Craig Interpolation in the Presence of Non-linear Constraints *

Stefan Kupferschmid and Bernd Becker

Albert-Ludwigs-Universität Freiburg, Germany
{skupfers|becker}@informatik.uni-freiburg.de

Abstract. An increasing number of applications in particular in the verification area leverages Craig interpolation. Craig interpolants (CIs) can be computed for many different theories such as: propositional logic, linear inequalities over the reals, and the combination of the preceding theories with uninterpreted function symbols. To the best of our knowledge all previous tools that provide CIs are addressing decidable theories. With this paper we make Craig interpolation available for an in general undecidable theory that contains Boolean combinations of linear and non-linear constraints including transcendental functions like $\sin(\cdot)$ and $\cos(\cdot)$. Such formulae arise e.g. during the verification of hybrid systems. We show how the construction rules for CIs can be extended to handle non-linear constraints. To do so, an existing SMT solver based on a close integration of SAT and Interval Constraint Propagation is enhanced to construct CIs on the basis of proof trees. We provide first experimental results demonstrating the usefulness of our approach: With the help of Craig interpolation we succeed in proving safety in cases where the basic solver could not provide a complete answer. Furthermore, we point out the (heuristic) decisions we made to obtain suitable CIs and discuss further possibilities to increase the flexibility of the CI construction.

Key words: SAT, SMT, Craig Interpolation, Interval Arithmetic, BMC

1 Introduction

The analysis and verification of hybrid systems is an important task, e.g. in the automotive or aviation industry. Wherever complex systems are developed for applications with safety critical aspects, the developers must fulfill a number of safety requirements, which in general slows down the process of development and increases its costs. This has motivated the development of tools that can deal with the verification task of such systems. One technique widely used in multiple verification tools is Craig interpolation. In this paper we present a method that allows the construction of *Craig interpolants* (CIs) [1] for arbitrary Boolean combinations of linear and non-linear constraint formulae.

* 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). Refer to www.avacs.org.

CIs are of particular use in bounded model checking (BMC). In [2] McMillan extended the traditional BMC procedure for Kripke structures to an unbounded model checking algorithm, i.e. a procedure that is able to show that a given system is safe in the sense that a certain safety property always holds. In [3, 4] this work is extended to the quantifier-free theory of linear inequalities and uninterpreted function symbols. The authors of [5] compute optimized representations for non-convex polyhedra with the help of Craig interpolation. More in detail, Craig interpolation here is used to remove redundant linear constraints with the aid of a *Satisfiability Modulo Theories* (SMT) solver. A representative for a state-of-the-art SMT solver that can produce CIs is presented in [6], but this work is limited to linear constraints. Since we focus on verification of hybrid systems, we are interested in a solver that can deal with the inherent linear and non-linear behavior of such systems. Our contributions are: (i) we generalize proof-based construction rules for CIs [2] to formulae containing non-linear constraints and prove the correctness of the rules, (ii) we built these modifications on top of the SMT solver iSAT [7], (iii) experimental results illustrate that the so computed CIs can be used in a similar fashion as suggested by [2] to verify safety properties but this time for systems containing non-linear dynamics; last but not least, we discuss the choices we made for the CI construction and discuss further possibilities to influence the construction of CIs and possibly find "good" CIs.

The remainder of this paper is structured as follows: First we describe the core of iSAT, the underlying solver used in this paper and then show how this solver can produce proofs of unsatisfiability. Next we present the construction rules to achieve valid CIs. We then present some promising results, illustrating that interpolation can be successfully applied to BMC problems containing non-linear constraints. Before concluding the paper we discuss how the strength of a CI can be influenced by analysing the slackness between contradictionary theory constraints that are detected during the solving process of iSAT.

2 Foundations

As already mentioned above, on the one hand side we of course need a solver that is able to handle linear as well as non-linear constraints, and on the other hand provides a proof in the case of unsatisfiability. In order to construct CIs and do the experimental work, we modified the SMT solver iSAT [7] in a corresponding way. Nevertheless, the main construction principles together with the applications presented should be transferable to any SMT solver that is capable of handling linear and non-linear constraints.

Since the solver underlying our work is iSAT, we provide a short description of the solver as far as it is necessary for the understanding on generating CIs. For more information on iSAT refer to [7].

2.1 Basics on iSAT

The iSAT algorithm aims at solving Boolean combinations of mixed linear and non-linear constraint formulae (including transcendental functions). The unde-

cidability of this theory in general follows from the fact that it is undecidable to answer whether a Diophantine equation has an integer solution or not [8].

iSAT contains an integration of a Davis-Putnam-Logemann-Loveland (DPLL) procedure [9,10] and interval constraint propagation (ICP)[1] allowing it to reason about (highly non-linear) arithmetic constraints. In contrast to decidable theories, iSAT can not always classify a problem as "satisfiable" or "unsatisfiable". Instead, the result can also be "unknown". However, iSAT can handle arithmetic expressions like $\sin(\cdot)$, $\cos(\cdot)$ or $\exp(\cdot)$ which are not supported by common SMT solvers.

The solving process of iSAT consists of two phases. First iSAT transforms an arbitrary Boolean combination of linear and non-linear constraints into an equisatisfiable *Conjunctive Normal Form* (CNF) with normalized constraints using the following syntax:

$$\begin{aligned}
\mathit{formula} &::= \{\mathit{clause} \wedge\}^* \mathit{clause} \\
\mathit{clause} &::= (\{\mathit{atom} \vee\}^* \mathit{atom}) \\
\mathit{atom} &::= \mathit{simple_bound} \mid \mathit{arithmetic_predicate} \\
\mathit{simple_bound} &::= \mathit{variable}\ \mathit{relation}\ \mathit{rational_const} \\
\mathit{arithmetic_predicate} &::= \mathit{variable}\ \mathit{relation}\ \mathit{uop}\ \mathit{variable} \mid \\
&\quad\ \mathit{variable}\ \mathit{relation}\ \mathit{variable}\ \mathit{bop}\ \mathit{variable} \\
&\quad\ \mathit{variable}\ \mathit{relation}\ \mathit{rational_const}\ \mathit{bop}\ \mathit{variable}
\end{aligned}$$

In the above syntax, *uop* and *bop* are unary and binary operation symbols respectively, including $+$, $-$, $\times$, $\sin(\cdot)$, etc., *rational_const* ranges over the rational constants, and $\mathit{relation} \in \{<, \leq, =, \geq, >\}$. To illustrate this phase, imagine that we have the following formula:

$$(x \geq 0) \wedge (x \leq 10) \wedge ((\sin(1/3x) + \sqrt{x} \geq y) \implies (y \geq 1/4x + 3)) \tag{1}$$

First we eliminate the Boolean operators by applying a Tseitin-transformation [12], e.g. 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 $r_3 = \sqrt{x}$. Finally, the normalized CNF problem looks like follows:

$$\begin{aligned}
&(b) \wedge (x \geq 0) \wedge (x \leq 10) \wedge (\overline{b} \vee r_2 + r_3 < y \vee y \geq r_4 + 3) \wedge \\
&(r_2 + r_3 \geq y \vee b) \wedge (y < r_4 + 3 \vee b) \wedge \\
&(r_1 = 1/3x) \wedge (r_2 = \sin(r_1)) \wedge (r_3 = \sqrt{x}) \wedge (r_4 = 1/4x)
\end{aligned} \tag{2}$$

Now all clauses are consistent with the syntax described above and can be transferred to the solver. In the remainder of the paper we will assume that a normalization has been performed in advance and thus the formula φ considered is normalized.

Before describing the solving process in detail, we informally define the underlying semantics. A constraint formula φ is satisfied by a valuation of its

[1] cf. [11] for an extensive survey

variables iff all its clauses are satisfied, that is, iff at least one atom is satisfied in any clause. An atom is satisfied wrt. the standard interpretation of the arithmetic operators and the ordering relations over the reals. A constraint formula φ is *satisfiable* iff there exists a satisfying valuation, referred to as a *solution* of φ. Otherwise, φ is *unsatisfiable*. We remark that by definition of satisfiability, a formula φ including or implying the empty clause, denoted by $\bot$, cannot be satisfied at all, i.e. if $\bot \in \varphi$ or $\varphi \implies \bot$ then φ is unsatisfiable.

Instead of real-valued variable valuations, iSAT manipulates interval ranges. Using the function $\rho : \mathit{Var} \to \mathbb{I}_{\mathbb{R}}$, where Var is a set of variables and $\mathbb{I}_{\mathbb{R}}$ is the set of convex subsets 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 $[-3, 5] \subset \mathbb{Z}$, for example. The Boolean domain is represented by $\mathbb{B} = [0, 1] \subset \mathbb{Z}$. If both ρ' and ρ are interval valuations, then ρ' is called a *refinement* of ρ iff $\rho'(v) \subseteq \rho(v)$ for each variable $v \in \mathit{Var}$. 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 $\mathit{lower}(\rho(x))$ and $\mathit{upper}(\rho(x))$, respectively. E.g., for the interval $\rho(x) = (-4, 9]$ we have $\mathit{lower}(\rho(x)) = (x > -4)$ and $\mathit{upper}(\rho(x)) = (x \leq 9)$.

Let x and y be variables, ρ be an interval valuation, and $\circ$ be a binary operation. Then $\rho(x \circ y)$ denotes the *interval hull* of $\rho(x) \hat{\circ} \rho(y)$ (i.e. the smallest enclosing interval which is representable by machine arithmetic), where the operator $\hat{\circ}$ corresponds to $\circ$ but is canonically lifted to sets. This is done analogously for unary operators. We say that an atom a is *inconsistent* under an interval valuation ρ, referred to as $\rho \sharp a$, iff no values in the intervals $\rho(x)$ of the variables x in a satisfy the atom a, i.e.

$$\begin{array}{lll}
\neg\exists v \in \rho(x) & : v \sim c & \text{if } a = (x \sim c), \\
\neg\exists v \in \rho(x), \neg\exists v' \in \rho(\circ y) & : v \sim v' & \text{if } a = (x \sim \circ y), \\
\neg\exists v \in \rho(x), \neg\exists v' \in \rho(y \circ z) & : v \sim v' & \text{if } a = (x \sim y \circ z)
\end{array}$$

where $\sim \in \{<, \leq, =, \geq, >\}$. Otherwise a is *consistent* under ρ. For instance, the constraint $x = 3 \cdot y$ is inconsistent for $\rho(x) = [5, 10]$ and $\rho(y) = [-2, 1]$.

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 confer [7]. In Algorithm 1, the pseudocode of iSAT is given. 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. an evaluated clause becomes inconsistent 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 8 with either *unsat*, *sat*, or *unknown*.

```
1 Data: CNF F
2 Result: sat, unsat or unknown
  /* Main DPLL loop. DecideVar returns false once the msw for all    */
  /* variables is reached, and no further decisions are possible.    */
3 while decideVar() do
  |  /* Propagates current decision and unit constraints.           */
4 |  if propagateICP() = Conflict then
  |  |  /* Function tries resolve the conflict by backtracking. If  */
  |  |  /* the conflict is unresolvable, problem is unsatisfiable.  */
5 |  |  if analyseBacktrack() = Unresolvable then return unsat;
6 |  end
7 end
  /* Final test to see if all the constraints are satisfied.        */
8 if allClausesSat() then return sat; else return unknown;
```

Algorithm 1: DPLL + ICP

As termination can not be guaranteed by dividing an interval range indefinitely, iSAT stops making decisions when every problem variable has reached a current interval width that is less or equal to a given minimal width. We call this width the minimal splitting width (msw). It is still possible for this current ρ to contain a solution, as such, this result is sometimes referred to as a *Candidate Solution*. However, if iSAT found an arising conflict as unresolvable during the conflict analysis routine the problem formula is classified as unsatisfiable. As our construction of CIs is proof-based, we describe in the next section how these certificates can be computed.

2.2 Proof Certificates in iSAT

This section summarises the work of [13] where iSAT is used to produce certificates. We introduce the two rules used in iSAT and provide simple examples. For a detailed description refer to [13].

The first rule is based on unit propagation, and the second one is based on resolution. Conceptually, the rules from modern SAT solvers for proof generation are adapted to our context.

In order to apply the deduction rule, there must be a clause cl that contains at most one atom a_i that does not evaluate to $false$ under the current interval assignment ρ. In other words, clause cl is either a unit-clause with its unit-literal a_i, or it is the conflicting clause. In order to extend ρ to a satisfying assignment we have to satisfy atom a_i. With the help of interval arithmetic, iSAT tries to derive new upper and/or lower bounds for the variables contained in a_i. Assume we want to solve a CNF formula φ that contains the clause $cl = (x < -8 \vee y = x^2)$, and the current variable intervals defined by ρ are $x \in [3, 7]$ and $y \in [-2, 25]$. Here, the first atom of cl is inconsistent because there exists no value between 3 and 7 that is less than -8 ($\rho \sharp (x < -8)$). The second atom, $y = x^2$, is therefore the unit literal of cl. The reason why a clause cl is a unit-clause is given by

a subset of the current interval assignment ρ containing all current lower and upper bounds that are responsible for the inconsistency of the remainder atoms ($reason\_unit(cl, y = x^2, \rho) = \{x \geq 3\}$). Now, we know that $x \in [3, 7]$ and we further have the unit literal $y = x^2$. Putting this information together we see that y must be in the interval $[9, 49]$. Since the current range is $y \in [-2, 25]$, we can derive a new lower bound for variable y that is $y \geq 9$ by applying ICP ($(x \geq 3) \overset{y=x^2}{\rightsquigarrow} (y \geq 9)$), and prune away unsatisfiable parts of the search space. Differently spoken, the clause derived by deduction contains the negation of the reasons and the derived new bound information as literals. In the more general case, we write $c_1 \rhd c_2$ to express that c_2 can be derived by applying the deduction rule on c_1. More formally, this rule is defined as follows:

$$\frac{\begin{array}{c} cl = (a_1 \vee \ldots \vee a_n), \\ \exists \rho : \exists i \in \{1, \ldots, n\} : \forall j \neq i : \rho \sharp a_j, (b'_1, \ldots, b'_k) \overset{a_i}{\rightsquigarrow} (b'), \\ \{b'_1, \ldots, b'_k\} \subseteq \{lower(\rho(x)), upper(\rho(x)) : x \in a_i\}, \\ reason\_unit(cl, a_i, \rho) = \{b_1, \ldots, b_m\} \end{array}}{(\neg b_1 \vee \ldots \vee \neg b_m \vee \neg b'_1 \vee \ldots \vee \neg b'_k \vee b')} \tag{3}$$

Therefore the deduction rule for the just presented example is:

$$\frac{\begin{array}{c} cl = (x < -8 \vee y = x^2), \\ \rho \sharp (x < -8), (x \geq 3) \overset{y=x^2}{\rightsquigarrow} (y \geq 9), \\ reason\_unit(cl, y = x^2, \rho) = \{x \geq 3\} \end{array}}{(x < 3 \vee y \geq 9)}$$

The second rule used in iSAT is resolution. In state-of-the-art SAT solvers, resolution is performed during conflict analysis. The same holds for iSAT. This rule can be defined as:

$$\frac{\begin{array}{c} c_1 = (a \vee a_1 \vee \ldots \vee a_n), c_2 = (b \vee b_1 \vee \ldots \vee b_m) \\ a, a_1, \ldots, a_n, b, b_1, \ldots, b_m \text{ are simple bounds} \\ a = (x \sim k), b = (x \sim' k'), \{v : v \sim k\} \cap \{v : v \sim' k'\} = \emptyset \end{array}}{c_{res} = (a_1 \vee \ldots \vee a_n \vee b_1 \vee \ldots \vee b_m)} \tag{4}$$

Resolution is applied on two clauses c_1 and c_2 where both clauses contain only simple bounds. In order to apply this rule, c_1 must contain a simple bound a that is contradictory to a simple bound b from clause c_2. To illustrate this rule consider the following two clauses: $c_1 = (x > 4 \vee y \leq 6 \vee z < 5)$ and $c_2 = (x < -8 \vee w > 3)$. The simple bound $x > 4$ from clause c_1 and the simple bound $x < -8$ from clause c_2 cannot be valid at the same time. By resolving these two simple bounds we generate the following resolvent $c_{res} = (y \leq 6 \vee z < 5 \vee w > 3)$. We just write $res(c_1, c_2, p) \vdash c_{res}$ as an abbreviation for the fact that c_{res} can be derived by applying the resolution rule to clauses c_1 and c_2 on variable p. Before presenting the construction of CIs in iSAT in the next section, we want to take a deeper look at the resolution steps. At first glance the resolution rule is very similar compared to resolution between two propositional clauses. But in the

just presented example the resolved simple bounds $x > 4$ and $x < -8$ contain besides the contradictory information also information about the *strength* of the contradiction. We call this *slackness information* and it can be obtained by computing the distance between these bounds, here $|4 - (-8)| = 12$. This is one of the main reasons why we implemented the generation of CIs in an interval arithmetic based SMT solver as we want to use the slackness information in later versions of iSAT to influence the construction of these interpolants. In order to understand the construction of CIs in iSAT, it is important to know the two rules, namely resolution and deduction. To illustrate how iSAT uses these rules, we present a small example. Suppose iSAT has to solve the normalized CNF φ:

$$\varphi = c_1 \wedge \overbrace{(\sin(x) < 0.3 \vee y < 7.5)}^{c_2} \wedge \overbrace{(y - x \leq 8 \vee b)}^{c_3} \wedge c_4 \wedge \ldots \wedge c_n$$

Here x and y are real valued variables and b is a Boolean variable. At each step iSAT maintains the current interval valuation ρ. Suppose that the current interval valuation is: $x \in [0, 1]$, $y \in [7, 10]$, and b is unassigned. Next, iSAT makes a decision on variable y by splitting the current interval at its midpoint, e.g. $y \in [8.5, 10]$. While propogating this decision, iSAT detects that clause c_2 is now unit as literal $(y < 7.5)$ evaluates to *false* under the current interval valuation. The unit-literal is $\sin(x) < 0.3$ and is used to fuel deduction. iSAT then derives a new upper bound $x \leq 0.31$ for variable x (more in detail, based on the deduction rule we may conclude $c_2 \rhd (y < 8.5 \vee x > 1 \vee x \leq 0.31)$). Again no conflicts are encountered, so iSAT is free to make another decision, and for instance sets b to *false*. With this decision clause c_3 becomes unit and literal $(y - x \leq 8)$ is used for deduction. Under the current interval valuation of x and y a new upper bound for variable y via ICP is deduced: $y \leq 8.31$ $(c_3 \rhd (b \vee x > 0.31 \vee y \leq 8.31))$. This newly derived bound conflicts with the current interval valuation $y \in [8.5, 10]$. iSAT resolves this conflict by analyzing the implication graph. Afterwards a conflict clause is learned to prune away unsatisfiable parts of the search space. During the conflict analysis a partial proof will be generated based on the two rules namely deduction and resolution. This partial proof is shown in Fig. 1. Moreover, iSAT has been modified to produce so-called partial CIs during conflict analysis.

As usual, we define $sup(F)$ to be the set of variables contained in a formula F. Then, according to [1] a CI is defined as follows.

Definition 1 (Craig Interpolant (CI)). *Let A and B be formulae with the property that $A \wedge B$ is unsatisfiable. A formula I is referred to as a CI if the following three properties hold:*

1. $sup(I) \subseteq sup(A) \cap sup(B)$
2. $\models A \Rightarrow I$
3. $\models I \Rightarrow \neg B$

We call I a *Craig interpolant* or just *interpolant* for the formula pair A and B. This interpolant contains only variables occurring in $sup(A) \cap sup(B)$ and can be seen as an over-approximation of the formula A that is completely disjoint

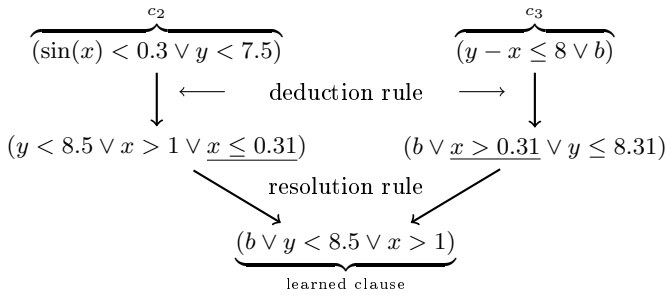

Fig. 1. Partial Proof Tree

from B (i.e. the intersection of I and B is empty). Now, using these basics, the next sections will show how CIs are calculated and how they are used in hybrid system verification by iSAT.

3 Construction of Craig Interpolants using iSAT

In Fig. 1 of Section 2 we showed how iSAT produces a partial proof. In this section we present how iSAT can compute *partial interpolants* using a partial proof. First we provide the construction rules, before proving their soundness. Furthermore, we will apply these rules to the partial proof shown in Fig. 1.

With every formula pair (A, B) we associate the following three sets of variables: 1) G contains all the variables occurring in both formulae, i.e. $G = sup(A) \cap sup(B)$, 2) L_A contains only variables located in the A-formula but not in the B-formula, and 3) L_B and contains all variables from the B-formula but not located in the A-formula. We assume that the formula pair (A, B) is in CNF format and we have a proof tree that derives the empty clause from $A \wedge B$. For every node c in the proof tree (internal nodes correspond to clauses derived by applying deduction or resolution) we define construction rules that generate partial interpolants. Note, all internal nodes of the proof tree, including the empty clause $\perp$, contain only simple bounds. With this in mind, we define the construction rules and the concept of projection as follows:

Definition 2 (Construction Rules). *Let c be a node in the proof tree corresponding to a clause containing only simple bounds. Then we denote the partial interpolant of clause c by $pi(c)$, and define*

$$pi(c) = \begin{cases} \bigvee\limits_{l_j \in c, var(l_j) \in G} l_j & : \quad (c \in A) \text{ or } (\exists c' \in A : c' \rhd c) \\ pi(c_1) \wedge pi(c_2) & : \quad res(c_1, c_2, p) \vdash c \text{ and } p \in G \cup L_B \\ pi(c_1) \vee pi(c_2) & : \quad res(c_1, c_2, p) \vdash c \text{ and } p \in L_A \\ true & : \quad else \end{cases}$$

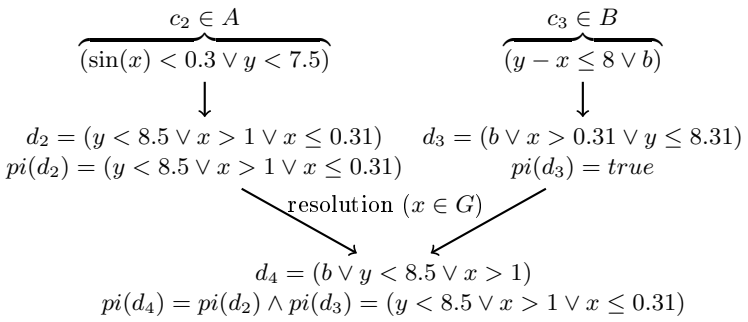

Fig. 2. Partial Interpolant

Definition 3 (Projection). *Let Θ be a disjunction of simple bound literals $\Theta = \bigvee_{j=1}^{k} l_j$ with $var(l_j) \in G \cup L_A \cup L_B$. Then, $\Theta \mid_A$ is the projection of Θ to L_A and $\Theta \mid_B$ is the projection of Θ to $L_B \cup G$:*

$$\Theta \mid_A = \bigvee_{var(l_j) \in L_A}^{k} l_j \qquad \Theta \mid_B = \bigvee_{var(l_j) \in G \cup L_B}^{k} l_j$$

In order to prove the correctness of the construction rules we prove the soundness of the following lemma:

Lemma 4. *Let $F = A \wedge B$ be a CNF formula that is unsatisfiable and let P be a proof of the unsatisfiability. Then for every internal proof node c, the partial interpolant $pi(c)$ is a CI of the formula pair (A', B') with $A' = A \wedge \neg(c \mid_A)$ and $B' = B \wedge \neg(c \mid_B)$.*

Before giving a proof sketch of the lemma above we note the following: Lemma 4 implies that $pi(\bot)$ is a valid CI for the formula pair (A, B). This is clear as $A' = A \wedge \neg(\bot|_A) = A \wedge true = A$ and $B' = B \wedge \neg(\bot|_B) = B \wedge true = B$. We will now illustrate how iSAT computes a partial interpolant for the partial proof in Fig. 1. Suppose clause c_2 belongs to the clause set A and clause c_3 belongs to B. Further, let $x \in G$, $y \in G$ and $b \in L_B$. In Fig. 2 we decorated the partial proof with the corresponding partial interpolants by applying the construction rules (Definition 2). For the computed partial interpolant $pi(d_4)$ of the clause d_4 we will now show that Lemma 4 is valid. To do so we compute $A' = A \wedge \neg(d_4 \mid_A)$ and $B' = B \wedge \neg(d_4 \mid_B)$. As d_4 does not contain variables from L_A, we know $\neg(d_4 \mid_A) = true$. Because of $\models A \Rightarrow d_2$ ($c_2 \in A$ and $c_2 \rhd d_2$) and $pi(d_4) = d_2$, we conclude $\models A' \Rightarrow pi(d_4)$. In order to show $\models pi(d_4) \Rightarrow \neg B'$ we show that the negation is unsatisfiable ($pi(d_4) \wedge B'$). To obtain B' we have to compute $\neg(d_4 \mid_B) = \neg b \wedge (y \geq 8.5) \wedge (x \leq 1) = f_4$. We know that $\models B' \Rightarrow d_3$ as $c_3 \rhd d_3$ and $c_3 \in B$. It is easy to see that f_4 and d_3 imply $f_5 = (x > 0.31)$. Furthermore, formula part f_4 together with $pi(d_4)$ imply $f_6 = (x \leq 0.31)$ which contradicts f_5 and thus proves the unsatisfiability of $pi(d_4) \wedge B'$.

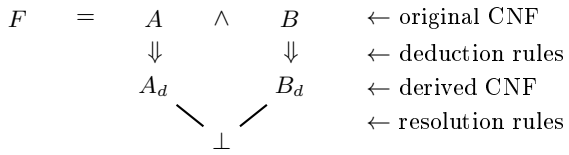

Fig. 3. Motivation

The proof of Lemma 4 is further motivated by the following fact: Suppose iSAT has classified a problem formula as being unsatisfiable by applying resolution. The resolution steps are either performed on clauses containing only simple bounds (e.g. $x > 0.31$) as literals or on simple bounds that have been derived using the deduction rule. To visualize this, Fig. 3 illustrates how iSAT derives the empty clause. Assume $F = A \wedge B$ is unsatisfiable. Then the deduction rule can only be applied to clauses from either A or B. For a clause $cl_{derived}$ derived by the deduction rule, we can state the following:

$$\begin{aligned} &\text{if } c \in A,\ c \rhd cl_{derived} \text{ then } \models A \Rightarrow cl_{derived} \\ &\text{if } c \in B,\ c \rhd cl_{derived} \text{ then } \models B \Rightarrow cl_{derived} \end{aligned} \tag{5}$$

Imagine that through deduction, iSAT was able to derive all clauses needed to produce the empty clause. In Fig. 3, the clause sets A_d and B_d contain all these derived clauses. iSAT derives the empty clause $\bot$ by applying resolution on clauses that are contained in $A_d \cup B_d$. Remember, by definition of the deduction rule, all clauses derived by this rule contain only simple bounds as literals. If we construct a CI I for the two clause sets A_d and B_d, it is also a valid CI for the original defined clause sets A and B. This is because A_d and B_d are over-approximations of A and B. As I is a CI for A_d and B_d we conclude that $\models A_d \Rightarrow I$ and $\models I \Rightarrow \neg B_d$. Together with (5) we conclude $\models A \Rightarrow A_d \Rightarrow I$ and $\models I \Rightarrow \neg B_d \Rightarrow \neg B$. Now we provide a proof sketch of Lemma 4:

Proof Sketch.[2] The proof is similar to that presented in [14]. There, the authors proved that the symmetric construction rules presented by Pavel Pudlàk's algorithm [15] are sound. In our case we have asymmetric rules (similar to McMillan [2]). But it is still possible to prove the invariant of Lemma 4. The proof is given by induction over the depth of a proof tree P computed by iSAT. This makes it necessary to distinguish between different cases and every case itself is proved by showing that the following three properties are valid: (1) For every literal l of $pi(c)$ it holds that $var(l) \in G$, (2) $A' \Rightarrow pi(c)$, and (3) $pi(c) \Rightarrow \neg B'$.

4 iSAT and BMC with Craig Interpolation

As mentioned in Section 1, CIs can be applied in BMC. By using interpolants, McMillan [2] modified a normal BMC procedure for so called Kripke Structures

[2] A detailed proof can be found in the appendix.

in such a way that the modified procedure could prove safety properties of a given system. Here, we extend the BMC approach of McMillan to our context so that systems described through rich arithmetic constraints can be verified. Initial experiments with an extension of iSAT as underlying solver illustrate the usefulness of this method. Before presenting the experimental results, we will shortly summarize the main ideas on how BMC can be turned into a proof system.

4.1 Basics

A BMC problem consists of a predicate $INIT(\boldsymbol{x}_0)$ describing the initial state, a predicate $TRANS(\boldsymbol{x}_i, \boldsymbol{x}_{i+1})$ defining how variables change from step i to step $i+1$, and lastly a predicate describing unsafe system states $TARGET(\boldsymbol{x}_k)$. A system trace is then defined as follows:

$$\Phi_k = INIT(\boldsymbol{x}_0) \wedge \bigwedge_{i=0}^{k-1} TRANS(\boldsymbol{x}_i, \boldsymbol{x}_{i+1}) \wedge TARGET(\boldsymbol{x}_k)$$

Here, the value k is called the *depth*, and the classical BMC approach tries to detect whether or not a system $\mathcal{S}$ can reach an unsafe state at a certain *depth*. This is done by iteratively checking whether $\Phi_0, \Phi_1, \ldots, \Phi_k$ is satisfiable or not. If no failures for large values of k can be found it could be the case that the target state is unreachable for every k. One approach to prove this is done by checking that all reachable states have been proven to be safe. To accomplish this, we can first check the initial reachable states, exactly those described through $INIT(\boldsymbol{x}_0)$. We define the state set that is reachable in exactly k transition steps through:

$$REACH_k = \exists \boldsymbol{x}_0, \ldots, \boldsymbol{x}_{k-1} INIT(\boldsymbol{x}_0) \wedge \bigwedge_{i=0}^{k-1} TRANS(\boldsymbol{x}_i, \boldsymbol{x}_{i+1})$$

One way to check whether all states have been explored by normal BMC and searching depth k could be:

$$REACH_k[\boldsymbol{x}_k/\boldsymbol{x}] \Rightarrow REACH_{k-1}[\boldsymbol{x}_{k-1}/\boldsymbol{x}] \vee \cdots \vee REACH_1[\boldsymbol{x}_0/\boldsymbol{x}] \vee INIT(\boldsymbol{x})$$

This is called a fixed-point-check (FPC). The notation $[\boldsymbol{x}_{k-1}/\boldsymbol{x}]$ stands for the substitution of the vector x_{k-1} through the vector x. The check above has the disadvantage of containing several $\exists$-quantifiers which require many quantifier eliminations to be performed in order to solve the formula. There are two main issues associated with quantifier elimination. The first is elimination of quantifiers can lead to an exponential blowup in the size of the formula. The second issue occurs when solving problems that contain transcendental functions (as we do), as there exist no such elimination rules. To obtain an alternative solution, Craig interpolation can be used. We define:

Definition 5 ($PREF_l$, $SUFF_l^k$)**.** *Given Φ_k we define $PREF_l$ and $SUFF_l^k$ as:*

$$PREF_l = INIT(\boldsymbol{x}_0) \wedge \bigwedge_{i=0}^{l-1} TRANS(\boldsymbol{x}_i, \boldsymbol{x}_{i+1})$$

$$SUFF_l^k = \bigwedge_{i=l}^{k-1} TRANS(\boldsymbol{x}_i, \boldsymbol{x}_{i+1}) \wedge \bigvee_{i=k-l}^{k} TARGET(\boldsymbol{x}_k)$$

Here l is the parameter responsible for the number of over-approximated transition steps ($l > 0$). If Φ_k is unsatisfiable, a CI p is computed for the formulae $A = PREF_l$ and $B = SUFF_l^k$. If p implies the initial state, a fixed-point has been reached (i.e. it has been proved that the target state is unreachable). If the initial state is not implied by the CI, we continue increasing the depth k by using p as a new initial state as p is an over-approximation of all states reachable in l steps from the initial state. By setting p as the new initial state the number of added unwindings has been increased by l because p over-approximates all the states reachable in l transition steps. Going on like this could eventually lead to a satisfiable problem formula. This does not mean that the target state is reachable as our initial state is an over-approximation, which can result in a spurious counterexample being detected. In these cases, the solver would then discard the previously calculated CIs, and start a new BMC run at the current unroll depth. For more details please refer to [2]. We want to remark that the challenge in finding a fixed-point depends highly on the generated over-approximations represented by the computed CIs. A problem arising in the context of BMC and Craig interpolation is found in the *strengths* of the computed interpolants. If a CI or over-approximation is to close to the exact reachable states we will have to iterate this procedure many times until a fixed-point is detected. Such CIs are called *strong*. However, if they are too *weak* it can happen that we will often detect counterexamples in the over-approximations. In order to compute interpolants of different strength in the future, we are going to take the slackness information provided by iSAT into account when computing CIs.

4.2 Experiments with iSAT

We implemented the presented approach into the solver iSAT. The data structure used in the iSAT extension stores the partial CIs as a modified *And-Inverter-Graph* (AIG) [5]. This can be done as the CIs in this case are Boolean combinations of simple bounds. Such formulae can be encoded as an equisatisfiable Boolean formula by introducing Boolean variables for each simple bound and further adding constraints that encode the corresponding relations among these simple bounds. The benefit of this data structure is that it supports all Boolean operations needed to construct interpolants. Further, the AIG package can perform satisfiability checks needed in iSAT's FPC routine when applying BMC and Craig interpolation. To get a better picture of how the CIs that iSAT produces look like, we will first give a small two dimensional example. Let A and B be two formulae defined as:

$$\begin{aligned} A := & ((x < 2.5) \Rightarrow (y \geq 2\sin(x))) \\ & \wedge((x \geq 2.5 \wedge x < 5) \Rightarrow (y \geq 0.125x^2 + 0.41)) \\ & \wedge((x \geq 5 \wedge x \leq 6) \Rightarrow (y \geq -0.5x + 6.04)) \\ B := & ((x < 3) \Rightarrow (y \leq -0.083 + (x\cos(0.1\exp(x))))) \\ & \wedge((x \geq 3 \wedge x \leq 6) \Rightarrow (y \leq -x^2 + 10x - 22.35)) \end{aligned} \tag{6}$$

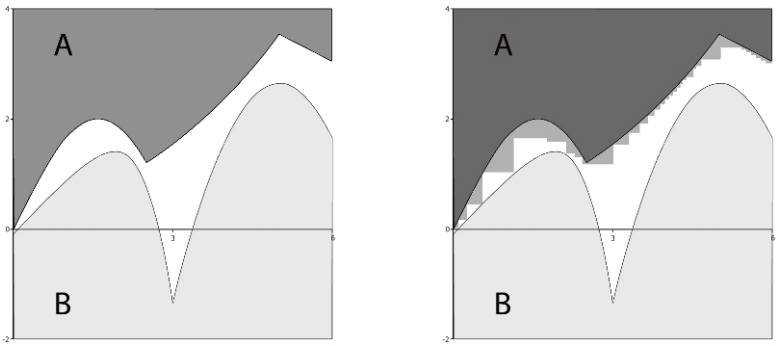

Fig. 4. On the left side you see two formulae A and B with the property that $A \wedge B$ is unsatisfiable. On the right side a CI for the formula pair (A, B) is pictured. The CI has been computed by using iSAT.

Using the initial bounds $x \in [0, 6]$ and $y \in [-2, 4]$, the problem can be visualized on the left hand side of Fig. 4. In this figure, the region where the formula A (B) is satisfied is labeled with **A** (**B**). It is quite obvious that $A \wedge B$ is unsatisfiable as the intersection of the two regions is empty, and iSAT is easily able to find an AB-refutation. A CI which is generated on-the-fly can be seen on the right hand side of Fig. 4. The CI ci covers the region of A and is thus implied by A. As ci has an empty intersection with B it directly follows that $ci \wedge B$ is unsatisfiable. The shape of the interpolant is a combination of boxes. This is explained by the construction rules and the fact that iSAT only performs resolution on clauses containing simple bounds as literals.

To show the usefulness of the CIs that iSAT can produce, we studied six different BMC benchmarks together with some valid safety properties. The transition relations of these benchmarks contain non-linear and linear equations. Of course, even our approach is not designed for pure linear systems, it should in principle work for such systems. To show this, we modeled two linear systems presented by Alur et al. in [16]. The first system describes a *thermostat* and the second one is a version of a *leaking gas burner*.

The first non-linear problem is called *the logistic map* [17] and is a polynomial mapping of degree 2. Mathematically, the logistic map is written as $x_{n+1} = r \cdot x_n(1 - x_n)$ where x_n is a number between zero and one. This map illustrates chaotic behaviour, but can exhibit periodic behavior by setting $r = 3.2$. When $r = 3.2$, the logistic map oscillates between two values, and we defined the safety regions to be $(0.78 \leq x \wedge x \leq 0.82) \vee (0.48 \leq x \wedge x \leq 0.52)$ (approximately 0.8 and 0.5).

The next example is the Hènon map [18], a chaotic map introduced by Michel Hènon and mathematically defined as $x_{n+1} = y_n + 1 - ax_n^2$ and $y_{n+1} = bx_n$. The map depends on two parameters a and b. Setting $a = 1.25$ and $b = 0.3$ makes the Hènon map oscillating between seven different values. The safety properties for

Benchmark	#Decisions	#Deductions	#AigNodes	Depth	Time FPC	Time
hènon	6797	13799353	10260	283	0.8	45.26
logistic	2085	596984	3187	60	0.05	2.02
accelerate	13	1809	88	7	0.00	0.02
cruise control	384	99841	1960	55	0.01	0.27
thermostat	346	93855	105291	6	0.19	1.18
gas burner	6189	3439885	34105	21	0.31	24.12

Table 1. Results

these maps are the disjunction of small intervals containing the periodic values in a similar fashion to the logistic map case.

Next, we consider two BMC problems describing an accelerating car. Taking the air resistance into account, the relationship between the car's velocity and the physical drag contains quadratic functions. The first benchmark describes the velocity of a car that is accelerating with constant force. Due to the air resistance the car cannot drive faster than $49.61\frac{m}{sec}$ which is our safety property. The initial state (INIT) and the transition relation (TRANS) of this hybrid system are:

$$INIT := \begin{array}{ll} v_{car_0} = 0 \wedge & // \text{ velocity at step } 0 \\ F_{res_0} = 1000 \wedge & // \text{ resultant force at step } 0 \\ a_{car_0} = 0.0005 \cdot F_{res_0} & // \text{ acceleration at step } 0 \end{array} \tag{7}$$

The transition relation computes the resultant force $F_{res_{i+1}}$ at time step $i+1$. In order to compute $F_{res_{i+1}}$ we need to compute F_{air_i} first. In this example the interaction caused by the drag simplifies to $F_{air_i} = 0.40635 \cdot v_{car_i}^2$.

$$TRANS := \begin{array}{ll} F_{res_{i+1}} = 1000 - F_{air_i} \wedge & // \text{ resultant force at step } i+1 \\ F_{air_i} = 0.5418 \cdot v_{car_i}^2 \wedge & // \text{ drag force at step } i \\ a_{car_{i+1}} = 0.0005 \cdot F_{res_{i+1}} \wedge & // \text{ acceleration at step } i+1 \\ v_{car_{i+1}} = v_{car_i} + a_{car_i} & // \text{ velocity at step } i+1 \end{array} \tag{8}$$

We also extended the above example by adding a controller that is responsible for accelerating the car. By doing this we end up with a simplified cruise control system. The job of the controller is to maintain a certain velocity by either accelerating with a constant force, or applying no force at all. The safety property in this example is that the velocity is between $9.9\frac{m}{sec}$ and $10.1\frac{m}{sec}$ (additionally, we set the velocity in the initial state to 10, e.g. $v_{car_0} = 10$).

The results of our work with iSAT and Craig interpolation can be seen in Table 1. The test machine used for the results stated here has a Quadcore Intel Q9450 processor @ 2.66GHz.

The columns #**Decisions** and #**Deductions** show the number of decisions and deductions iSAT needs to solve the entire problem. The number of internal AIG-nodes needed to store the partial CIs is given in column #**Aig-Nodes**. The unroll depth where the FPC was successful is provided in column **Depth**. The last two columns provide information about the time needed for performing the FPC (**Time FPC**) and the overall time (**Time**). The examples presented

demonstrate that it is possible to successfully apply Craig interpolation in the case of systems containing non-linear behaviour, and the time needed to solve the FPC is negligible. Further the column #**Aig-Nodes** shows that the approximate size of the CIs stays relatively small in all cases. Concerning the Hènon Map where the property and the behaviour are very complex, the overall time is increasing as many CIs have to be computed until the FPC is successful, in this case the solving depth is 283. To further demonstrate the efficacy of Craig interpolation for iSAT we also considered the behaviour of "pure" iSAT on the benchmarks given above.In the case of the Hènon map iSAT obtains unsat for BMC unrolling depth $k < 158$ and terminates with a candidate solution at unroll depth 158. In contrast to this, using CIs in iSAT helps to prove the unsatisfiability for all k and thus prove safety. In order to find a fixed-point we had to modify certain parameters that influence the overall search procedure. It seems to be profitable to modify the iSAT variable decision heuristic to first decide Boolean simple bounds, next real simple bounds, and in the end iSAT is allowed to split certain intervals. Besides different decision heuristics, one can generate different CIs by changing the minimal splitting width. At the moment we do not take slackness information into account when computing CIs. Suppose you want to verify that a variable $x \geq 10$ cannot reach any negative value when divided again and again by 2. A possible CI representing those states that are reachable in one system step could look like $x \geq 5$. But you could also compute a CI with $x \geq 0$. If the later CI becomes the new initial state for the next iteration one could achieve the same CI again, and thus implies that we found a fixed-point. In order to compute such CIs we will have to take slackness information during resolution steps into account and influence the computation of CIs accordingly.

5 Conclusion

In this paper we introduced a method to generate CIs for formulae containing non-linear equations. Furthermore, we implemented our approach in the SMT solver iSAT, which is based on interval arithmetic embedded in a DPLL framework allowing it to reason about linear and non-linear constraints. To the best of our knowledge this is the first approach to compute CIs for arbitrary formulae containing Boolean combinations of linear and non-linear constraints. We showed that the CIs constructed can be used to verify safety properties, extending the work done by McMillan. Currently, we study heuristics to strengthen CIs for a given formula pair (A, B), e.g. by exploiting slackness between different constraints. We are also integrating a linear program solver (LP-solver) to combine LP-solving and iSAT's Craig interpolation and thus increase the performance of our solver for systems containing a large number of linear constraints.

References

1. Craig, W.: Linear reasoning: A new form of the Herbrand-Gentzen theorem. Journal of Symbolic Logic (3) (1957) 250–268

2. McMillan, K.L.: Interpolation and SAT-based Model Checking. In: 15th International Conference on Computer Aided Verification (CAV). (2003) 1–13
3. McMillan, K.L.: An interpolating theorem prover. Theor. Comput. Sci. (1) (2005)
4. Beyer, D., Henzinger, T.A., Jhala, R., Majumdar, R.: The software Model Checker Blast: Applications to Software Engineering. International Journal on Software Tools for Technology Transfer (STTT) (5-6) (2007) 505–525
5. Scholl, C., Disch, S., Pigorsch, F., Kupferschmid, S.: Using an SMT solver and Craig interpolation to detect and remove redundant linear constraints in representations of non-convex polyhedra. In: International Workshop on Satisfiability Modulo Theories. (2008) 18–26
6. Cimatti, A., Griggio, A., Sebastiani, R.: Efficient Generation of Craig Interpolants in Satisfiability modulo theories. CoRR (2009)
7. Fränzle, M., Herde, C., Ratschan, S., Schubert, T., Teige, T.: Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure. JSAT Special Issue on Constraint Programming and SAT (2007) 209–236
8. Matiyasevich, Y.V.: Enumerable sets are Diophantine. Soviet Mathematics. Doklady **11**(2) (1970) 354–358
9. Davis, M., Logemann, G., Loveland, D.: A Machine Program for Theorem Proving. CACM (1962) 394–397
10. Davis, M., Putnam, H.: A Computing Procedure for Quantification Theory. Journal of the ACM (3) (1960) 201–215
11. Benhamou, F., Granvilliers, L.: Continuous and Interval Constraints. In: Handbook of Constraint Programming. Foundations of Artificial Intelligence. (2006) 571–603
12. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic, Part 2 (1970) 115–125
13. Kupferschmid, S., Becker, B., Teige, T., Fränzle, M.: Proof certificates and non-linear arithmetic constraints. In: IEEE Design and Diagnostics of Electronic Circuits and Systems, IEEE (2011)
14. Yorsh, G., Musuvathi, M.: A Combination Method for Generating Interpolants. In: CADE. (2005) 353–368
15. Pudlàk, P.: Lower bounds for resolution and cutting planes proofs and monotone computations. In: J. of Symbolic Logic. (1995) 981–998
16. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. THEORETICAL COMPUTER SCIENCE **138** (1995) 3–34
17. May, R.M.: Simple Mathematical Models with very Complicated Dynamics. Nature (1976) 459
18. Hènon, M.: A two-dimensional mapping with a strange attractor. In: Communications in Mathematical Physics. (1976)

A Proof of the presented Construction Rules

For Craig interpolation, every formula pair (A, B) implies the following three sets of variables. The first set is G and contains all the variables occurring in both formulae, i.e. $G = sup(A) \cup sup(B)$. The second set is L_A that contains only variables located in the A-formula but not in the B-formula. The last set is L_B and contains all variables from the B-formula but not located in the A-formula.

Definition 3 (Construction Rules). *Let c be a clause containing only simple bounds. Then we denote the partial interpolant of clause c by $pi(c)$.*

$$pi(c) = \begin{cases} \bigvee\limits_{l_j \in c, var(l_j) \in G} l_j & : \ (c \in A) \ or \ (\exists c' \in A : c' \rhd c) \\ pi(c_1) \wedge pi(c_2) & : \ res(c_1, c_2, p) \vdash c \ and \ p \in G \cup L_B \\ pi(c_1) \vee pi(c_2) & : \ res(c_1, c_2, p) \vdash c \ and \ p \in L_A \\ true & : \ else \end{cases}$$

Definition 4 (Projection). *Let Θ be a disjunction of simple bound literals $\Theta = \bigvee\limits_{j=1}^{k} l_j$ with $var(l_j) \in G \cup L_A \cup L_B$. Then, $\Theta \mid_A$ is the projection of Θ to L_A and $\Theta \mid_B$ is the projection of Θ to $L_B \cup G$:*

$$\Theta \mid_A = \bigvee_{var(l_j) \in L_A}^{k} l_j \qquad\qquad \Theta \mid_B = \bigvee_{var(l_j) \in G \cup L_B}^{k} l_j$$

Lemma 5. *Let $F = A \wedge B$ be a CNF formula that is unsatisfiable and let P be a proof of the unsatisfiability. Then for every internal proof node c, the partial interpolant $pi(c)$ is a Craig interpolant of the formula pair (A', B') with $A' = A \wedge \neg(c \mid_A)$ and $B' = B \wedge \neg(c \mid_B)$.*

Proof. It is important to know that the proof tree P was computed by iSAT during conflict analysis. iSAT's assignments are conjunctions of simple bounds. If such an assignment leads to a conflict, the conflict is resolved by resolving simple bounds that are either: contained in clauses containing only simple bounds as literals; or these simple bounds have been derived by the deduction rule. In both cases it is guaranteed that all clauses involved during a resolution step contain only simple bounds. Another technical detail is that the deduction rule is only applied on clauses containing at least one literal that is not a simple bound (e.g. an arithmetic predicate). For the proof of Lemma 4 we may assume that every problem clause contains only literals that are consistent under the initial assignment. This is not a restriction as clauses containing literals that are simplified to *true* under the initial assignment can just be removed. In the case that a clause contains a literal that evaluates to $false$ we can remove this literal. In the case that the empty clause $\perp$ is located in A (B), the formula $false$ ($true$) is a valid Craig interpolant for the formula pair (A, B).

In the following we prove by induction over the *depth* of the proof tree P that a partial interpolant $pi(c)$ that is computed for a poof node c (e.g. clauses) of P is a valid Craig interpolant in the sense of Lemma 4. To do so we have to show three properties:

1. For every literal l of $pi(c)$ it holds that $var(l) \in G$
2. $A' \Rightarrow pi(c)$ (or $A' \wedge \neg pi(c)$ is unsatisfiable)
3. $pi(c) \Rightarrow \neg B'$ (or $pi(c) \wedge B'$ is unsatisfiable)

The construction rules (Definition 2) directly imply that every literal l occurring in any partial interpolant is a simple bound and in addtition $var(l) \in G$ is valid.

1. Base case:
   For every clause c with $depth = 0$ we distinguish between three different cases. In the first case c belongs to the clause set A and contains only simple bounds as literals. Following the construction rules the partial interpolant $pi(c)$ of c is computed by just removing every simple bound literal l from c with $var(l) \notin G$. Here A' is defined as $A \wedge \neg(c\ |_A)$. Formula $\neg(c\ |_A)$ is a (possibly empty) conjunction of unit clauses that contain literals l with $var(l) \in L_A$. Since in this case $c \in A'$ we can resolve every literal of c that contains a variable from L_A by iteratively applying a resolution with a unit clause of $\neg(c\ |_A)$. It is easy to see that the final resolvent implies $pi(c)$ and that is why $A' \wedge \neg pi(c)$ is unsatisfiable. Next we show the unsatisfiability of $B' \wedge pi(c)$:
   $$B' \wedge pi(c) \equiv$$
   $$B \wedge \neg(c\ |_B) \wedge pi(c)$$
   Here, clause c does not contain a literal l with $var(l) \in L_B$. That is why $\neg(c\ |_B) \equiv \neg pi(c)$ is valid and so we can conclude that formula $B' \wedge pi(c)$ is unsatisfiable.
   The second case is: $c \in B$ and c contains only simple bounds. Here, the construction rules define $pi(c) = true$. It is trivial to see that $A' \Rightarrow pi(c)$ holds. As $c \in B'$ and B' contains $\neg(c\ |_B)$ (in this case this is the negation of c) it follows that $pi(c) \wedge B'$ is unsatisfiable.
   In the third case c contains a literal l that is not a simple bound. Clause c must contain an arithmetic predicate (e.g. $x+y < z$). The construction rules do not cover this case and that is why $pi(c)$ is undefined. Anyway, if there is a path from clause c to the empty clause in the proof tree P it must be the case that the deduction rule has been applied on c to derive some new clause c' that contains only simple bounds. For this reason it suffices to prove that every partial interpolant $pi(c')$ for clause c' fulfills Lemma 4. Following the construction rules in this case, and the fact that either $A \Rightarrow c'$ or $B \Rightarrow c'$ is valid (as c' is derived by deduction from either $c \in A$ or $c \in B$) we can apply the same proof as shown above.
2. Induction hypothesis is that every clause c with $depth = n$ that contains only simple bounds has a valid partial interpolant $pi(c)$ by applying the presented construction rules.

3. The induction step is proved by showing that for every partial interpolant $pi(c_{res})$ for any clause c_{res} derived by applying resolution on arbitrary clauses c_1 and c_2 Lemma 4 holds. The hypothesis implies that $pi(c_1)$ and $pi(c_2)$ are valid Craig interpolants in the sense of Lemma 4. As said in the beginning of this proof we do not have to show that every clause c with $depth > 1$ and derived by deduction has a valid partial interpolant $pi(c)$ as such clauses (e.g. proof nodes) will never appear when iSAT computes a proof tree P. Therefore, we just distinguish between two cases. In the first case, the resolved simple bound contains a variable from the set L_A. First we show $A' \Rightarrow pi(c_{res})$ is valid. Here, $pi(c_{res})$ is defined as follows: $pi(c_{res}) = pi(c_1) \vee pi(c_2)$. The hypothesis says:
(a) $\models A \wedge \neg(c_1 \mid_A) \Rightarrow pi(c_1)$
(b) $\models A \wedge \neg(c_2 \mid_A) \Rightarrow pi(c_2)$
As c_{res} is the resolvent of c_1 and c_2 we can easily conclude the following (it is important to know that the simple bounds that are resolved contain a variable from L_A):

$$\neg(c_{res} \mid_A) \Rightarrow \neg(c_1 \mid_A) \vee \neg(c_2 \mid_A)$$

Next we rewrite the formula A':

$$A' \equiv A \wedge \neg(c_{res} \mid_A)$$

Putting everything together we conclude:

$$A' \Rightarrow A \wedge (\neg(c_1 \mid_A) \vee \neg(c_2 \mid_A)) \equiv$$

$$(A \wedge \neg(c_1 \mid_A)) \vee (A \wedge \neg(c_2 \mid_A))$$

Using the hypothesis we derive:

$$A' \Rightarrow pi(c_1) \vee pi(c_2)$$

In a very similar way we can prove that $pi(c_{res}) \wedge B'$ is unsatisfiable. The hypothesis implies the following two unsatisfiable formulae:
(a) $B \wedge \neg(c_1 \mid_B) \wedge pi(c_1)$
(b) $B \wedge \neg(c_2 \mid_B) \wedge pi(c_2)$
As the resolved literals contain a variable from L_A, it is easy to see that the following formula is valid:

$$\models \neg(c_1 \mid_B) \wedge \neg(c_2 \mid_B) \equiv \neg(c_{res} \mid_B)$$

Next we rewrite formula $B' \wedge pi(c_{res})$:

$$B' \wedge pi(c_{res}) \equiv$$

$$B \wedge \neg(c_{res} \mid_B) \wedge pi(c_{res}) \equiv$$

$$B \wedge \neg(c_1 \mid_B) \wedge \neg(c_2 \mid_B) \wedge pi(c_{res}) \equiv$$

$$B \wedge \neg(c_1 \mid_B) \wedge \neg(c_2 \mid_B) \wedge (pi(c_1) \vee pi(c_2)) \equiv$$

$$(B \wedge \neg(c_1 \mid_B) \wedge pi(c_1) \wedge \neg(c_2 \mid_B)) \vee (B \wedge \neg(c_2 \mid_B) \wedge pi(c_2) \wedge \neg(c_1 \mid_B))$$

The last obtained formula is unsatisfiable as it is the disjunction of two unsatisfiable formulae (known from the hypothesis in this case).
Lastly we consider the case with the following preconditions: c_{res} is derived by resolution from c_1 and c_2 and the resolved simple bounds contain a variable $v \in G \cup L_B$. Together with the definition of operator $\mid_A$ ($\mid_B$) and $v \in G \cup L_B$ we conclude:
(a) $\models \neg(c_{res} \mid_A) \equiv \neg(c_1 \mid_A) \wedge \neg(c_2 \mid_A)$
(b) $\models \neg(c_{res} \mid_B) \Rightarrow \neg(c_1 \mid_B) \vee \neg(c_2 \mid_B)$
In this case the construction rules define $pi(c_{res}) = pi(c_1) \wedge pi(c_2)$. First we show that formula $A' \wedge (\neg pi(c_{res}))$ is unsatisfiable:

$$A' \wedge (\neg pi(c_{res})) \equiv$$

$$A \wedge \neg(c_{res} \mid_A) \wedge (\neg pi(c_{res})) \equiv$$

$$A \wedge \neg(c_1 \mid_A) \wedge \neg(c_2 \mid_A) \wedge (\neg pi(c_{res})) \equiv$$

$$A \wedge \neg(c_1 \mid_A) \wedge \neg(c_2 \mid_A) \wedge (\neg(pi(c_1) \wedge pi(c_2))) \equiv$$

$$(A \wedge \neg(c_1 \mid_A) \wedge \neg(c_2 \mid_A) \wedge \neg(pi(c_1))) \vee (A \wedge \neg(c_1 \mid_A) \wedge \neg(c_2 \mid_A) \wedge \neg(pi(c_2)))$$

This formula is unsatisfiable as the hypothesis in this case includes that the following two formulae are unsatisfiable:
(a) $A \wedge \neg(c_1 \mid_A) \wedge \neg(pi(c_1))$
(b) $A \wedge \neg(c_2 \mid_A) \wedge \neg(pi(c_2))$
Next we show that formula $B' \wedge pi(c_{res})$ is also unsatisfiable. This is a direct implication from the following facts:
(a) $\models \neg(c_{res} \mid_B) \Rightarrow (\neg(c_1 \mid_B) \vee \neg(c_2 \mid_B))$
(b) $B \wedge \neg(c_1 \mid_B) \wedge pi(c_1)$ is unsatisfiable (entailed in the hypothesis)
(c) $B \wedge \neg(c_2 \mid_B) \wedge pi(c_2)$ is unsatisfiable (entailed in the hypothesis)

$$B' \wedge pi(c_{res}) \equiv$$

$$B \wedge \neg(c_{rec} \mid_B) \wedge pi(c_{res}) \equiv$$

$$B \wedge \neg(c_{res}) \mid_B) \wedge pi(c_1) \wedge pi(c_2)$$

The last obtained formula can be overapproximated by replacing $(\neg c_{res})$ with $(\neg(c_1 \mid_B) \vee \neg(c_2 \mid_B))$. We obtain:

$$B \wedge (\neg(c_1 \mid_B) \vee \neg(c_2 \mid_B)) \wedge pi(c_1) \wedge pi(c_2) \equiv$$

$$(B \wedge \neg(c_1 \mid_B) \wedge pi(c_1) \wedge pi(c_2)) \vee (B \wedge \neg(c_2 \mid_B) \wedge pi(c_2) \wedge pi(c_1))$$

We derived an unsatisfiable formula. □