflict analysis – due to the compression of the implication graph this is likely to be different. This leads to other activity values and may therefore guide the solver to traverse the search space in a different way – leading to changed runtimes for individual benchmarks.

As stated in Section 2 the user may specify the two parameters minimal splitting width and minimum progress to influence the solving process. Usually smaller values reduce the number of candidate solutions and increase the number of conclusive answers, but at the cost of more deductions and decisions to be made – which is likely to increase runtime and the size of the implication graph. If we compress the implication graph in such situations, this may allow completing the solving process instead of hitting a MEMOUT inbetween. The benchmark train_system3 is listed twice in Table 1 and is an example for such a case. In the second run marked with $\star$ we used a minimal splitting width of 0.0001 and a minimum progress of 0.00001 – this is a hundredth of the values used for all other measurements. The benchmark is a bounded model checking problem. With a minimum splitting width of 0.01 and a minimum progress of 0.001 the unsatisfiability of depth 64 could not be proven, while this was possible with the smaller values for those two parameters. Without implication graph compression the solver was unable to complete the benchmark and ran into a MEMOUT, while with implication graph compression the solver was able to finish.

6. Conclusion

We presented a method to compress the implication graph of iSAT3. The method exploits the fact that ICP may produce long sequences of slowly shrinking intervals leading to a huge implication graph occupying a large amount of memory. Our approach addresses this problem by doing on-the-fly resolution between implication clauses. This considerably reduces the overall memory consumption – in some cases even more than an order of magnitude and may therefore allow to complete the solving process instead of hitting a MEMOUT inbetween.

References

[BG06] F. Benhamou and L. Granvilliers. Continuous and Interval Constraints. In *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, pages 571–603. 2006.

[CHS09] Geoffrey Chu, Aaron Harwood, and Peter J. Stuckey. Cache Conscious Data Structures for Boolean Satisfiability Solvers. *JSAT*, 6(1-3):99–120, 2009.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.

[EKKT08] Andreas Eggers, Natalia Kalinnik, Stefan Kupferschmid, and Tino Teige. Challenges in Constraint-Based Analysis of Hybrid Systems. In Angelo Oddi, François Fages, and Francesca Rossi, editors, *CSCLP*, volume 5655 of *Lecture Notes in Computer Science*, pages 51–65. Springer, 2008.

[ES03] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[FHT+07] Martin Fränzle, Christian Herde, Tino Teige, S. Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling, and Computation*, 1(3-4):209–236, 2007.

[HEFT08] Christian Herde, Andreas Eggers, Martin Fränzle, and Tino Teige. Analysis of Hybrid Systems Using HySAT. In *ICONS*, pages 196–201. IEEE Computer Society, 2008.

[Her11] Christian Herde. *Efficient solving of large arithmetic constraint systems with complex Boolean structure: proof engines for the analysis of hybrid discrete-continuous systems*. PhD thesis, 2011.

[PD07] Knot Pipatsrisawat and Adnan Darwiche. A Lightweight Component Caching Scheme for Satisfiability Solvers. In João Marques-Silva and Karem A. Sakallah, editors, *SAT*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer, 2007.

[Rob65] John Alan Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *J. ACM*, 12(1):23–41, 1965.

[SB09] Niklas Sörensson and Armin Biere. Minimizing Learned Clauses. In Oliver Kullmann, editor, *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 237–243. Springer, 2009.

[SKB13] Karsten Scheibler, Stefan Kupferschmid, and Bernd Becker. Recent Improvements in the SMT Solver iSAT. In Christian Haubelt and Dirk Timmermann, editors, *MBMV*, pages 231–241. Institut für Angewandte Mikroelektronik und Datentechnik, Fakultät für Informatik und Elektrotechnik, Universität Rostock, 2013.

[SS96] João P. Marques Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In *ICCAD*, pages 220–227, 1996.

[Tse68] Grigori S. Tseitin. On the complexity of derivations in propositional calculus. In A. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logics*. 1968.