Interpolant tree automata and their application in Horn clause verification

Kafle, Bishoksan; Gallagher, John Patrick

Published in:
Electronic Proceedings in Theoretical Computer Science

DOI:
10.4204/EPTCS.216.6

Publication date:
2016

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (APA):
Kafle, B., & Gallagher, J. P. (2016). Interpolant tree automata and their application in Horn clause verification. *Electronic Proceedings in Theoretical Computer Science*, *216*, 104-117. https://doi.org/10.4204/EPTCS.216.6

Interpolant Tree Automata and their Application in Horn Clause Verification *

Bishoksan Kafle
Roskilde University, Denmark
kafle@ruc.dk

John P. Gallagher
Roskilde University, Denmark
IMDEA Software Institute, Spain
jpg@ruc.dk

This paper investigates the combination of abstract interpretation over the domain of convex polyhedra with interpolant tree automata, in an abstraction-refinement scheme for Horn clause verification. These techniques have been previously applied separately, but are combined in a new way in this paper. The role of an interpolant tree automaton is to provide a generalisation of a spurious counterexample during refinement, capturing a possibly infinite set of spurious counterexample traces. In our approach these traces are then eliminated using a transformation of the Horn clauses. We compare this approach with two other methods; one of them uses interpolant tree automata in an algorithm for trace abstraction and refinement, while the other uses abstract interpretation over the domain of convex polyhedra without the generalisation step. Evaluation of the results of experiments on a number of Horn clause verification problems indicates that the combination of interpolant tree automaton with abstract interpretation gives some increase in the power of the verification tool, while sometimes incurring a performance overhead.
Keywords: Interpolant tree automata, Horn clauses, abstraction-refinement, trace abstraction.

1 Introduction

In this paper we combine two existing techniques, namely abstract interpretation over the domain of convex polyhedra and interpolant tree automata in a new way for Horn clause verification. Abstract interpretation is a scalable program analysis technique which computes invariants allowing many program properties to be proven, but suffers from false alarms; safe but not provably safe programs may be indistinguishable from unsafe programs. Refinement is considered in this case. In previous work [27] we described an *abstraction-refinement* scheme for Horn clause verification using abstract interpretation and refinement with finite tree automata. In that approach refinement eliminates a single spurious counterexample in each iteration of the abstraction-refinement loop, using a clause transformation based on a tree automata difference operation. In contrast to that work, we apply the method of Wang and Jiao [33] for constructing an *interpolant tree automaton* from an infeasible trace. This generalises the trace of a spurious counterexample, recognising a possibly infinite number of spurious counterexamples, which can then be eliminated in one iteration of the abstraction-refinement loop. We combine this construction in the framework of [27]. The experimental results on a set of Horn clause verification problems are reported, and compared with both [27] and the results of Wang and Jiao [33] using trace abstraction and refinement.

In Section 2 we introduce the key concepts of constrained Horn clauses and finite tree automata. Section 3 contains the definitions of interpolants and the construction of a tree interpolant automaton

*The research leading to these results has been supported by the EU FP7 project 318337, *ENTRA - Whole-Systems Energy Transparency* and the EU FP7 project 611004, coordination and support action ICT-Energy.

G.W. Hamilton, A. Lisitsa, A.P. Nemytykh (Eds): VPT 2016
EPTCS 216, 2016, pp. 104–117, doi:10.4204/EPTCS.216.6

```
c1. fib(A, B):- A>=0, A=<1, B=1.
c2. fib(A, B) :- A>1, A2=A-2,
            A1=A-1, B=B1+B2,  fib(A1,B1), fib(A2,B2).
c3. false:- A>5, B<A, fib(A,B).
```

Figure 1: Example CHCs (Fib) defining a Fibonacci function.

following the techniques of Wang and Jiao [33]. In Section 4 we describe our algorithm combining abstract interpretation with tree interpolant automata, including in Section 4.1 an experimental evaluation and comparison with other approaches. Finally in Section 5 we discuss related work.

2 Preliminaries

A constrained Horn clause (CHC) is a first order predicate logic formula of the form $\forall(\phi \wedge p_1(X_1) \wedge \ldots \wedge p_k(X_k) \rightarrow p(X))$ $(k \geq 0)$, where ϕ is a first order logic formula (constraint) with respect to some background theory and $p_1, \ldots, p_k, p$ are predicate symbols. We assume (wlog) that X_i, X are (possibly empty) tuples of distinct variables and ϕ is expressed in terms of X_i, X, which can be achieved by adding equalities to ϕ. $p(X)$ is the head of the clause and $\phi \wedge p_1(X_1) \wedge \ldots \wedge p_k(X_k)$ is the body. There is a distinguished predicate symbol *false* which is interpreted as false. Clauses whose head is *false* are called *integrity constraints*. Following the notation used in constraint logic programming a clause is usually written as $H \leftarrow \phi, B_1, \ldots, B_k$ where $H, B_1, ..., B_k$ stand for atomic formulas (atoms) $p(X), p_1(X_1), ..., p_k(X_k)$. A set of CHCs is sometimes called a (constraint logic) program.

An interpretation of a set of CHCs is represented as a set of *constrained facts* of the form $A \leftarrow \phi$ where A is an atom and ϕ is a formula with respect to some background theory. $A \leftarrow \phi$ represents a set of ground facts $A\theta$ such that $\phi\theta$ holds in the background theory (θ is called a grounding substitution). An interpretation that satisfies each clause in P is called a model of P. In some works [6, 28], a *model* is also called a *solution* and we use them interchangeably in this paper.

Horn clause verification problem. Given a set of CHCs P, the CHC verification problem is to check whether there exists a model of P. It can easily be shown that P has a model if and only if the fact *false* is not a consequence of P.

An example set of CHCs, encoding the Fibonacci function is shown in Figure 1. Since its derivations are trees, it serves as an interesting example from the point of view of *interpolant tree automata*.

Definition 1 (Finite tree automaton [7]) *An FTA $\mathscr{A}$ is a tuple (Q, Q_f, Σ, Δ), where Q is a finite set of states, $Q_f \subseteq Q$ is a set of final states, Σ is a set of function symbols, and Δ is a set of transitions of the form $f(q_1, \ldots, q_n) \rightarrow q$ with $q, q_1, \ldots, q_n \in Q$ and $f \in \Sigma$. We assume that Q and Σ are disjoint.*

We assume that each CHC in a program P is associated with an identifier by a mapping $\mathsf{id}_P : P \rightarrow \Sigma$. An identifier (an element of Σ) is a function symbol whose arity is the same as the number of atoms in the clause body. For instance a clause $p(X) \leftarrow \phi, p_1(X_1), \ldots, p_k(X_k)$ is assigned a function symbol with arity k. As will be seen later, these identifiers are used to build trees that represent *derivations* using the clauses. A set of derivation trees (traces) of a set of atoms of a program P can be abstracted and represented by an FTA. We provide such a construction in Definition 2.

Definition 2 (Trace FTA for a set of CHCs) *Let P be a set of CHCs. Define the trace FTA for P as $\mathscr{A}_P = (Q, Q_f, \Sigma, \Delta)$ where*

- $Q = \{p \mid p \text{ is a predicate symbol of } P\} \cup \{false\}$;
- $Q_f = \{false\}$;
- Σ *is a set of function symbols;*
- $\Delta = \{c_j(p_1,\ldots,p_k) \to p \mid \text{where } c_j \in \Sigma,\ p(X) \leftarrow \phi, p_1(X_1),\ldots,p_k(X_k) \in P,\ c_j = \mathsf{id}_P(p(X) \leftarrow \phi, p_1(X_1),\ldots,p_k(X_k))\}$.

The elements of $\mathscr{L}(\mathscr{A}_P)$ are called trace-terms or trace-trees or simply traces of P rooted at *false*.

Example 1 *Let Fib be the set of CHCs in Figure 1. Let* id_{Fib} *map the clauses to identifiers* c_1, c_2, c_3 *respectively. Then* $\mathscr{A}_{Fib} = (Q, Q_f, \Sigma, \Delta)$ *where:*

$$\begin{array}{lcl} Q & = & \{\mathtt{fib}, \mathtt{false}\} \\ Q_f & = & \{\mathtt{false}\} \\ \Sigma & = & \{c_1, c_2, c_3\} \\ \Delta & = & \{c_1 \to \mathtt{fib},\ c_2(\mathtt{fib}, \mathtt{fib}) \to \mathtt{fib}, \\ & & c_3(\mathtt{fib}) \to \mathtt{false}\} \end{array}$$

Similarly, we can also construct an FTA representing a single trace. It should be noted that the whole idea of representing program traces by FTAs is to use automata theoretic operations for dealing with program traces, for example, removal of an undesirable trace from a set of program traces. Let P be a set of CHCs and let $t \in \mathscr{L}(\mathscr{A}_P)$. There exists an FTA $\mathscr{A}_t$ such that $\mathscr{L}(\mathscr{A}_t) = \{t\}$. We illustrate the construction with an example.

Example 2 (Trace FTA) *Consider the FTA in Example 1. Let* $t = c_3(c_2(c_1, c_1)) \in \mathscr{L}(\mathscr{A}_P)$*. Then* $\mathscr{A}_t = (Q, Q_f, \Sigma, \Delta)$ *is defined as:*

$$\begin{array}{lcl} Q & = & \{\mathtt{e}_1, \mathtt{e}_2, \mathtt{e}_3, \mathtt{e}_4\} \\ Q_f & = & \{\mathtt{e}_1\} \\ \Sigma & = & \{c_1, c_2, c_3, c_4\} \\ \Delta & = & \{c_1 \to \mathtt{e}_3,\ c_1 \to \mathtt{e}_4,\ c_2(\mathtt{e}_3, \mathtt{e}_4) \to \mathtt{e}_2, \\ & & c_3(\mathtt{e}_2) \to \mathtt{e}_1\} \end{array}$$

where Σ *is the same as in* $\mathscr{A}_P$ *and the states* $\mathtt{e_i}$ $(i = 1 \ldots 4)$ *represent the nodes in the trace-tree, with root node* $\mathtt{e}_1$ *as the final state.*

A trace-term is a representation of a *derivation trees*, called an AND-tree [32, 13] giving a proof of an atomic formula from a set of CHCs.

Definition 3 (AND-tree for a trace term $T(t)$ (adapted from [27])) *Let P be a set of CHCs and let $t \in \mathscr{L}(\mathscr{A}_P)$. An AND-tree corresponding to t, denote by $T(t)$, is the following labelled tree, where each node of $T(t)$ is labelled by an atom, a clause identifier and a constraint.*

1. *For each sub-term $c_j(t_1,\ldots,t_k)$ of t there is a corresponding node in $T(t)$ labelled by an atom $p(X)$, an identifier c_j such that $c_j = \mathsf{id}_P(p(X) \leftarrow \phi, p_1(X_1),\ldots,p_k(X_k))$, and a constraint ϕ; the node's children (if $k > 0$) are the nodes corresponding to $t_1,\ldots,t_k$ and are labelled by $p_1(X_1),\ldots,p_k(X_k)$.*
2. *The variables in the labels are chosen such that if a node n is labelled by a clause, the local variables in the clause body do not occur outside the subtree rooted at n.*

We assume that each node in $T(t)$ is uniquely identified by a natural number. We omit t from $T(t)$ when it is clear from the context.

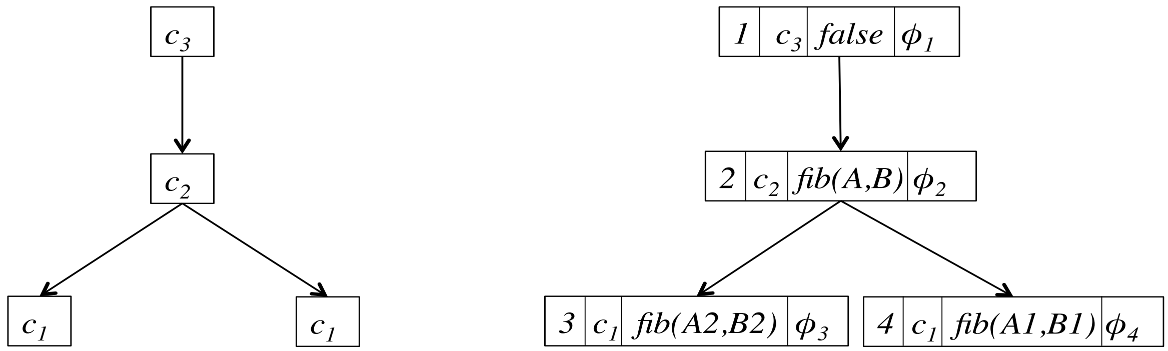

Figure 2: A trace-term $c_3(c_2(c_1,c_1))$ of Fib (left) and its AND-tree (right), where $\phi_1 \equiv \mathtt{A} > 5 \wedge \mathtt{B} < \mathtt{A}$; $\phi_2 \equiv \mathtt{A} > 1 \wedge \mathtt{A2} = \mathtt{A} - 2 \wedge \mathtt{A1} = \mathtt{A} - 1 \wedge \mathtt{B} = \mathtt{B1} + \mathtt{B2}$; $\phi_3 \equiv \mathtt{A2} \geq 0 \wedge \mathtt{A2} \leq 1 \wedge \mathtt{B2} = 1$; $\phi_4 \equiv \mathtt{A1} \geq 0 \wedge \mathtt{A1} \leq 1 \wedge \mathtt{B1} = 1$.

The formula represented by an AND-tree T, represented by $F(T)$ is

1. ϕ, if T is a single leaf node labelled by a constraint ϕ; or
2. $\phi \wedge \bigwedge_{i=1..n}(F(T_i))$ if the root node of T is labelled by the constraint ϕ and has subtrees $T_1,\ldots,T_n$.

The formula $F(T)$ where T is the AND-tree in Figure 2 is

$$\begin{array}{l}\mathtt{A} > 5 \wedge \mathtt{B} < \mathtt{A} \wedge \mathtt{A} > 1 \wedge \mathtt{A2} = \mathtt{A} - 2 \wedge \mathtt{A1} = \mathtt{A} - 1 \wedge \mathtt{B} = \mathtt{B1} + \mathtt{B2} \\ \mathtt{A2} \geq 0 \wedge \mathtt{A2} \leq 1 \wedge \mathtt{B2} = 1 \wedge \mathtt{A1} \geq 0 \wedge \mathtt{A1} \leq 1 \wedge \mathtt{B1} = 1\end{array}$$

We say that an AND-tree T is satisfiable or feasible if $F(T)$ is satisfiable, otherwise unsatisfiable or infeasible. Similarly, we say a trace-term is satisfiable (unsatisfiable) iff its corresponding AND-tree is satisfiable (unsatisfiable). The trace-term $c_3(c_2(c_1,c_1))$ in Figure 2 is unsatisfiable since $F(c_3(c_2(c_1,c_1)))$ is unsatisfiable.

3 Interpolant tree automata

Refinement of trace abstraction is an approach to program verification [19]. In this approach, if a property is not provable in an abstraction of program traces then an abstract trace showing the violation of the property is emitted. If such a trace is not feasible with respect to the original program, it is eliminated from the trace abstraction which is viewed as a refinement of the trace abstraction. The notion of interpolant automata [19] allows one to generalise an infeasible trace to capture possibly infinitely many infeasible traces which can then be eliminated in one refinement step. In this section, we revisit the construction of an *interpolant tree automaton* [33] from an infeasible trace-tree. The automaton serves as a generalisation of the trace-tree; and we apply this construction in Horn clause verification.

Definition 4 ((Craig) Interpolant [10]) *Given two formulas ϕ_1, ϕ_2 such that $\phi_1 \wedge \phi_2$ is unsatisfiable, a (Craig) interpolant is a formula I with (1) $\phi_1 \rightarrow I$; (2) $I \wedge \phi_2 \rightarrow$* false*; and (3) vars(I) $\subseteq$ vars(ϕ_1) $\cap$ vars(ϕ_2). An interpolant of ϕ_1 and ϕ_2 is represented by $I(\phi_1,\phi_2)$.*

The existence of an interpolant implies that $\phi_1 \wedge \phi_2$ is unsatisfiable [29]. Similarly, if the background theory underlying the CHCs P admits (Craig) interpolation [10], then every infeasible derivation using the clauses in P has an interpolant [28].

Example 3 (Interpolant example) *Let* $\phi_1 \equiv \mathtt{A2} \leq 1 \wedge \mathtt{A} > 1 \wedge \mathtt{A2} = \mathtt{A} - 2 \wedge \mathtt{A1} = \mathtt{A} - 1 \wedge \mathtt{B} = \mathtt{B1} + \mathtt{B2}$ *and* $\phi_2 \equiv \mathtt{A} > 5 \wedge \mathtt{B} < \mathtt{A}$ *such that* $\phi_1 \wedge \phi_2$ *is unsatisfiable. Since the formula* $I \equiv A \leq 3$ *fulfills all the conditions of Definition 4, it is an interpolant of* ϕ_1 *and* ϕ_2.

Given a node i in an AND-tree T, we call T_i the sub-tree rooted at i, ϕ_i the formula label of node i, $F(T_i)$ the formula of the sub-tree rooted at node i and $G(T_i)$, the formula $F(T)$ except the formula $F(T_i)$, which is defined as follows:

1. *true*, if T is a single leaf node labelled by constraint ϕ and the node is i; or
2. ϕ, if T is a single leaf node labelled by constraint ϕ and the node is different from i; or
3. *true*, if the root node of T is labelled by the constraint ϕ and the node is i; or
4. $\phi \wedge \bigwedge_{l=1..n}(G(T_l))$ if the root node of T is labelled by the constraint ϕ, and the node is different from i and has subtrees $T_1, \ldots, T_n$.

Definition 5 (Tree Interpolant of an AND-tree ([33])) *Let T be an infeasible AND-tree. A tree interpolant $TI(T)$ for T is a tree constructed as follows:*

1. *The root node i of $TI(T)$ is labelled by i, the atom of the node i of T and the formula false;*
2. *Each leaf node i of $TI(T)$ is labelled by i, the atom of the node i of T and by $I(F(T_i), G(T_i))$;*
3. *Let i be any other node of T. We define F_1 as $(\phi_i \wedge \bigwedge_{k=1}^{n} I_k)$ where $\bigwedge_{k=1}^{n} I_k$ ($n \geq 1$) is the conjunction of formulas representing the interpolants of the children of the node i in $TI(T)$. Then the node i of $TI(T)$ is labelled by i, the atom of the node i of T and the formula $I(F_1, G(T_i))$.*

The *tree interpolant* corresponding to *AND tree* in Figure 2(b) is shown in Figure 3(b).

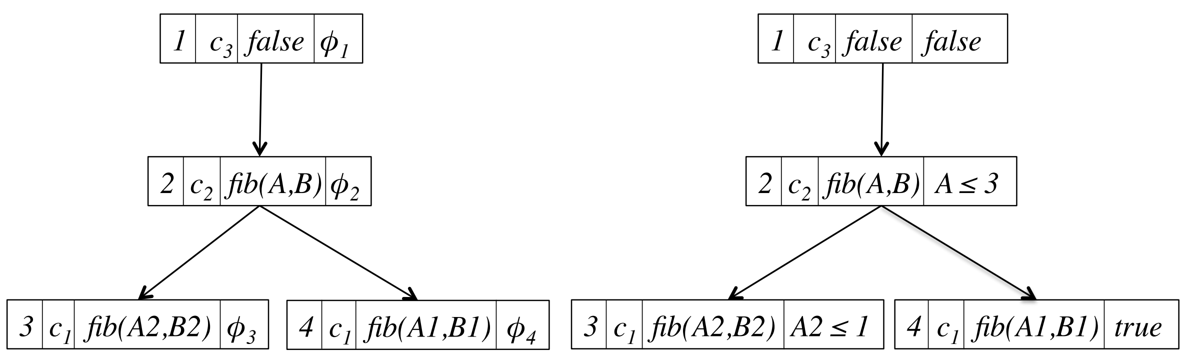

Figure 3: *AND tree* of Figure 2 (left) and its *tree interpolant* (right). Let I_j represents an interpolant of the node j. Then $I_1 \equiv false$; $I_4 \equiv I(\phi_4, \phi_3 \wedge \phi_1 \wedge \phi_2)$; $I_3 \equiv I(\phi_3, \phi_1 \wedge \phi_2 \wedge I_4)$; $I_2 \equiv I(I_3 \wedge I_4 \wedge \phi_2, \phi_1)$.

Since there is a one-one correspondence between an AND-tree and a trace-term, we can define a tree interpolant for a trace-term as follows:

Definition 6 (Tree Interpolant of a trace-term $TI(t)$) *Given an infeasible trace-term t, its tree interpolant, represented as $TI(t)$, is the tree interpolant of its corresponding AND-tree.*

Definition 7 (Interpolant mapping Π_{TI}) *Given a tree interpolant TI for some tree, Π_{TI} is a mapping from the atom labels and node numbers of each node in TI to the formula label such that $\Pi_{TI}(A^j) = \psi$ where A is the atom label and ψ is the formula label at node j.*

For our example program Π_{TI} is the following:

$$\{false^1 \mapsto false, fib^2(A,B) \mapsto A \leq 3, fib^3(A2,B2) \mapsto A \leq 1, fib^4(A1,B1) \mapsto true\}$$

Property 1 (Tree interpolant property) *Let $TI(T)$ be a tree interpolant for some infeasible* AND*-tree T. Then*

1. $\Pi_{TI}(r^i) = false$ *where r is the atom label of the root of $TI(T)$;*
2. *for each node j with children $j_1, ..., j_n$ ($n \geq 0$) the following property holds:* $(\bigwedge_{k=0}^{n} \Pi_{TI}(A^{j_k})) \wedge \phi_j \rightarrow \Pi_{TI}(A^j)$ *where ϕ_j is the formula label of the node j of T;*
3. *for each node j the following property holds:* $vars(\Pi_{TI}(A^j)) \subseteq (vars(F(T_j)) \cap vars(G(T_j)))$*, where the formula $F(T_j)$ and $G(T_j)$ corresponds to T.*

Definition 8 (Interpolant tree automaton for Horn clauses $\mathscr{A}_t^I$=(Q, Q_f, Σ, Δ) [33]) *Let P be a set of CHCs, $t \in \mathscr{L}(\mathscr{A}_P)$ be any infeasible trace-term and $TI(t)$ be a tree interpolant of t. Let $\sigma : As \times J \rightarrow Q$ where σ maps an atom at node $i \in J$ of $TI(t)$ to an FTA state in Q. Define $\rho : Pred^J \rightarrow Pred$ which maps a predicate name with superscript to a predicate name of P. Then the interpolant automaton of t is defined as an FTA $\mathscr{A}_t^I$ such that*

- $Q = \{\sigma(A,i) : A \text{ is the atom label of the node } i \text{ of } TI(t)\}$;
- $F = \{\sigma(A,i) : A \text{ is the atom label of the root of } TI(t)\}$;
- Σ *is a set of function symbols of P;*
- $\Delta = \{c(p_1^{j_1}, \ldots, p_k^{j_k}) \rightarrow p^j \mid cl = p(X) \leftarrow \phi, p_1(X_1), \ldots, p_k(X_k) \in P,\ c = \mathsf{id}_P(cl),\ \rho(p^i) = p,$ $\rho(p_m^i) = p_m \text{ for } m = 1..k \text{ and } \Pi_{TI}(p^j)(X) \leftarrow \phi, \Pi_{TI}(p_1^{j_1})(X_1), \ldots, \Pi_{TI}(p_k^{j_k})(X_k)\}$.

Example 4 (Interpolant automata for $c_3(c_2(c_1,c_1))$)

$$\begin{array}{lcl}
Q & = & \{\mathtt{fib}^2, \mathtt{fib}^3, \mathtt{fib}^4, \mathtt{error}\} \\
Q_f & = & \{\mathtt{error}\} \\
\Sigma & = & \{c_1, c_2, c_3\} \\
\Delta & = & \{c_1 \rightarrow \mathtt{fib}^2, c_1 \rightarrow \mathtt{fib}^3, c_1 \rightarrow \mathtt{fib}^4, \\
& & c_2(\mathtt{fib}^2,\mathtt{fib}^2) \rightarrow \mathtt{fib}^4, c_2(\mathtt{fib}^2,\mathtt{fib}^3) \rightarrow \mathtt{fib}^2, \\
& & c_2(\mathtt{fib}^2,\mathtt{fib}^3) \rightarrow \mathtt{fib}^4, c_2(\mathtt{fib}^2,\mathtt{fib}^4) \rightarrow \mathtt{fib}^4, \\
& & c_2(\mathtt{fib}^3,\mathtt{fib}^2) \rightarrow \mathtt{fib}^2, c_2(\mathtt{fib}^3,\mathtt{fib}^2) \rightarrow \mathtt{fib}^4, \\
& & c_2(\mathtt{fib}^3,\mathtt{fib}^3) \rightarrow \mathtt{fib}^2, c_2(\mathtt{fib}^3,\mathtt{fib}^3) \rightarrow \mathtt{fib}^4, \\
& & c_2(\mathtt{fib}^3,\mathtt{fib}^4) \rightarrow \mathtt{fib}^2, c_2(\mathtt{fib}^3,\mathtt{fib}^4) \rightarrow \mathtt{fib}^4, \\
& & c_2(\mathtt{fib}^4,\mathtt{fib}^2) \rightarrow \mathtt{fib}^4, c_2(\mathtt{fib}^4,\mathtt{fib}^3) \rightarrow \mathtt{fib}^2, \\
& & c_2(\mathtt{fib}^4,\mathtt{fib}^3) \rightarrow \mathtt{fib}^4, c_2(\mathtt{fib}^4,\mathtt{fib}^4) \rightarrow \mathtt{fib}^4, \\
& & c_3(\mathtt{fib}^2) \rightarrow \mathtt{error}, c_3(\mathtt{fib}^3) \rightarrow \mathtt{error}\}
\end{array}$$

The construction described in Definition 8 recognizes only infeasible traces terms of P as stated in Theorem 1.

Theorem 1 (Soundness) *Let P be a set of CHCs and $t \in \mathscr{L}(\mathscr{A}_P)$ be any infeasible trace-term. Then the interpolant automaton $\mathscr{A}_t^I$ recognises only infeasible trace-terms of P.*

Definition 9 (Conjunctive interpolant mapping) *Given an interpolant mapping Π_{TI} of a tree interpolant TI, we define a conjunctive interpolant mapping for an atom label A of any node in TI, represented as $\Pi^c_{TI}(A)$, to be the following formula $\Pi^c_{TI}(A) = \bigwedge_j \Pi_{TI}(A^j)$, where j ranges over the nodes of TI. It is the conjunction of interpolants of all the nodes of TI with atom label A. The conjunctive interpolant mapping of TI is represented is $\Pi^c_{TI} = \{\Pi^c_{TI}(A) \mid A \; is \; the \; atom \; label \; of \; TI\}$.*

It is desirable that the *interpolant tree automaton* of a trace $t \in \mathscr{L}(\mathscr{A}_P)$ recognizes as many infeasible traces as possible, in an ideal situation, all infeasible traces of P. This is possible under the condition described in Theorem 2.

Theorem 2 (Model and Interpolant Automata) *Let $t \in \mathscr{L}(\mathscr{A}_P)$ be any infeasible trace-term. If $\Pi^c_{TI(t)}$ is a model of P, then the interpolant automaton of t recognises all infeasible trace-terms of P.*

4 Application to Horn clause verification

An *abstraction-refinement* scheme for Horn clause verification is described in [27] which is depicted in Figure 4. In this, a set of CHCs P is analysed using the techniques of *abstract interpretation* over the domain of convex polyhedra which produces an over-approximation M of the minimal model of P. The set of traces used during the analysis can be captured by an FTA $\mathscr{A}^M_P$ (see Definition 10). This automaton recognizes all trace-terms of P except some infeasible ones. Some of the infeasible trace-terms are removed by the abstract interpretation. P is solved or safe (that is, it has a model) if $false \notin M$. If this is not the case, a trace-term $t \in \mathscr{L}(A^M_P)$ is selected and checked for feasibility. If the answer is positive, P has no model, that is, P is unsafe.

Otherwise t is considered spurious and this drives the refinement process. The refinement in [27] consists of constructing an automaton $\mathscr{A}'_P$ which recognizes all traces in $\mathscr{L}(\mathscr{A}^M_P) \setminus \mathscr{L}(\mathscr{A}_t)$ and generating a refined set of clauses from P and $\mathscr{A}'_P$. The automata difference construction refines a set of traces (abstraction), which induces refinement in the original program. The refined program is again fed to the abstract interpreter. This process continues until the problem is safe, unsafe or the resources are exhausted. We call this approach Refinement of Abstraction in Horn clauses using Finite Tree automata, RAHFT in short.

The approach just described lacks generalisation of spurious counterexamples during refinement. However, in our current approach, we generalise a spurious counterexample through the use of *interpolant automata*. Section 3 describes how to compute an *interpolant automaton* (taken from [33]) corresponding to an infeasible Horn clause derivation. We first construct an *interpolant automaton* viz. $\mathscr{A}^I_t$ corresponding to t. In Figure 4, this is shown by a blue line (in the middle) connecting the Abstraction and Refinement boxes. The refinement proceeds as in RAHFT with the only difference that $\mathscr{A}'_P$ now recognizes all traces in $\mathscr{L}(\mathscr{A}^M_P) \setminus \mathscr{L}(\mathscr{A}^I_t)$. We call this approach Refinement of Abstraction in Horn clauses using Interpolant Tree automata, RAHIT in short.

Next we briefly describe how to generate an FTA, $\mathscr{A}^M_P$, corresponding to a set of clauses P using the approximation produced by *abstract interpretation*. Finally we show some experimental results using our current approach on a set of Horn clause verification benchmarks.

Obtaining an FTA from a program and a model. Let M be a set of constrained atoms of the form $p(X) \leftarrow \phi$ where p is a program predicate and ϕ is a constraint over X. Given such an set M, define γ_M to be the mapping from atoms to constraints such that $\gamma_M(p(X)) = \phi$ for each constrained fact $p(X) \leftarrow \phi$.

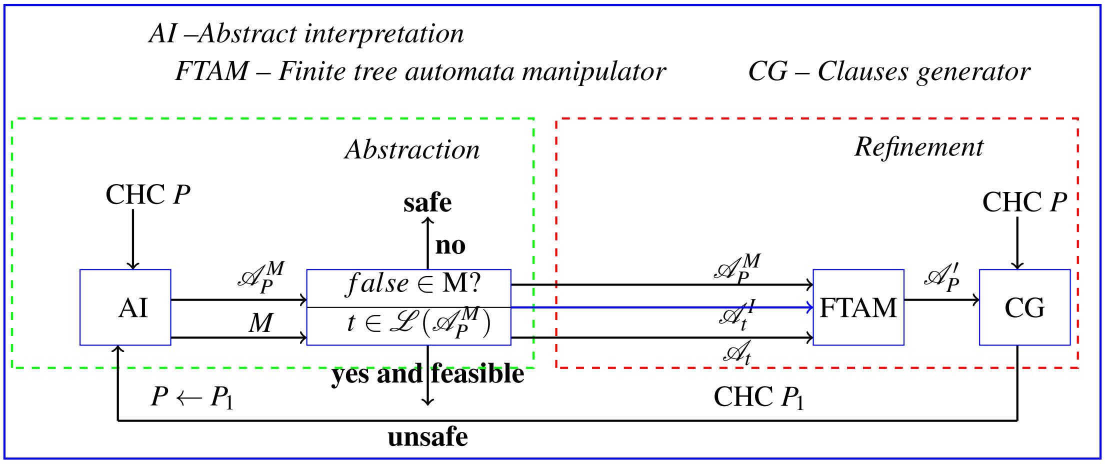

Figure 4: *Abstraction-refinement scheme in Horn clause verification [27]. M is an approximation produced as a result of abstract interpretation.* $\mathscr{A}_P'$ *recognizes all traces in* $\mathscr{L}(\mathscr{A}_P^M) \setminus \mathscr{L}(\mathscr{A}_t)$.

M is a model of P (called a *syntactic solution* in [33]) if for each clause $p(X) \leftarrow \phi, p_1(X_1), \ldots, p_n(X_n)$ in P, $\phi \wedge \bigwedge_{i=1}^n \gamma_M(p_i(X_i)) \rightarrow \gamma_M(p(X))$.

Given such an M, we construct an FTA corresponding to P, which is the same as $\mathscr{A}_P$ (Definition 2) except that transitions corresponding to clauses whose bodies are not satisfiable in the model are omitted, since they cannot contribute to feasible derivations.

Definition 10 (FTA defined by a model.) *Let P be a set of CHCs and M be a model defined by a set of constrained facts. Then the FTA* $\mathscr{A}_P^M = (Q, Q_f, \Sigma, \Delta_M)$ *where* Q, Q_f *and* Σ *are the same as for* $\mathscr{A}_P$ *(Definition 2) and* Δ_M *is the following set of transitions.*

$$\Delta_M = \{c(p_1, \ldots, p_n) \rightarrow p \mid \begin{array}{l} id_P(c) = p(X) \leftarrow \phi, p_1(X_1), \ldots, p_n(X_n), \\ \mathsf{SAT}(\phi \wedge \bigwedge_{i=1}^n \gamma_M(p_i(X_i))) \end{array}$$

Lemma 1 *Let P be a set of clauses and M be a model of P then* $\mathscr{L}(\mathscr{A}_P^M)$ *includes all feasible trace-terms of P rooted at false.*

In our experiments, the abstract interpretation was over the domain of convex polyhedra, yielding a set of constrained facts where each constraints is a conjunction of linear equalities and inequalities representing a convex polyhedron.

Example 5 (FTA produced as a result of *abstract interpretation*) *For our example program in Figure 1, the convex polyhedral abstraction produces an over-approximation M which is represented as*

$$M = \{fib(A, B) \leftarrow A >= 0, B >= 1, -A + B >= 0\}$$

Since there is no constrained fact for false in M, this is a model for the example program. Our abstraction-refinement approach terminates at this point. However for the purpose of example, we show the FTA constructed for the example program using M. Since the bodies of each clauses except the integrity constraint are satisfied under M, the FTA is same as the one depicted in Example 1 except the transition $c_3(\texttt{fib}) \rightarrow \texttt{false}$*, which is removed because of abstract interpretation.*

4.1 Experiments

For our experiment, we have collected a set of 68 programs from different sources.

1. A set of 30 programs from SV-COMP'15 repository[1] [3] (recursive category) and translated them to Horn clauses using inter-procedural encoding of SeaHorn [18, 17] producing (mostly) non-linear Horn clauses.
2. A set of 38 problems taken from the source repository[2], compiled by the authors of the tool Eldarica [23]. This set consists of problems, among others, from the NECLA static analysis suite, from the paper [25]. These tasks are also considered in [33] and are interpreted over *integer linear arithmetic*.

We made the following comparison between the tools.

1. We compare RAHIT with RAHFT, which compares the effect of removing a set of traces rather than a single trace.
2. We compare RAHIT with the *trace-abstraction* tool [33] (*TAR* from now on). RAHIT uses polyhedral approximation combined with trace abstraction refinement whereas *TAR* uses only trace abstraction refinement.

The results are summarized in Table 1.

Implementation: Most of the tools in our tool-chain depicted in Figure 4 are implemented in Ciao Prolog [22] except the one for determinisation of FTA, which is implemented in Java following the algorithm described in [14]. Our tool-chain obtained by combining various tools using a *shell script* serves as a proof of concept which is not optimised at all. For handling constraints, we use the Parma polyhedra library [1] and the Yices SMT solver [12] over *linear real arithmetic*. The construction of tree interpolation uses constrained based algorithm presented in [30] for computing interpolant of two formulas.

Description: In Table 1, *Program* represents a verification task, *Time (secs) RAHFT* and *Time (secs) RAHIT* - respectively represent the time in seconds taken by the the tool RAHFT and RAHIT respectively for solving a given task. Similarly, the number of *abstraction-refinement* iteration needed in these cases to solve a task are represented by *#Itr. RAHFT* and *#Itr. RAHIT*. Similarly, *Time (secs) TAR* and *#Itr. TAR* represent the time taken and the number of iterations needed by the tool *TAR*. The experiments were run on a MAC computer running OS X on 2.3 GHz Intel core i7 processor and 8 GB memory.

Discussion: The comparison between RAHFT and RAHIT would reflect purely the role of *interpolant tree automata* in Horn clause verification (Table 1) since the only difference between them is the refinement part using *(interpolant) tree automata*. The results show that RAHIT is more effective in practice than its counterpart RAHFT. This is justified by the number of tasks 61/68 solved by RAHIT using fewer iterations compared to RAHFT, which only solves 56/68 tasks. This is due to the generalisation of a spurious counterexample during refinement, which also captures other infeasible traces. Since these traces can be removed in the same iteration, it (possibly) reduces the number of refinements, however the solving time goes up because of the cost of computing an interpolant automaton. It is not always the case

[1] http://sv-comp.sosy-lab.org/2015/benchmarks.php

[2] https://github.com/sosy-lab/sv-benchmarks/tree/master/clauses/LIA/Eldarica

that RAHIT takes less iterations for a task (for example *Addition03 false-unreach*) than RAHFT. This is because the restructuring of the program obtained as a result of removing a set of traces may or may not favour polyhedral approximation. It is still not clear to us how to produce a right restructuring which favours polyhedral approximation. RAHIT times out on `cggmp2005_true-unreach` whereas RAHFT solves it in 5 iterations. We suspect that this is due to the cost of generating interpolant automata. We are not sure about the complexity of interpolant generation algorithm we used (the size of the formula generated was quite large with respect to the original program, magnitude not known) and there are several calls to the theorem prover to label each tree node with interpolants. So the bigger is the trace-tree, the longer it takes to compute the interpolant tree. In average, RAHIT needs 2.08 iterations and 11.40 seconds time to solve a task whereas RAHFT needs 2.32 iterations and 10.55 seconds.

The use of *interpolant tree automata* for trace generalisation and the tree automata based operations for trace-refinement are same in both RAHIT and *TAR*. Since *TAR* is not publicly available, we chose the same set of benchmarks used by *TAR* for the purpose of comparison and presented the results (the results corresponding to *TAR* are taken from [33]). The computer used in our experiments and in *TAR* [33] have similar characteristics. RAHIT solves more than half of the problems only with *abstract interpretation* over the domain of convex polyhedra without needing any refinement, which indicates its power. RAHIT solves 33/38 problems where as *TAR* solves 28/38 problems. In average, RAHIT takes less time than *TAR*. In many cases *TAR* solves a task faster than RAHIT, however it spends much longer time in some tasks. Our current constraint solver is over *linear real arithmetic*. If we use it over *linear integer arithmetic* then the results may differ. We made some observation with the problems *boustrophedon.c*, *boustrophedon_expansed.c* and *cousot.correct* (which are supposed to be interpreted over integers). In them, if we replace strict inequalities ($>,<$) with non-strict inequalities ($\geq,\leq$) over integers (for example replace $X > Y$ with $X \geq Y + 1$), then we can solve them only with *abstract interpretation* without refinement which were not solved before the transformation using our solver. On the other hand, RAHIT times out for *mergesort.error* whereas *TAR* solves it in a single iteration. This indicates that the choice of a spurious counterexample and the quality of interpolant generated from it for generalisation have some effects on verification.

5 Related work

Horn Clauses, as an intermediate language, have become a popular formalism for verification [5, 15], attracting both the logic programming and software verification communities [4]. As a result of these, several verification techniques and tools have been developed for CHCs, among others, [17, 16, 26, 11, 27, 24, 23]. To the best of our knowledge, the use of automata based approach for *abstraction-refinement* of Horn clauses is relatively new [27, 33], though the original framework proposed for imperative programs goes back to [19, 20].

The work described in [27] uses FTA based approach for refining *abstract interpretation* over the domain of convex polyhedra [8], which is similar to *trace abstraction* [19, 21, 33] with the following differences. In [27], there is an interaction between state abstraction by *abstract interpretation* [9] and trace abstraction by FTA but there is no generalisation of spurious counterexamples. On one hand, [19, 21, 33] use *trace-abstraction* with the generalisation of spurious counterexamples using *interpolant automata* and may diverge from the solution due to the lack of right generalisation. On the other hand, *abstract interpretation* [9] is one of the most promising techniques for verification which is scalable but suffers from *false alarms*. When combined with refinement *false alarms* can be minimized. Our current work takes the best of both of these approaches.

Program	Time (secs) RAHFT	#Itr. RAHFT	Time (secs) RAHIT	#Itr. RAHIT	Time (secs) TAR [33]	#Itr. TAR
addition	1	0	1	0	0.26	3
anubhav.correct	2	0	2	0	1.72	9
bfprt	1	0	1	0	0.43	6
binarysearch	2	0	2	0	0.36	5
blast.correct	5	1	11	1	8.93	65
boustrophedon.c	TO	-	TO	-	53.65	193
boustrophedon_expansed.c	TO	-	TO	-	69.06	340
buildheap	44	9	44	9	TO	-
copy1.error	11	0	11	0	12.79	19
countZero	1	0	1	0	TO	-
cousot.correct	TO	-	TO	-	TO	-
gopan.c	3	0	3	0	TO	-
halbwachs.c	TO	-	TO	-	TO	-
identity	1	0	1	0	7.67	34
inf1.error	4	1	9	1	0.51	6
inf6.correct	5	1	5	1	1.96	33
insdel.error	2	0	2	0	0.17	1
listcounter.correct	1	0	1	0	TO	
listcounter.error	9	1	9	1	0.21	1
listreversal.correct	4	0	4	0	35.79	149
listreversal.error	9	0	9	0	0.3	1
loop.error	3	0	3	0	3	3
loop1.error	8	0	8	0	10.87	19
mc91.pl	139	24	7	3	0.57	7
merge	2	0	2	0	0.86	10
mergesort.error	TO	-	TO	-	0.32	1
palindrome	2	0	2	0	0.61	6
parity	3	1	4	1	0.62	7
rate_limiter.c	3	0	3	0	49.96	130
remainder	1	0	1	0	1.5	17
running	3	1	8	2	0.4	5
scan.error	3	0	3	0	TO	-
string_concat.error	6	0	6	0	TO	-
string_concat1.error	TO	-	TO	-	TO	-
string_copy.error	3	0	3	0	TO	-
substring.error	5	0	5	0	0.55	1
substring1.error	15	0	15	0	2.84	5
triple	27	10	13	1	0.86	6
average (over 38)			**8.78**	**0.93**	**9.52**	**38.64**
solved/total			**33/38**	-	**28/38**	
Primes_true-unreach	16	4	4	1		
sum_10x0_false-unreach	5	2	12	2		
afterrec_false-unreach	2	1	3	1		
id_o3_false-unreach	6	3	7	3		
cggmp2005_variant_true-unreach	2	1	3	1		
recHanoi01_true-unreach	8	3	10	3		
cggmp2005b_true-unreach	3	1	3	1		
gcd02_true-unreach	11	4	11	4		
diamond_false-unreach	3	1	3	1		
Addition03_false-unreach	6	2	13	5		
diamond_true-unreach-call1	2	1	3	1		
id_i5_o5_false-unreach	19	8	12	5		
diamond_true-unreach-call2	6	1	5	1		
cggmp2005_true-unreach	10	5	TO	-		
gsv2008_true-unreach	3	1	3	1		
Fibocci01_true-unreach	52	10	29	6		
id_b3_o2_false-unreach	5	2	3	1		
Ackermann02_false-unreach	68	17	25	7		
mcmillan2006_true-unreach	2	1	3	1		
ddlm2013_true-unreach	TO	-	17	7		
sum_2x3_false-unreach	2	1	3	1		
fibo_5_true-unreach	TO	-	77	7		
Addition01_true-unreach	6	2	5	2		
Ackermann04_true-unreach	TO	-	59	8		
Addition02_false-unreach	4	2	5	2		
id_i10_o10_false-unreach	TO	-	39	10		
gcd01_true-unreach	9	4	5	2		
id_o10_false-unreach	TO	-	38	10		
gcnr2008_false-unreach	13	4	6	2		
Fibocci04_false-unreach	TO	-	91	11		
average (over 68)	**10.55**	**2.32**	**11.40**	**2.08**		
solved/total	**56/68**		**61/68**			

Table 1: Experiments on software verification problems. In the table “TO” means time out which is set for 300 seconds, “-” indicates the insignificance of the result.

6 Conclusion

This paper brings together *abstract interpretation* over the domain of convex polyhedra and *interpolant tree automata* in an *abstraction-refinement* scheme for Horn clause verification and combines them in a new way. Experimental results on a set of software verification benchmarks using this scheme demonstrated their usefulness in practice; showing some slight improvements over the previous approaches. In the future, we plan to evaluate its effectiveness in a larger set of benchmarks, compare our approach with other similar approaches and improve the implementation aspects of our tool. Further study is needed to find a suitable combination of abstract interpretation and interpolation based techniques, based on a deeper understanding of the interaction among interpolation, trace elimination and abstract interpretation.

References

[1] Roberto Bagnara, Patricia M. Hill & Enea Zaffanella (2008): *The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems*. *Sci. Comput. Program.* 72(1-2), pp. 3–21, doi:10.1016/j.scico.2007.08.001.

[2] Christel Baier & Cesare Tinelli, editors (2015): *TACAS . Proceedings*. *LNCS* 9035, Springer, doi:10.1007/978-3-662-46681-0.

[3] Dirk Beyer (2015): *Software Verification and Verifiable Witnesses - (Report on SV-COMP 2015)*. In Baier & Tinelli [2], pp. 401–416, doi:10.1007/978-3-662-46681-0_31.

[4] Nikolaj Bjørner, Fabio Fioravanti, Andrey Rybalchenko & Valerio Senni, editors (2014): *Proceedings First Workshop on Horn Clauses for Verification and Synthesis, HCVS 2014, Vienna, Austria, 17 July 2014*. *EPTCS* 169, doi:10.4204/EPTCS.169.

[5] Nikolaj Bjørner, Arie Gurfinkel, Kenneth L. McMillan & Andrey Rybalchenko (2015): *Horn Clause Solvers for Program Verification*. In Lev D. Beklemishev, Andreas Blass, Nachum Dershowitz, Bernd Finkbeiner & Wolfram Schulte, editors: *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, *LNCS* 9300, Springer, pp. 24–51, doi:10.1007/978-3-319-23534-9_2.

[6] Nikolaj Bjørner, Kenneth L. McMillan & Andrey Rybalchenko (2013): *On Solving Universally Quantified Horn Clauses*. In Francesco Logozzo & Manuel Fähndrich, editors: *SAS*, *LNCS* 7935, Springer, pp. 105–125. Available at `http://dx.doi.org/10.1007/978-3-642-38856-9_8`.

[7] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison & M. Tommasi (2007): *Tree Automata Techniques and Applications*. Available on: `http://www.grappa.univ-lille3.fr/tata`. Release October, 12th 2007.

[8] P. Cousot & N. Halbwachs (1978): *Automatic Discovery of Linear Restraints Among Variables of a Program*. In: *Proceedings of the 5th Annual ACM Symposium on Principles of Programming Languages*, pp. 84–96, doi:10.1145/512760.512770.

[9] Patrick Cousot & Radhia Cousot (1977): *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*. In Robert M. Graham, Michael A. Harrison & Ravi Sethi, editors: *POPL*, ACM, pp. 238–252. Available at `http://doi.acm.org/10.1145/512950.512973`.

[10] William Craig (1957): *Linear Reasoning. A New Form of the Herbrand-Gentzen Theorem*. *J. Symb. Log.* 22(3), pp. 250–268, doi:10.2307/2963593.

[11] Emanuele De Angelis, Fabio Fioravanti, Alberto Pettorossi & Maurizio Proietti (2014): *VeriMAP: A Tool for Verifying Programs through Transformations*. In Erika Ábrahám & Klaus Havelund, editors: *TACAS*, *LNCS* 8413, Springer, pp. 568–574. Available at `http://dx.doi.org/10.1007/978-3-642-54862-8_47`.

[12] Bruno Dutertre (2014): *Yices 2.2*. In Armin Biere & Roderick Bloem, editors: *Computer-Aided Verification (CAV'2014)*, *Lecture Notes in Computer Science* 8559, Springer, pp. 737–744, doi:10.1007/978-3-319-08867-9_49.

[13] J. P. Gallagher & L. Lafave (1996): *Regular Approximation of Computation Paths in Logic and Functional Languages*. In O. Danvy, R. Glück & P. Thiemann, editors: *Partial Evaluation*, *Springer-Verlag LNCS* 1110, pp. 115–136. Available at `http://dx.doi.org/10.1007/3-540-61580-6_7`.

[14] John P. Gallagher, Mai Ajspur & Bishoksan Kafle (2015): *An Optimised Algorithm for Determinisation and Completion of Finite Tree Automata*. *CoRR* abs/1511.03595. Available at `http://arxiv.org/abs/1511.03595`.

[15] John P. Gallagher & Bishoksan Kafle (2014): *Analysis and Transformation Tools for Constrained Horn Clause Verification*. *TPLP* 14(4-5 (additional materials in online edition)), pp. 90–101. Available at `http://journals.cambridge.org/action/displaySuppMaterial?cupCode=1&type=4&jid=TLP&volumeId=14&issueId=4-5&aid=9303163`.

[16] Sergey Grebenshchikov, Nuno P. Lopes, Corneliu Popeea & Andrey Rybalchenko (2012): *Synthesizing software verifiers from proof rules*. In Jan Vitek, Haibo Lin & Frank Tip, editors: *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '12, Beijing, China - June 11 - 16, 2012*, ACM, pp. 405–416, doi:10.1145/2254064.2254112. Available at `http://dl.acm.org/citation.cfm?id=2254064`.

[17] Arie Gurfinkel, Temesghen Kahsai, Anvesh Komuravelli & Jorge A. Navas (2015): *The SeaHorn Verification Framework*. In Daniel Kroening & Corina S. Pasareanu, editors: *CAV , Proceedings, Part I*, *LNCS* 9206, Springer, pp. 343–361, doi:10.1007/978-3-319-21690-4_20.

[18] Arie Gurfinkel, Temesghen Kahsai & Jorge A. Navas (2015): *SeaHorn: A Framework for Verifying C Programs (Competition Contribution)*. In Baier & Tinelli [2], pp. 447–450, doi:10.1007/978-3-662-46681-0_41.

[19] Matthias Heizmann, Jochen Hoenicke & Andreas Podelski (2009): *Refinement of Trace Abstraction*. In Jens Palsberg & Zhendong Su, editors: *Static Analysis, 16th International Symposium, SAS 2009*, *LNCS* 5673, Springer, pp. 69–85, doi:10.1007/978-3-642-03237-0_7.

[20] Matthias Heizmann, Jochen Hoenicke & Andreas Podelski (2010): *Nested interpolants*. In Manuel V. Hermenegildo & Jens Palsberg, editors: *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010*, ACM, pp. 471–482, doi:10.1145/1706299.1706353. Available at `http://dl.acm.org/citation.cfm?id=1706299`.

[21] Matthias Heizmann, Jochen Hoenicke & Andreas Podelski (2013): *Software Model Checking for People Who Love Automata*. In Sharygina & Veith [31], pp. 36–52, doi:10.1007/978-3-642-39799-8_2.

[22] Manuel V. Hermenegildo, Francisco Bueno, Manuel Carro, Pedro López-García, Edison Mera, José F. Morales & Germán Puebla (2012): *An overview of Ciao and its design philosophy*. *TPLP* 12(1-2), pp. 219–252, doi:10.1017/S1471068411000457.

[23] Hossein Hojjat, Filip Konecný, Florent Garnier, Radu Iosif, Viktor Kuncak & Philipp Rümmer (2012): *A Verification Toolkit for Numerical Transition Systems - Tool Paper*. In Dimitra Giannakopoulou & Dominique Méry, editors: *FM 2012: Formal Methods - 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings*, *LNCS* 7436, Springer, pp. 247–251, doi:10.1007/978-3-642-32759-9_21.

[24] Joxan Jaffar, Vijayaraghavan Murali, Jorge A. Navas & Andrew E. Santosa (2012): *TRACER: A Symbolic Execution Tool for Verification*. In P. Madhusudan & Sanjit A. Seshia, editors: *CAV*, *LNCS* 7358, Springer, pp. 758–766. Available at `http://dx.doi.org/10.1007/978-3-642-31424-7_61`.

[25] Ranjit Jhala & Kenneth L. McMillan (2006): *A Practical and Complete Approach to Predicate Refinement*. In Holger Hermanns & Jens Palsberg, editors: *TACAS*, *LNCS* 3920, Springer, pp. 459–473. Available at `http://dx.doi.org/10.1007/11691372_33`.

[26] Bishoksan Kafle & John P. Gallagher (2015): *Constraint Specialisation in Horn Clause Verification*. In Kenichi Asai & Kostis Sagonas, editors: *Proceedings of the 2015 Workshop on Partial Eval-*

uation and Program Manipulation, PEPM, Mumbai, India, January 15-17, 2015, ACM, pp. 85–90, doi:10.1145/2678015.2682544. Available at `http://dl.acm.org/citation.cfm?id=2678015`.

[27] Bishoksan Kafle & John P. Gallagher (2015): *Horn clause verification with convex polyhedral abstraction and tree automata-based refinement*. *Computer Languages, Systems & Structures*, doi:10.1016/j.cl.2015.11.001.

[28] Kenneth L. McMillan & Andrey Rybalchenko (2013): *Solving Constrained Horn Clauses using Interpolation*. Technical Report, Microsoft Research.

[29] Philipp Rümmer, Hossein Hojjat & Viktor Kuncak (2013): *Disjunctive Interpolants for Horn-Clause Verification*. In Sharygina & Veith [31], pp. 347–363, doi:10.1007/978-3-642-39799-8_24.

[30] Andrey Rybalchenko & Viorica Sofronie-Stokkermans (2010): *Constraint solving for interpolation*. *J. Symb. Comput.* 45(11), pp. 1212–1233. Available at `http://dx.doi.org/10.1016/j.jsc.2010.06.005`.

[31] Natasha Sharygina & Helmut Veith, editors (2013): *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*. *LNCS* 8044, Springer, doi:10.1007/978-3-642-39799-8.

[32] Robert F. Stärk (1989): *A Direct Proof for the Completeness of SLD-Resolution*. In Egon Börger, Hans Kleine Büning & Michael M. Richter, editors: *CSL '89, 3rd Workshop on Computer Science Logic, Kaiserslautern, Germany, October 2-6, 1989, Proceedings*, *LNCS* 440, Springer, pp. 382–383.

[33] Weifeng Wang & Li Jiao (2015): *Trace abstraction refinement for solving Horn clauses*. Technical Report ISCAS-SKLCS-15-19, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences. Available on: `http://lcs.ios.ac.cn/~wangwf/TechReportISCAS-SKLCS-15-19.pdf`.