Integration of an LP Solver into Interval Constraint Propagation

Ernst Althaus\textsuperscript{1,2}, Bernd Becker\textsuperscript{3}, Daniel Dumitriu\textsuperscript{1}, and Stefan Kupferschmid\textsuperscript{3}

\textsuperscript{1} Johannes Gutenberg University, Mainz, Germany
\{ernst.althaus,dumitriu\}@uni-mainz.de
\textsuperscript{2} Max Planck Institute for Computer Science, Saarbrücken, Germany
\textsuperscript{3} Albert Ludwig University, Freiburg, Germany
\{becker,skupfers\}@informatik.uni-freiburg.de

\textbf{Abstract.} This paper describes the integration of an LP solver into iSAT, a Satisfiability Modulo Theories solver that can solve Boolean combinations of linear and nonlinear constraints. iSAT is a tight integration of the well-known DPLL algorithm and interval constraint propagation allowing it to reason about linear and nonlinear constraints. As interval arithmetic is known to be less efficient on solving linear programs, we will demonstrate how the integration of an LP solver can improve the overall solving performance of iSAT.

1 Introduction

We are considering the sat modulo theory (SMT) problem, which reads as follows. We are given a set of variables \( \{x_1, \ldots, x_m\} \) and a set \( C = \{c_1, \ldots, c_n\} \) of constraints of some class of constraints over these variables, e.g. linear inequalities. Furthermore, we are given a Boolean structure over \( C \), i.e. a Boolean formula \( \phi \), whose variables are the elements of \( C \). The task is to decide the feasibility of \( \phi \), i.e. a Boolean assignment \( b : C \mapsto \{\text{true}, \text{false}\} \) to the constraints, such that \( \phi \) is satisfied for \( b \) and such that there is \( x \in \mathbb{R}^m \) satisfying the constraints \( \{c_i \mid b(c_i) = \text{true}\} \) and \( \{\neg c_i \mid b(c_i) = \text{false}\} \). Hence we assume that the class of constraints is closed under negation.

SMT is an extension of the classical satisfiability problem. Such formulae arise in many applications dealing with the verification of Hybrid Systems [12].

The problem can be decided, if the feasibility problem of a conjunction of constraints of the class of constraints can be solved. The most prominent algorithm is the DPLL algorithm [6] in which the Boolean values for the constraints are determined via a search with backtracking, like in ordinary SAT algorithms. Furthermore, for each partial assignment of Boolean values, the corresponding set of constraints is checked for feasibility. There are many tricks to improve

\textsuperscript{*} This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, www.avacs.org) and as part of its priority program "SPP 1307: Algorithm Engineering", grant AL 1139/1-2.
the running time, like the so-called *unit propagation* – if all but one literal of a clause are assigned to false, the remaining literal has to be true. In particular for the class of linear inequalities, there are many implementations to tackle the problem; we mention here only Yices [8], MathSAT [4], and OpenSMT [5].

We are interested in more complicated classes of constraints, i.e., we allow for arbitrary equalities and inequalities (strict and non-strict) composed from addition, multiplication, exponentiation, and (co-)sine. Hence, the problem under consideration becomes undecidable in general\(^4\).

This large class of constraints allows us to model Hybrid Systems in a more precise way as lots of physical systems contain a quadratic, or exponential behavior that we want to analyze.

As we are dealing with transcendental functions the problem is in general undecidable, therefore we do not aim to solve it, but we either report the infeasibility or report a point \(x \in \mathbb{R}^m\) such that all equalities and inequalities are satisfied up to some predefined accuracy in the arithmetic. In particular, strict inequalities are sometimes only satisfied non-strict. In order to do so, we have to assume that all variables \(x_i\) and all subformulae\(^5\) have a bounded domain. We do so by an extension of the DPLL procedure in which we additionally allow the splitting of the domain of a variable. To become efficient, we use *interval constraint propagation* (ICP), i.e., given a constraint \(c_i\) with fixed assignment \(b(c_i)\) and the current domains of the variables, we try to narrow the domains without removing any point satisfying the constraint. For example, if \(x + y \leq 0\) is a constraint and \(x \in [0, 1], y \in [-2, 2]\) are the current domains, we can narrow the domain of \(y\) to \([-2, 0]\) as there is no point in the original domain that satisfies \(x + y \leq 0\) with \(y > 0\). We refer to [2] for details.

This algorithm does not require a specialized solver for a particular class of constraints as long as the narrowing of the domains can be done efficiently. Nevertheless, the work of Gao et al. [10] indicates that an additional check for feasibility for all linear constraints can improve upon the running time. They employ an LP solver to characterize the feasible region described by the linear constraints and try to hand over this information to an ICP solver (they use the predecessor of our solver). Our work differs in that the overall search process is guided by the ICP solver and not by an LP solver as implemented in [10]. As ICP is known to be less efficient when solving linear systems we perform additional LP solver calls to detect unsatisfiable search parts earlier, and thus prevent the solver from making unnecessary ICP calls. Furthermore, our results demonstrate that the combination of an ICP solver and an LP solver increases the number of problems that we can solve, especially on unsatisfiable problem instances.

We call a specialized LP solver once the domain propagation reached a fixed point. The solver first tries to decide the feasibility or infeasibility using a solution/Farkas proof of a previous LP; if this is not successful, there are some

\(^4\) It is well known that nonlinear arithmetic over the real numbers involving transcendental functions like \(\sin\) is undecidable [19].

\(^5\) We do not give a formal definition here and refer to [9] for details.
heuristics to decide whether the LP is solved or not. Once an LP has been shown to be infeasible, a small subset of the constraints causing the infeasibility based on the Farkas proof is given to the solver to speed up the search. This technique is called conflict learning.

The typical application of our algorithm is for model checking, in which the infeasibility of the formula means that the system is safe. If we report infeasible, this has to be safe and not corrupted due to rounding errors. Hence the domain narrowing is done in a conservative way, i.e., we always adopt the rounding mode such that the domain overapproximates the real domain. Using an LP solver, we have to make sure that we do not declare a feasible LP to be infeasible or report an infeasible subsystem that is feasible. In previous approaches, this was guaranteed using an LP solver based on rational arithmetic. In this paper, we show how a state-of-the-art floating point based LP solver can be used. We compare an approach that goes along the lines proposed by Dhiflaoui et al. [7] to an approach that goes along the lines proposed by Neumaier and Scherhina [15]. The later extension of this approach by Althaus and Dumitriu [1] is not required as all linear systems are bounded.

Notice that in this context, some linear systems are infeasible due to the strictness of some bounds; as a result, an arbitrary small change of the bounds can make an infeasible system feasible, hence rounding errors are very critical.

The paper is organized as follows. We first review the interval constraint programming approach of our underlying solver and then describe how we integrate an LP solver in Section 2. In Section 3, we show how a floating point based LP solver can be used to solve the linear programs including strict inequalities, thereby certifying the correctness of its result. Before giving a conclusion, we report on some experiments in Section 4.

2 Integration of an LP solver into iSAT

In this section we present our approach that combines iSAT, a DPLL based interval constraint solver, and an LP solver. In order to do this we first provide a short introduction to iSAT (for a more detailed account please refer to [9]).

2.1 Introducing iSAT

In the following let $\varphi$ be a Boolean combination of linear and nonlinear constraint formula. The front-end of iSAT computes normalized constraints and the Conjunctive Normal Form (CNF). After that we end up with a formula having the following syntax:

\[
\text{formula ::= \{clause \land\}^* \text{clause}} \\
\text{clause ::= \{\{atom \lor\}^* \text{atom}\}} \\
\text{atom ::= simple\_bound | arithmetic\_predicate} \\
\text{simple\_bound ::= variable \sim rational\_const} \\
\text{arithmetic\_predicate ::= variable \sim uop variable | variable \sim variable bop variable | variable \sim variable bop rational\_const}
\]
In the above syntax, \textit{uop} and \textit{bop} are unary and binary operation symbols respectively, including +, −, ×, \textit{sin}(\cdot), etc.. \textit{rational\_const} ranges over the rational constants, and \(\sim \in \{\leq, \geq\}\). To illustrate this phase, consider the following formula:

\[
(x \geq 0) \land (x \leq 10) \land ((\sin(1/3x) + \sqrt{x} \geq y) \implies (y \geq 1/4x + 3)) \quad (1)
\]

First we eliminate the Boolean operator \(\implies\) by applying a Tseitin transformation [20]. To do so, the implication will be replaced by a new auxiliary Boolean variable \((b)\). The remaining formula is then normalized by introducing additional real variables \(r_1, r_2\) and \(r_3\) and the following constraints \(r_1 = 1/3x, r_2 = \sin(r_1)\) and \(r_3 = \sqrt{x}\). Sometimes we call the additional introduced variables \((r_1, r_2, r_3)\) \textit{auxiliary variables}. Finally, the normalized CNF problem looks as follows:

\[
\begin{align*}
(x \geq 0) & \land (x \leq 10) \land (b \lor r_2 + r_3 < y \lor y \geq 3 + r_4) \\
(r_2 + r_3 & \geq y \lor b) \land (y \geq 3 + r_4 \lor b) \land \\
(r_1 & = 1/3x) \land (r_2 = \sin(r_1)) \land (r_3 = \sqrt{x}) \land (r_4 = 1/4x)
\end{align*}
\]

(2)

All clauses now have the syntax described above and can be transferred to the solver. Before describing the solving process in detail, we informally describe the underlying semantics. A constraint formula \(\varphi\) is satisfied by a valuation of its variables if all its clauses are satisfied, that is, if at least one atom is satisfied in any clause. An atom is satisfied w.r.t. the standard interpretation of the arithmetic operators and the ordering relations over the reals. A constraint formula \(\varphi\) is \textit{satisfiable} if there exists a satisfying valuation, referred to as a \textit{solution} of \(\varphi\). Otherwise, \(\varphi\) is \textit{unsatisfiable}. We remark that by definition of satisfiability, a formula \(\varphi\) including or implying the empty clause, denoted by \(\bot\), cannot be satisfied at all, i.e. if \(\bot \in \varphi\) or \(\varphi\) is unsatisfiable.

Instead of real-valued variable valuations, ISAT manipulates interval ranges. By using the function \(\rho : \text{Var} \to \mathbb{I}_\mathbb{R}\), where \(\text{Var}\) is a set of variables and \(\mathbb{I}_\mathbb{R}\) is the set of interval ranges of \(\mathbb{R}\), we define a range for each variable. Note, that we also support discrete variable domains (integer and Boolean). To this end, it suffices to clip the interval of integer variables accordingly, such that \([-3.4, 6.0)\) becomes \([-3, 5]\), for example. The Boolean domain is represented by \(\mathbb{B} = \{0, 1\} \subset \mathbb{Z}\). If both \(\rho'\) and \(\rho\) are interval valuations, then \(\rho'\) is called a \textit{refinement} of \(\rho\) if \(\rho'(v) \subseteq \rho(v)\) for each variable \(v \in \text{Var}\). The lower and upper interval borders of an interval \(\rho(x)\) for a variable \(x\) can be encoded as simple bounds. We denote the lower and upper interval border of the interval \(\rho(x)\) by \(\text{lower}(\rho(x))\) and \(\text{upper}(\rho(x))\), respectively. E.g., for the interval \(\rho(x) = (-4, 9]\) we have \(\text{lower}(\rho(x)) = (x > -4)\) and \(\text{upper}(\rho(x)) = (x \leq 9)\).

Let \(x\) and \(y\) be variables, \(\rho\) be an interval valuation, and \(\circ\) be a binary operation. Then \(\rho(x \circ y)\) denotes the \textit{interval hull} of \(\rho(x)\circ\rho(y)\) (i.e. the smallest enclosing interval which is representable by machine arithmetic), where the operator \(\circ\) corresponds to \(\circ\) but is canonically lifted to sets. This is done analogously for unary operators. In order to compute the interval hull \(\rho(x \circ y)\) we are using ICP. This allows us to narrow intervals of the variables, i.e. given a formula
We say that an atom $a$ is inconsistent under an interval valuation $\rho$, referred to as $\rho^\flat a$, if no values in the intervals $\rho(x)$ of the variables $x$ in $a$ satisfy the atom $a$, i.e.

\begin{align*}
-\exists v \in \rho(x) : v \sim c & \text{ if } a = (x \sim c), \\
-\exists v \in \rho(x), -\exists v' \in \rho(y) : v \sim v' & \text{ if } a = (x \sim oy), \\
-\exists v \in \rho(x), -\exists v' \in \rho(y \circ z) : v \sim v' & \text{ if } a = (x \sim y \circ z)
\end{align*}

where $\sim \in \{<,\leq,=,\geq,>\}$. Otherwise $a$ is consistent under $\rho$. For our purpose we do not need the definition of interval satisfaction. It is sufficient to talk about atoms which are still consistent. We remark that proving the satisfiability of an iSAT formula is not trivial. For more details we refer to [9, Subsection 4.5].

In Algorithm 1, the pseudocode of iSAT is given by ignoring the code between line 7 and line 11. Before the main iSAT routine starts, it is assumed that all the unit clause information contained in the original formula has already been propagated, which can sometimes allow us to derive tighter bounds. Once this is ensured, Algorithm 1 begins by making a decision, and splitting the interval range of a variable, e.g. splits a variable’s range in half (line 3). This decision will be propagated in line 4. If a conflict is detected (e.g. a clause evaluates to false during propagation) it will be analyzed in line 5. The conflict analysis routine uses the implication graph of the solver to compute the reasons for the conflict. By doing so a conflict clause is learned, allowing iSAT to prune off unsatisfiable parts of the search space. iSAT terminates in either line 5 or 13 with either unsat, sat, or unknown. If iSAT has managed it to split every problem variable up to a so called minimum splitting width (msw) and no conflict is detected the main DPLL loop is terminated and a satisfiability check for every problem clause is fueled in line 13.

2.2 Integration of an LP solver

The integration of an LP solver affects the normalization and the solving part of iSAT. In the normalization part iSAT detects every linear constraint that is contained in the input formula. To get a better picture we will give an example.

In equation 1 of the previous subsection there are three linear constraints ($x \geq 0$, $x \leq 10$, and $y \geq 1/4x + 3$). Every linear constraint that does not have the syntax of a simple bound will be given to the LP solver. In this case the only linear constraint that is not a simple bound is $y \geq 1/4x + 3$. Here the normalization routine would transform the linear constraint into $-1/4x + y - s = 0$. In other
Data: CNF F
Result: sat, unsat or unknown

/* Main DPLL loop. DecideVar returns 0 once the msw for */
/* all variables is reached, and no further decisions */
/* are possible. */

while decideVar() do
    /* Propagates current decision and unit constraints. */
    if propagateICP() = Conflict then
        /* Function tries to resolve the conflict by backtracking. */
        /* If conflict is unresolvable, problem is unsatisfiable. */
        if analyseBacktrack() = Unresolvable then return unsat;
    end
    /* ICP did not find a conflict. Try to find a conflict */
    else if checkLPFeasibility() = Infeasible then
        insertCert()
        if analyseBacktrack() = Unresolvable then
            return unsat;
        end
    end
end
/* Final test: Are all constraints satisfied? */
if allClausesSat() then return sat; else return unknown;

Algorithm 1: DPLL + ICP + LP

words we introduce for every linear constraint a so-called slack variable s. This way we produce an initially feasible LP tableau that can be transmitted to the LP solver; hence the normalization part produces the input for the iSAT solver and for the LP solver. The input for iSAT looks like:

\[
\begin{align*}
(x \geq 0) \land (x \leq 10) \land (b \lor r_2 + r_3 < y \lor y \geq r_4 + 3) \land \\
(r_2 + r_3 \geq y \lor b) \land (s \geq 3 \lor b) \land \\
(r_1 = 1/3x) \land (r_2 = \sin(r_1)) \land (r_3 = \sqrt{x}) \land \\
(r_4 = 1/4x) \land (s = y - r_4)
\end{align*}
\]  

(3)

And the input for the LP solver is:

\[-1/4x + y - s = 0\]  

(4)

Of course, iSAT is able to solve the formula given in equation 3 without the help of the LP solver. But as interval arithmetic is known to be less efficient when solving linear programs our aim is to improve the effectiveness of the solver by integrating an LP solver.

To do this we modify the iSAT procedure by adding additional feasibility checks for the linear constraints under the current interval valuation (line 7). These checks are performed after no conflict has been detected by the propagation phase (line 4). If the LP solver reports infeasible, a clause containing the
negations of those column bounds that are responsible for the infeasibility is inserted into ISAT’s learned clause database (line 8) thus preventing ISAT of entering this certain part of the search space again. To keep the number of literals small we compute these bounds from the Farkas’ Lemma that is explained in the next section.

We further implemented the option of either adding the linear constraints only to the LP solver and just the nonlinear constraints are added to ISAT or the linear constraints are added to both solvers.

3 Solving the Linear Programs and Computation of Small Infeasible Subsets

We start our description assuming real arithmetic of the computer and describe necessary changes due to the floating point arithmetic afterwards. Besson [3] proposes a similar approach to ours, in that he computes infeasible subsystems with floating point arithmetic and certifies a correct result by solving a system of linear equations with rational arithmetic. His approach to compute infeasible subsystems for a system of linear inequalities including strict ones seems to be more complicated than ours.

Given a system of linear inequalities with some strict bounds as outlined in the previous section, deciding the feasibility can be achieved simply by maximizing the minimal distance to one of the strict inequalities, i.e. by solving the following linear program:

\[
\begin{align*}
\max & \quad \delta \\
\text{s.t.} & \quad Ax = 0 \\
& \quad x + u^s \delta \leq u \\
& \quad x - l^s \delta \geq l \\
& \quad \delta \geq 0,
\end{align*}
\]

where we define \( l^s_i = 1 \) if the lower bound \( l_x \) on \( x_i \) is strict and \( l^s_i = 0 \) otherwise, \( u^s_i = 1 \) if the upper bound \( u_x \) on \( x_i \) is strict and \( u^s_i = 0 \) otherwise.

If the LP is infeasible or has an objective function value of 0, the system of linear inequalities is infeasible, otherwise the system is feasible. A small but infeasible set of inequalities can be obtained then either from the Farkas certificate or from the dual solution. As the linear equations are globally valid, we do not include them in the certificate, but only a set of bounds.

More precisely, if the LP is infeasible, it has no solution with \( \delta = 0 \). Basically this means that the system has no solution, even if all strict bounds would be non-strict. Hence, we can drop \( \delta \) and we know that the simplified linear system \( Ax = 0, l \leq x \leq u \) is infeasible. Hence its Farkas system (a linear system which is feasible if and only if the given system is infeasible) is feasible and reads as follows:

\[
\begin{align*}
p^T A + q^T r^T - r^T = & \quad 0 \\
q^T u - r^T l^s = & \quad -1 \\
q, r \geq & \quad 0
\end{align*}
\]
Notice that if we drop all variables with value 0, the linear system (I) is still feasible and hence the original LP without the corresponding constraints is still infeasible. Hence, the certificate consists of all lower bounds $\ell_i$ for which the corresponding Farkas system variable $r_i > 0$ and all upper bounds $u_i$ for which the corresponding $q_i > 0$. The Farkas certificate for the (non-simplified) LP can be obtained from all LP solvers after the original LP has been solved and can be directly used for the simplified LP.

If the LP is feasible with objective function value 0, its dual has a solution with objective function value 0, i.e. the following system of linear equations is feasible:

$$\begin{align*}
q^T u - r^T \ell &= 0 \\
p^T A + q^T - r^T &= 0 \\
q^T u^s + r^T \ell^s &= 1 \\
q, r &\geq 0
\end{align*}$$

Furthermore, if the system is feasible, the LP has objective function value 0 or is infeasible. Again the certificate consists of all lower bounds $\ell_i$ for which $r_i > 0$ and all upper bounds $u_i$ for which $q_i > 0$ and can be obtained from the LP solution.

Assume now that the LP is solved with a state-of-the-art floating point solver. We discuss three alternative methods to certify the infeasibility and the computed Farkas certificate; we implemented the latter two of them. In case any of these methods fails, it returns that the feasibility status of the LP is unknown.

The first method was proposed by Dhifaoui et al. [7]. They take the LP basis and try to verify its correctness using rational arithmetic. Basically, the basis gives a subset of the variables such that the system (I), respectively (II), has a solution with all non-basic variables having value 0, and the system of equations reduced to the basic variables has a unique solution which is non-negative. Given the basis, we can solve a system of linear equations and check that the solution is non-negative, in order to certify that the system (I), respectively (II), has a solution. This is done using rational arithmetic. The experiments of Dhifaoui et al. show that for some instances, the time to solve these systems of linear equations is much higher than the time to solve the LP. This observation has been confirmed by a more efficient implementation by Koch [14].

In the second method, we solve a smaller system of equations in order to become more efficient. More precisely, we select the variables with floating point value larger than some $\varepsilon > 0$ and solve the corresponding subsystem of equations. Notice that these variables are a subset of the basic variables; this subset is strict if the basis is degenerate, which is often the case in practical applications. In this case, we get a linear system which is overdetermined. In Section 4, we show that these systems are often significantly smaller and can be solved very efficiently.

The third method was proposed by Neumaier and Shcherbina [15] and can be applied as all variables have finite bounds. We discuss that case of the infeasible LP first. Given the floating point solution $(p, q, r)$, they compute $pA$
using interval arithmetic. From this, they compute intervals for \( q \) and \( r \) such that the system \( p^T A + q^T - r^T = 0 \) definitely has a solution by setting \( q_i = \max(0, -\text{upper}(pA)_i), \max(0, -\text{lower}(pA)_i) \) and \( r_i = \max(0, \text{lower}(pA)_i), \max(0, \text{upper}(pA)_i) \). Then we certify, again using interval arithmetic, that \( q^T u - r^T \ell < 0 \). Notice that it suffices that the value is strictly negative, as a solution of the linear system can then be obtained by scaling. The certificate consists of all upper bounds \( u_i \) such that the interval \( q_i \) contains a non-zero and all lower bounds \( \ell_i \) such that the interval \( r_i \) contains a non-zero. Notice that in contrast to the two methods above, it is possible that the certificate contains both the lower and upper bound of a variable.

If the LP has objective function value 0, we have to certify this with interval arithmetic. This is only possible if \( p \) can be represented exactly by floating point numbers and if during computation of \( pA \) the intervals remain point intervals. Still, the certificates are often trivial enough such that this is achieved. Notice that we cannot expect to obtain a method that can handle non-point intervals, since a slight change of the bounds can change the objective function value and hence the feasibility status.

4 Experiments

We have implemented our approach into iSAT. To become efficient, we further try to detect implications among the linear constraints as presented in [16] before starting the search. By doing so the number of LP solver calls is decreased. Other methods, like the elimination of variables by exploiting equalities appearing as top-level conjuncts of the formula or the simplification of the Boolean part of the input formula, both suggested in [5], have not yet been implemented; they would probably speed up our algorithm considerably. As LP solver, we use SoPlex [18, 21]; we do not use Gurobi [11] or CPLEX [13], as we plan to certify the feasibility of LPs using the approach of Althaus and Dumitriu [1] in a future version, where we will need access to the \( LU \) decomposition of the matrix.

We tested the performance of our implementation on the QF-LRA benchmarks (quantifier-free linear real arithmetic) from SMT-LIB [17], which contains SMT problems having only linear constraints. In order to be able to use them within our ICP solver, we artificially make all variables bounded between \(-10^5\) and \(10^5\). These instances can be solved by specialized solvers, which have much better running times. Nevertheless, we have chosen these benchmarks because they come from a standard benchmark library.

The iSAT solver is tuned to prove the unsatisfiability of a formula and returns \textit{unknown} whenever it finds a candidate solution. Hence, for feasible solutions it often returns \textit{unknown} and thus we cannot compare running times for those instances. Therefore, we restrict to unsatisfiable instances.

All experiments are performed on a 2.3 GHz Quad-Core AMD Opteron machine with 4 GB of physical memory running Ubuntu Linux with kernel version 2.6.32. We compiled our program with \texttt{g++} 4.4.3 with the optimization flag \texttt{-O2}.
4.1 Comparison of different LP solving techniques

We compared four different methods to solve the linear systems, all of them giving certified correct results, i.e. results that are not corrupted by possible errors in the floating point arithmetic. In addition to the three methods described in Section 3, we use an LP solver based on rational arithmetic. For this purpose, we use the LP solver Yices [8], one of the fastest SMT solvers available, which is specialized to handle strict inequalities.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>ICP w/o LPS</th>
<th>Neumaier/S</th>
<th>ICP+our</th>
<th>rational LPS</th>
<th>our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>10cl.m-i.b</td>
<td>0 U 0.09</td>
<td>27197 249.29</td>
<td>1198 146.61</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>10cl.m-i.b</td>
<td>0 U 0.03</td>
<td>0 U 0.04</td>
<td>0 U 0.04</td>
<td>0 U 0.04</td>
<td>0 U 0.03</td>
</tr>
<tr>
<td>1198 146.61</td>
<td>1429 U 2.21</td>
<td>873 U 1.17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2cl.m-i.b</td>
<td>110 ? 0.14</td>
<td>1120 ? 1.55</td>
<td>179 ? 4.54</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>2cl.m-i.b</td>
<td>562 ? 3.81</td>
<td>timeout</td>
<td>2100 U 141.69</td>
<td>3889 U 51.4</td>
<td></td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>0 U 0.01</td>
<td>0 U 0.02</td>
<td>0 U 0.02</td>
<td>0 U 0.02</td>
<td>0 U 0</td>
</tr>
<tr>
<td>25 U 0.13</td>
<td>20 U 16.22</td>
<td>27 U 0.66</td>
<td>25 U 0.56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>110 ? 0.19</td>
<td>2502 ? 5.23</td>
<td>260 ? 8.65</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>963 ? 28.92</td>
<td>memout</td>
<td>timeout</td>
<td>4962 ? 93.46</td>
<td></td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>0 U 0.01</td>
<td>0 U 0.02</td>
<td>0 U 0.02</td>
<td>0 U 0.03</td>
<td>0 U 0.02</td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>150 ? 0.69</td>
<td>131 U 62.46</td>
<td>159 U 5.17</td>
<td>150 U 3.57</td>
<td></td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>133 ? 0.24</td>
<td>2808 ? 7.65</td>
<td>272 ? 19.61</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>4265 ? 89.62</td>
<td>memout</td>
<td>timeout</td>
<td>4065 U 10.01</td>
<td></td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>0 U 0.01</td>
<td>0 U 0.02</td>
<td>0 U 0.03</td>
<td>0 U 0.02</td>
<td>0 U 0.02</td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>351 ? 2.48</td>
<td>486 U 157.94</td>
<td>394 U 20.51</td>
<td>7301 U 21.27</td>
<td></td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>142 ? 0.29</td>
<td>5411 ? 18.33</td>
<td>387 ? 31.41</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>10126 ? 299</td>
<td>memout</td>
<td>timeout</td>
<td>7301 U 21.27</td>
<td></td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>0 U 0.02</td>
<td>0 ? 0</td>
<td>0 U 0.02</td>
<td>0 U 0.02</td>
<td>0 ? 0.01</td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>954 ? 10.68</td>
<td>timeout</td>
<td>1469 U 63.3</td>
<td>964 U 25.61</td>
<td></td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>106 ? 5.37</td>
<td>17776 ? 76.72</td>
<td>269 ? 8.44</td>
<td>7358 U 70.69</td>
<td>1192 U 38.84</td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>5704 ? 244.22</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>0 U 0.03</td>
<td>0 U 0.02</td>
<td>0 U 0.02</td>
<td>0 U 0.03</td>
<td>0 U 0.02</td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>1189 ? 15.54</td>
<td>timeout</td>
<td>timeout</td>
<td>2227 U 246.17</td>
<td>2172 U 58.50</td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>170 ? 0.52</td>
<td>10231 ? 65.95</td>
<td>594 ? 51.88</td>
<td>timeout</td>
<td>timeout</td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>0 U 0.03</td>
<td>0 U 0.03</td>
<td>0 U 0.02</td>
<td>0 U 0.03</td>
<td>0 U 0.03</td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>2132 ? 42.59</td>
<td>timeout</td>
<td>timeout</td>
<td>8896 U 27.91</td>
<td></td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>179 ? 0.58</td>
<td>1877 ? 84.4</td>
<td>679 ? 300.35</td>
<td>15778 U 207.85</td>
<td>2113 U 115.45</td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>560 ? 68.28</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>106 ? 6.90</td>
<td>18938 ? 162.94</td>
<td>timeout</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>0 U 0.01</td>
<td>0 U 0.02</td>
<td>0 U 0.02</td>
<td>0 U 0.02</td>
<td>0 U 0.02</td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>7791 ? 281.53</td>
<td>timeout</td>
<td>timeout</td>
<td>timeout</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Benchmarks: the first line indicates the solver for the LP, where ICP w/o LPS means that no LP solver is used, i.e. all linear constraints are only handled with the ICP-approach. Neumaier/S that we use the approach proposed by Neumaier and Schcherbina to certify the infeasibility of the linear program and ICP is not used, ICP+our that we use our approach and the ICP-approach of iSAT, rational LPS that we use the rational LP solver Yices and ICP is not used, and our that we use only our approach and ICP is not used. For each version, we provide the size of the search tree (nodes), the running time in seconds (time) and the result (rs), where U means that the solver correctly returned unsatisfiable and ? that it returned unknown. The running time of the fastest approach was additionally marked with boldface.
Due to internals of iSAT, it terminates sometimes with unknown before the time limit. These instances cannot be considered as correctly solved.

In Table 1, we show for each method the number of nodes in the search tree and the total running time for a typical subset of the instances, i.e. the clock_synchr instances. With our approach for certifying the correctness of the floating point LP solver, 197 out of the 300 infeasible instances are solved, whereas only 193 are solved with the second best approach, i.e. using a rational LP solver. The small difference in the number of solved problem instances is explained by the fact that both approaches are very similar. However, the main advantage of our approach is that it needs approximately half of the solving time especially when focusing on instances with larger running times. Using the

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>nodes</th>
<th>time</th>
<th>basis</th>
<th>ifs</th>
<th>t_{LPS}</th>
<th>t_{Gauss}</th>
<th>#inf</th>
<th>#?</th>
</tr>
</thead>
<tbody>
<tr>
<td>10cl.w-case-s.b</td>
<td>0</td>
<td>0.03</td>
<td>31</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2cl.m-i.b</td>
<td>873</td>
<td>1.17</td>
<td>131</td>
<td>7</td>
<td>0.52</td>
<td>0.19</td>
<td>65</td>
<td>0</td>
</tr>
<tr>
<td>2cl.m-i.in</td>
<td>3889</td>
<td>51.4</td>
<td>249</td>
<td>10.91</td>
<td>24.6</td>
<td>20.49</td>
<td>367</td>
<td>65</td>
</tr>
<tr>
<td>2cl.w-cases.b</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2cl.w-cases.in</td>
<td>25</td>
<td>0.96</td>
<td>87</td>
<td>15.76</td>
<td>0.03</td>
<td>0.45</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>3cl.m-i.b</td>
<td>2153</td>
<td>3.52</td>
<td>195</td>
<td>7.34</td>
<td>2</td>
<td>0.27</td>
<td>99</td>
<td>0</td>
</tr>
<tr>
<td>3cl.m-i.in</td>
<td>4662</td>
<td>93.46</td>
<td>388</td>
<td>11.79</td>
<td>58.8</td>
<td>21.45</td>
<td>461</td>
<td>230</td>
</tr>
<tr>
<td>3cl.w-cases.b</td>
<td>0</td>
<td>0.02</td>
<td>17</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3cl.w-cases.in</td>
<td>150</td>
<td>3.57</td>
<td>128</td>
<td>17.88</td>
<td>0.51</td>
<td>2.72</td>
<td>71</td>
<td>7</td>
</tr>
<tr>
<td>4cl.m-i.b</td>
<td>4045</td>
<td>10.01</td>
<td>267</td>
<td>6.81</td>
<td>5.03</td>
<td>0.21</td>
<td>135</td>
<td>0</td>
</tr>
<tr>
<td>4cl.w-cases.b</td>
<td>0</td>
<td>0.02</td>
<td>19</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4cl.w-cases.in</td>
<td>379</td>
<td>3.38</td>
<td>173</td>
<td>19.49</td>
<td>2.72</td>
<td>6.36</td>
<td>141</td>
<td>133</td>
</tr>
<tr>
<td>5cl.m-i.b</td>
<td>7391</td>
<td>21.27</td>
<td>347</td>
<td>6.97</td>
<td>11.57</td>
<td>0.44</td>
<td>173</td>
<td>0</td>
</tr>
<tr>
<td>5cl.m-i.in</td>
<td>6964</td>
<td>25.61</td>
<td>222</td>
<td>19.79</td>
<td>9.77</td>
<td>12.4</td>
<td>233</td>
<td>14</td>
</tr>
<tr>
<td>5cl.w-cases.b</td>
<td>0</td>
<td>0.01</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5cl.w-cases.in</td>
<td>11952</td>
<td>38.84</td>
<td>435</td>
<td>6.86</td>
<td>21.9</td>
<td>0.37</td>
<td>217</td>
<td>0</td>
</tr>
<tr>
<td>6cl.m-i.b</td>
<td>0</td>
<td>0.02</td>
<td>23</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6cl.w-cases.b</td>
<td>2172</td>
<td>58.5</td>
<td>275</td>
<td>20.35</td>
<td>27.61</td>
<td>22.74</td>
<td>403</td>
<td>61</td>
</tr>
<tr>
<td>7cl.m-i.b</td>
<td>15465</td>
<td>78.39</td>
<td>531</td>
<td>7.04</td>
<td>42.77</td>
<td>0.74</td>
<td>264</td>
<td>0</td>
</tr>
<tr>
<td>7cl.w-cases.b</td>
<td>0</td>
<td>0.03</td>
<td>25</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7cl.w-cases.in</td>
<td>88666</td>
<td>277.91</td>
<td>332</td>
<td>20.91</td>
<td>171.34</td>
<td>70.51</td>
<td>1048</td>
<td>276</td>
</tr>
<tr>
<td>8cl.m-i.b</td>
<td>21113</td>
<td>115.45</td>
<td>635</td>
<td>6.8</td>
<td>59.07</td>
<td>0.69</td>
<td>279</td>
<td>0</td>
</tr>
<tr>
<td>8cl.w-cases.b</td>
<td>0</td>
<td>0.03</td>
<td>27</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8cl.w-cases.in</td>
<td>28586</td>
<td>164.25</td>
<td>747</td>
<td>7.26</td>
<td>86.45</td>
<td>0.86</td>
<td>327</td>
<td>0</td>
</tr>
<tr>
<td>9cl.m-i.b</td>
<td>0</td>
<td>0</td>
<td>29</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9cl.w-cases.b</td>
<td>0</td>
<td>0</td>
<td>29</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2. For our approach, we give some additional details for the instances that are solved. Beside the size of the search tree (nodes) and the running time (time), which can already be found in Table 1, we give the average size of the basis of the LPs (basis), the average size of the infeasible subsystem (ifs), the time needed by the LP solver (t_{LPS}), and the time needed for the Gaussian elimination (t_{Gauss}). Furthermore, we give the total number of linear systems that are declared to be infeasible by the floating point LP solver (#inf) and the number of times our method to certify infeasibility is not successful (#?).
approach of Neumaier and Scherbina to certify the infeasibility of the LPs, we can solve 174 of the instances.

We implemented the option to additionally handle linear constraints with the ICP approach and show for this setting the same numbers. This option turns out to produce worse results, e.g. it solves only 146 instead of 193 of the instances. It is not surprising that the combination of ICP and LP solving is less efficient: in this case ICP is redundant and less accurate compared to an LP solver. Furthermore, deriving that two linear constraints cannot hold at the same time by applying ICP steps can lead to a huge amount of arithmetic computations, causing the slowdown of the overall search process.

4.2 Detailed evaluation of our certifying approach

In Table 2 we show the details of our approach for certifying the correctness of the floating point based LP solver. For a subclass of the infeasible instances, we show the average size of the basis, the average size of the infeasible subsystem, the running time to solve the linear program, the running time to certify its correctness, the number of times our certifying is not successful, the number of nodes in the search tree and the total running time.

If our approach is successful, it returns exactly the same infeasible subsystem as the approach of Dhiflaoui et al., but solves a much smaller system of equations with rational arithmetic. More precisely, we solve a system with as many variables as the size of the infeasible subsystem, whereas they solve a system in the size of the basis. We show the differences in the size of the solved systems in Table 2 and do not report on the approach of Dhiflaoui et al. in our comparison on different LP certifying methods.

4.3 Summary

The integration of a floating point based LP solver into ICP can greatly reduce the running time needed to solve the benchmark instances.

Furthermore, the size of the infeasible subsets and hence the size of the system of linear equations we have to solve with rational arithmetic is very small on average, as one can see in column 

\(ifs\) (compare to column 

\(basis\)). This is helpful in two aspects: it gives small conflict clauses that can reduce the search space, and the running time for the Gaussian elimination is not the bottleneck – as opposed to solving the equation system given by the LP basis as in the work of Dhiflaoui et al. [7].

We believe that we become even more efficient, if we implement some further tricks to speed up the solution process, like storing LP solutions, carefully deciding which LPs should be solved or implementing further preprocessing techniques.
5 Conclusion

We have presented a tight integration of an LP solver into interval constraint propagation and experimented with our preliminary implementation. Already in this preliminary scenario which offers multiple possibilities for further optimization, the benefit of this integration is obvious.

As future work, we want to improve our approach, for instance by finding new methods to propagate bounds based on the linear constraints, as well as appropriate linear relaxations of nonlinear constraints.

References