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

In this paper, we study the role of *interpolant tree automata* in Horn clause verification. In an *abstraction-refinement* scheme for Horn clause verification, an interpolant tree automaton serves as a generalisation of a spurious counterexample during refinement. We apply them to verify Horn clauses with respect to some integrity constraints and present some experimental results on some software verification benchmarks. The results show some improvements over the previous approaches.

Keywords: Interpolant tree automata, Horn clauses, Abstraction-refinement.

1 Introduction

In this paper, we study the role of *interpolant tree automata* during refinement in Horn clause verification. Recently, Kafle et al. [27] described an *abstraction-refinement* scheme for Horn clause verification. In their approach refinement is based on operations over finite tree automata (FTA), where a single spurious counterexample is removed in each iteration of the *abstraction-refinement* loop. In contrast to [27], we generalise a spurious counterexample corresponding to any infeasible trace by *interpolant tree automaton* possibly recognizing infinite number of spurious counterexamples and eliminating them in one go of the *abstraction-refinement* loop. Following [33], we construct an *interpolant tree automaton* from an infeasible trace. Finally, we use them in Horn clause verification and present some experimental results on some software verification benchmarks. The results show some improvements over the previous approaches. We make the following contributions in this paper:

1. we combine *abstract interpretation* over the domain of convex polyhedra with *interpolant tree automata* in an *abstraction-refinement* scheme for Horn clause verification (Section 4);
2. we evaluate the effectiveness of this combination on a set of software verification benchmarks (Section 4.1).

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 X_i, X are (possibly empty) vectors of distinct variables, ϕ is a first order logic formula (constraint) with respect to some background theory expressed in terms of X_i, X; $p_1, \ldots, p_k, p$ are predicate symbols, $p(X)$ 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. We call clauses

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

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

Figure 1: Example CHCs Fib: it defines a Fibonacci function.

whose head is $false$ *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) $p(X), p_1(X_1), ..., p_k(X_k)$. The unifiers are encoded with constraints. A program is a set of CHCs, normally represented by P.

An interpretation of a set of CHCs P is represented as a set of *constrained facts* of the form $A \leftarrow \phi$ where A is an atom and ϕ is a satisfiable formula (not necessarily satisfiable) with respect to some background theory. An interpretation that makes each clause in P $\mathtt{true}$ 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.

Definition 1 (Horn clause verification problem) *Given a set of CHCs P with integrity constraint(s), the CHC verification problem is to check whether there exists a model 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*.

Horn clause derivations can be captured using a formal representation known as finite tree automaton.

Definition 2 (Finite tree automaton) *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. We assume that Q and Σ are disjoint.*

We assume that each CHC is associated with an identifier. An identifier 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. Given a set of CHCs and a set Σ of ranked function symbols, we define $\mathsf{id}_P : P \to \Sigma$ to be a mapping from clauses to function symbols. An identifier could be the name of a clause, but we want the name to be ranked so that we can get more information regarding the clause.

Definition 3 (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 for P.

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

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

Definition 4 (FTA for a trace-term) *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 via an example. We assume that each node in the trace-tree is labelled by an identifier.*

Example 2 (Trace FTA) *Consider the FTA in Example 1. Let $t = c_3(c_2(c_1, c_1)) \in \mathscr{A}_P$. Each $\mathtt{e_i}$ $(i = 1..4)$ represents an identifier in the trace-tree. Then $\mathscr{A}_t = (Q, Q_f, \Sigma, \Delta)$ is defined as:*

$$\begin{array}{lcl} Q & = & \{\mathtt{e_1}, \mathtt{e_2}, \mathtt{e_3}, \mathtt{e_4}\} \\ Q_f & = & \{\mathtt{e_1}\} \\ \Sigma & = & \{c_1, c_2, c_3, c_4\} \\ \Delta & = & \{c_1 \rightarrow \mathtt{e_3},\ c_1 \rightarrow \mathtt{e_4},\ c_2(\mathtt{e_3}, \mathtt{e_4}) \rightarrow \mathtt{e_2}, \\ & & \ c_3(\mathtt{e_2}) \rightarrow \mathtt{e_1}\} \end{array}$$

and Σ is the same as in $\mathscr{A}_P$.

For each trace-term there exists a corresponding derivation tree called an AND-tree, which is unique up to variable renaming [32, 12].

Definition 5 (AND-tree for a trace term) *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 a clause, an atom and a formula.*

1. *For each sub-term $c_j(t_1, \ldots, t_k)$ of t there is a corresponding node in $T(t)$ labelled by (a renamed variant of) some clause $p(X) \leftarrow \phi, p_1(X_1), \ldots, p_k(X_k)$ such that $c_j = \mathsf{id}_P(p(X) \leftarrow \phi, p_1(X_1), \ldots, p_k(X_k))$, an atom $p(X)$ and a formula 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 number (Id). We omit t from $T(t)$ when it is clear from the context.

The trace-tree $c_3(c_2(c_1, c_1))$ and its corresponding AND-tree is shown in Figure 2.

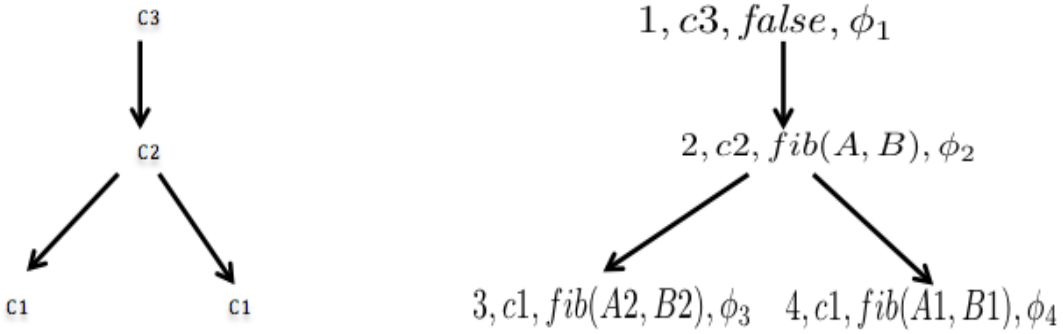

Figure 2: (a) left: a trace-term of Fib and (b) right: its AND-tree, where $\phi_1 \equiv \mathtt{A} > 5 \wedge \mathtt{B} < \mathtt{A}; \phi_2 \equiv \mathtt{A} > 1 \wedge \mathtt{A2} = \mathtt{A} - 2 \wedge \mathtt{A1} = \mathtt{A} - 1 \wedge \mathtt{B} = \mathtt{B1} + \mathtt{B2}; \phi_3 \equiv \mathtt{A2} \geq 0 \wedge \mathtt{A2} \leq 1 \wedge \mathtt{B2} = 1; \phi_4 \equiv \mathtt{A1} \geq 0 \wedge \mathtt{A1} \leq 1 \wedge \mathtt{B1} = 1$. The node labels are node Id, clause, atom and formula respectively.

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

1. ϕ, if T is a single leaf node labelled by the clause of form $H \leftarrow \phi$; or
2. $\phi \wedge \bigwedge_{i=1..n}(F(T_i))$ if the root node of T is labelled by the clause $H \leftarrow \phi, B_1, \ldots, B_n$ and has subtrees $T_1, \ldots, T_n$.

The formula of the AND-tree in Figure 2(b) is
$F(t) = \mathtt{A} > 5 \wedge \mathtt{B} < \mathtt{A} \wedge \mathtt{A} > 1 \wedge \mathtt{A2} = \mathtt{A} - 2 \wedge \mathtt{A1} = \mathtt{A} - 1 \wedge \mathtt{B} = \mathtt{B1} + \mathtt{B2}$
$\wedge \mathtt{A2} \leq 1 \wedge \mathtt{B2} = 1 \wedge \mathtt{A1} \geq 0 \wedge \mathtt{A1} \leq 1 \wedge \mathtt{B1} = 1$.

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 t in Figure 2 is unsatisfiable since $F(t)$ is unsatisfiable.

From this point on, a trace represents a trace of $\mathscr{L}(\mathscr{A}_P)$ and an AND-tree (derivation tree) represents a tree corresponding to a trace of $\mathscr{L}(\mathscr{A}_P)$ unless otherwise stated.

3 Interpolant tree automata

In this section, we describe a procedure for computing an *interpolant tree automaton* 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 6 ((Craig) Interpolant [9]) *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 [9], then every infeasible derivation using the clauses in P has an interpolant [28].

Example 3 (Interpolant example) *Let $\phi_1 \equiv A2 \leq 1 \wedge A > 1 \wedge A2 = A - 2 \wedge A1 = A - 1 \wedge B = B1 + B2$ and $\phi_2 \equiv A > 5 \wedge B < A$ such that $\phi_1 \wedge \phi_2$ is unsatisfiable. Since the formula $I \equiv A \leq 3$ fulfills all the conditions of the definition 6, it is an interpolant of ϕ_1 and ϕ_2.*

Given a node i in an AND-tree T, we call 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 the clause of form $H \leftarrow \phi$ and the node id i; or
2. ϕ, if T is a single leaf node labelled by the clause of form $H \leftarrow \phi$ and the node id different from i; or
3. true, if the root node of T is labelled by the clause $H \leftarrow \phi, B_1, \ldots, B_n$ and the node id i; or
4. $\phi \wedge \bigwedge_{l=1..n}(G(T_l))$ if the root node of T is labelled by the clause $H \leftarrow \phi, B_1, \ldots, B_n$ and the node id different from i and has subtrees $T_1, \ldots, T_n$.

Definition 7 (Tree Interpolant of an AND-tree) *Let T be an infeasible AND-tree corresponding to a trace-term of P. 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 $I(F(T_i), G(T_i))$;*

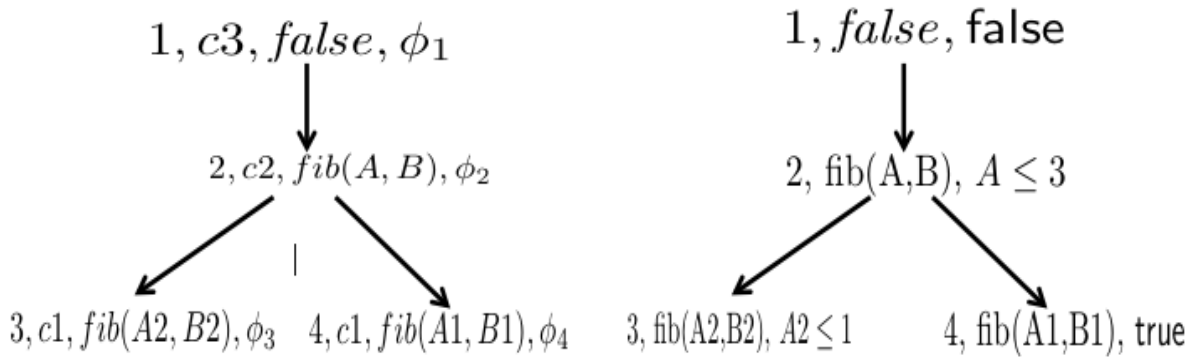

Figure 3: (a) left: *AND tree* of Figure 2 and (b) right: its *tree interpolant* . The node labels are node Id, atom and interpolant respectively. Let I_j represents an interpolant of the node j. Then we have: $I_1 \equiv \mathsf{false}$; $I_4 \equiv I(\phi_4, \phi_3 \wedge \phi_1 \wedge \phi_2)$; $I_3 \equiv I(\phi_3, \phi_1 \wedge \phi_2 \wedge I_4)$; $I_2 \equiv I(I_3 \wedge I_4 \wedge \phi_2, \phi_1)$.

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).

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

Definition 9 (Interpolant mapping Π_{TI}) *Given a tree interpolant TI, Π_{TI} is a mapping from the labels atom and Id of each node in TI to the label formula such that $\Pi_{TI}(A^j) = \phi$ where A is the atom label and ϕ is the formula label at node j.*

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

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

Property 1 (Tree interpolant property) *Let $TI(T)$ be a tree interpolant for some infeasible* AND*-tree T corresponding to a trace-term of P. Then*

1. $\Pi_{TI}(r^i) = \mathsf{false}$ *where r is the atom label of the root of $TI(T)$;*
2. *for each node j with children $j_0, ..., j_n$ $(n \geq 0)$ 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.*

Now we define an injective mapping $\sigma : Atom \times J \rightarrow FTA\ States$ where σ maps an atom occurring in a set of CHCs and an index to a FTA state.

Definition 10 (Interpolant tree automaton for Horn clauses [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. 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 $\mathscr{A}_t^I = (Q, Q_f, \Sigma, \Delta)$ 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 10 recognizes only infeasible traces terms of P which is stated in the 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 11 (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 $\Pi_{TI}^c(A)$, to be the following formula $\Pi_{TI}^c(A) = \bigwedge_j \Pi_{TI}(A^j)$, 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 $\Pi_{TI}^c = \{\Pi_{TI}^c(A) \mid A \text{ is the atom label of } TI\}$.*

It is desirable that the *interpolant tree automata* of a trace $t \in \mathscr{L}(\mathscr{A}_P)$ recognize 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 vs. Interpolant Automata) *Let $t \in \mathscr{L}(\mathscr{A}_P)$ be any infeasible trace-term. If $\Pi_{TI(t)}^c$ 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 $\mathscr{A}_P$. This automaton recognizes all trace-terms of P except some infeasible ones. Some of the infeasible trace-terms are removed because of *abstract interpretation*. P is solved or has model if $false \notin M$. If this is not the case, a trace-term $t \in \mathscr{A}_P$ is selected and checked for feasibility. If the answer is positive, P has no model, that is, P is unsolved. Otherwise t is considered spurious which drives the refinement process. The refinement in [27] consists of constructing an automaton $\mathscr{A}'_P$ which recognizes all traces in $\mathscr{L}(\mathscr{A}_P) \setminus \mathscr{L}(\mathscr{A}_t)$ 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 solved, unsolved or the resources are exhausted. We call this approach Refinement of Abstraction in Horn clauses using Finite Tree automata, `RAHFT` in short. This approach [27] lacks generalisation of spurious counterexamples during refinement. But in our current approach, we generalise a spurious counterexample through the use of *interpolant automata*. Section 3 describes on how to compute an *interpolant automaton* (adapted 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 $\mathscr{L}(\mathscr{A}_P) \setminus \mathscr{L}(\mathscr{A}^I_t)$. We call this approach Refinement of Abstraction in Horn clauses using Interpolant Tree automata, `RAHIT` in short.

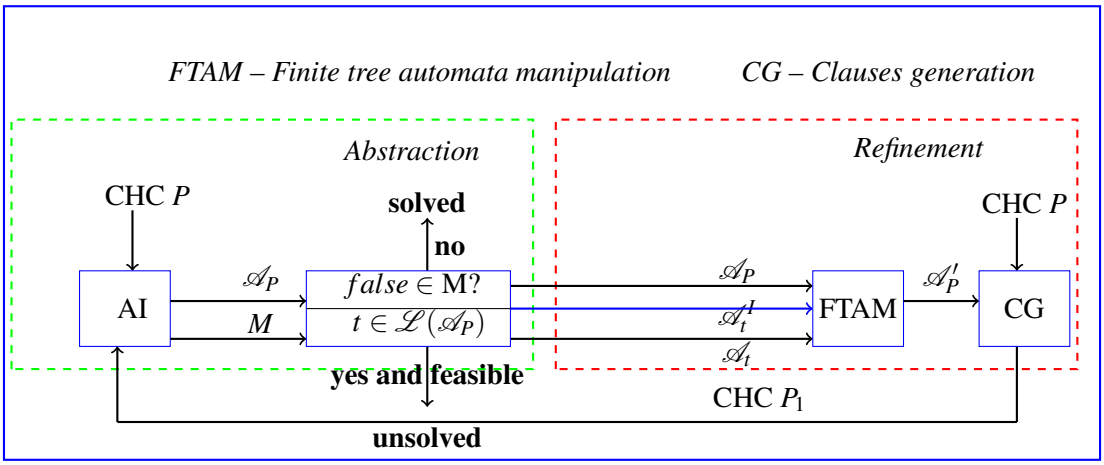

Figure 4: *Abstraction-refinement scheme in Horn clause verification [27]. M is an approximation produced as a result of abstract interpretation. $\mathscr{A}'_P$ recognizes all traces in $\mathscr{L}(\mathscr{A}_P) \setminus \mathscr{L}(\mathscr{A}_t)$.*

Next we briefly describe on how to generate an abstract FTA corresponding to a set of clauses P which captures all trace-terms of P except some infeasible ones using *abstract interpretation* and show some experimental results using our current approach on some software verification benchmarks.

In Figure 5, we present an algorithm, which constructs an abstract FTA corresponding to P from an approximation of the minimal model of P. Note that the FTA produced by the algorithm is same as

procedure ABSTRACT AUTOMATA CONSTRUCTION USING CONVEX POLYHEDRAL APPROXIMATION FOR HORN CLAUSES(Input: CHCs P and polyhedral approximation M, Output: $\mathscr{A}_P = (Q, Q_f, \Sigma, \Delta)$)

$\Delta \leftarrow \emptyset$;
$Q \leftarrow \emptyset$;
$\Sigma \leftarrow \{\mathsf{id}_P(cl) : cl \in P\}$

for all $c : p(X) \leftarrow \phi, p_1(X_1) \ldots p_k(X_k)(k \geq 0) \in P$ **do**
▷ where $c = \mathsf{id}_P(p(X) \leftarrow \phi, p_1(X_1), \ldots, p_k(X_k)$
if $\mathsf{SAT}(\phi \wedge \gamma(p_1(X_1), M) \wedge \ldots \wedge \gamma(p_1(X_1), M))$ **then**
$\Delta \leftarrow \Delta \cup \{c(p_1, \ldots, p_k) \rightarrow p\}$
$Q \leftarrow Q \cup \{p, p_1, \ldots p_k\}$
end if
end for
$Q \leftarrow Q \cup \{false\}$
$Q_f \leftarrow \{false\}$
return $\mathscr{A}_P = (Q, Q_f, \Sigma, \Delta)$
end procedure

Figure 5: Generation of abstract automata using Convex Polyhedral Analysis. Let $\gamma : \mathscr{A} \times \mathscr{M} \rightarrow \mathscr{F}$ be an injective mapping. $\gamma(A, M)$ returns a formula $\phi \in \mathscr{F}$ corresponding to an atom $A \in \mathscr{A}$ in $M \in \mathscr{M}$.

$\mathscr{A}_P$ (definition 3) except that transitions corresponding to clauses whose bodies are not satisfiable in the convex polyhedral approximation are omitted, since they cannot contribute to feasible derivations. We refer to [21] for details on how to construct an over-approximation of the minimal model of P using *abstract interpretation* over the domain of convex polyhedra. An over-approximation of a set of CHCs is represented as a set of *constrained facts* of the form $A \leftarrow \phi$ (one for each A) where A is an atomic formula $p(Z_1, \ldots, Z_n)$ where $Z_1, \ldots, Z_n$ are distinct variables and ϕ is a formula over $Z_1, \ldots, Z_n$ with respect to some background theory.

Given an over-approximation M, the algorithm in Figure 5 generates an abstract FTA for P. The idea is as follows: for each of the clause of the form $c : p(X) \leftarrow \phi, p_1(X_1) \ldots p_k(X_k)(k \geq 0) \in P$, if the conjunction of over-approximation of the body atoms $p_1(X_1) \ldots p_k(X_k)$ under M is satisfiable (see *if ... then condition* in Figure 5), then we add the corresponding transition $\{c(p_1, \ldots, p_k) \rightarrow p\}$ to the set of transitions, and the corresponding set of states $\{p, p_1, \ldots p_k\}$ to the set of states. The alphabet Σ is the set of clause identifiers of P and the set of final states is the singleton set $\{false\}$.

Lemma 1 (Soundness of Algorithm 5) *Let P be a set of clauses and M be an over-approximation of the minimal model of P, then the abstract FTA generated for P using the procedure described in Figure 5 recognizes all feasible trace-terms of P.*

Example 5 (Abstract 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 follows in textual form: {`fib(A,B):-[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 abstract FTA constructed for the example program using M. Since the bodies of each clauses except the integrity constraint are satisfied under M,*

the abstract 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 [17, 16] 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[3]. 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 over *integer linear arithmetic*.

The goal of this experiment is to study the role of *interpolant tree automata* in Horn clause verification following the scheme described in [27]. For this purpose, we made the following comparison between the tools:

1. compare *RAHIT* with *RAHFT*, which compares the effect of removing a set of traces rather than a single trace; and

2. compare *RAHIT* with the *trace-abstraction* tool [33] (*TAR* from now on), which differs in using polyhedral approximation rather than property-based abstraction, and in explicitly removing the traces by program transformation.

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 [13]. 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 [11] 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 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.

[1] http://sv-comp.sosy-lab.org/2015/benchmarks.php
[2] https://github.com/sosy-lab/sv-benchmarks/tree/master/clauses/LIA/Eldarica
[3] http://lara.epfl.ch/w/eldarica

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 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 favor polyhedral approximation. It is still not clear to us how to produce a right restructuring which favors polyhedral approximation. *RAHIT* times out on `cggmp2005_true-unreach` due to the cost of generating interpolant automata whereas *RAHFT* solves it in 5 iterations. In average, *RAHIT* needs 2.08 iterations and 11.40 seconds time to solve a task whereas *RAHIT* 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 (tasks 31-68 in Table 1) used by *TAR* for the purpose of comparison and presented the results (the results corresponding to *TAR* are taken from [33]). *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 which gives high average. In average (for tasks 31-68 in Table 1), *RAHIT* needs less than one iteration and 8.78 seconds to solve a task whereas *TAR* needs almost 38 iterations and 9.52 seconds. We made an interesting observation with the problems *boustrophedon.c*, *boustrophedon_expansed.c* and *cousot.correct*. If we replace greater than $>$ and lesser than $<$ constraints with greater equal $\geq$ and lesser equal $\leq$ for integer problems (for example replace $X > Y$ with $X \geq Y + 1$), then we can solve the above mentioned problems only with *abstract interpretation* without refinement which were unsolved before the transformation. This points out that if we use our underlying solver over *linear integer arithmetic* then the results may differ. But *RAHIT* times out for *mergesort.error* whereas *TAR* solves it in a single iteration. This shows that the choice of a spurious counterexample and the quality of interpolant generated from it for generalisation have effects on verification.

5 Related work

Horn Clauses, as an intermediate language, have become a popular formalism for verification [5, 14], 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, [16, 15, 26, 10, 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 [18, 19].

The work described in [27] uses FTA based approach for refining *abstract interpretation* over the domain of convex polyhedra [7], which is similar in essence to *trace abstraction* [18, 20, 33] with the following differences. In [27], there is an interaction between state abstraction by *abstract interpretation* [8] and trace abstraction by FTA but there is no generalisation of spurious counterexamples. On one hand,

Nr.	Program	Time (secs) RAHFT	#Itr. RAHFT	Time (secs) RAHIT	#Itr. RAHIT	Time (secs) TAR [33]	#Itr. TAR
1	Primes_true-unreach	16	4	4	1	NA	NA
2	sum_10x0_false-unreach	5	2	12	2	NA	NA
3	afterrec_false-unreach	2	1	3	1	NA	NA
4	id_o3_false-unreach	6	3	7	3	NA	NA
5	cggmp2005_variant_true-unreach	2	1	3	1	NA	NA
6	recHanoi01_true-unreach	8	3	10	3	NA	NA
7	cggmp2005b_true-unreach	3	1	3	1	NA	NA
8	gcd02_true-unreach	11	4	11	4	NA	NA
9	diamond_false-unreach	3	1	3	1	NA	NA
10	Addition03_false-unreach	6	2	13	5	NA	NA
11	diamond_true-unreach-call1	2	1	3	1	NA	NA
12	id_i5_o5_false-unreach	19	8	12	5	NA	NA
13	diamond_true-unreach-call2	6	1	5	1	NA	NA
14	cggmp2005_true-unreach	10	5	TO	-	NA	NA
15	gsv2008_true-unreach	3	1	3	1	NA	NA
16	Fibonacci01_true-unreach	52	10	29	6	NA	NA
17	id_b3_o2_false-unreach	5	2	3	1	NA	NA
18	Ackermann02_false-unreach	68	17	25	7	NA	NA
19	mcmillan2006_true-unreach	2	1	3	1	NA	NA
20	ddlm2013_true-unreach	TO	-	17	7	NA	NA
21	sum_2x3_false-unreach	2	1	3	1	NA	NA
22	fibo_5_true-unreach	TO	-	77	7	NA	NA
23	Addition01_true-unreach	6	2	5	2	NA	NA
24	Ackermann04_true-unreach	TO	-	59	8	NA	NA
25	Addition02_false-unreach	4	2	5	2	NA	NA
26	id_i10_o10_false-unreach	TO	-	39	10	NA	NA
27	gcd01_true-unreach	9	4	5	2	NA	NA
28	id_o10_false-unreach	TO	-	38	10	NA	NA
29	gcnr2008_false-unreach	13	4	6	2	NA	NA
30	Fibonacci04_false-unreach	TO	-	91	11	NA	NA
31	addition	1	0	1	0	0.26	3
32	anubhav.correct	2	0	2	0	1.72	9
33	bfprt	1	0	1	0	0.43	6
34	binarysearch	2	0	2	0	0.36	5
35	blast.correct	5	1	11	1	8.93	65
36	boustrophedon.c	TO	-	TO	-	53.65	193
37	boustrophedon_expansed.c	TO	-	TO	-	69.06	340
38	buildheap	44	9	44	9	TO	-
39	copy1.error	11	0	11	0	12.79	19
40	countZero	1	0	1	0	TO	-
41	cousot.correct	TO	-	TO	-	TO	-
42	gopan.c	3	0	3	0	TO	-
43	halbwachs.c	TO	-	TO	-	TO	-
44	identity	1	0	1	0	7.67	34
45	inf1.error	4	1	9	1	0.51	6
46	inf6.correct	5	1	5	1	1.96	33
47	insdel.error	2	0	2	0	0.17	1
48	listcounter.correct	1	0	1	0	TO	
49	listcounter.error	9	1	9	1	0.21	1
50	listreversal.correct	4	0	4	0	35.79	149
51	listreversal.error	9	0	9	0	0.3	1
52	loop.error	3	0	3	0	3	3
53	loop1.error	8	0	8	0	10.87	19
54	mc91.pl	139	24	7	3	0.57	7
55	merge	2	0	2	0	0.86	10
56	mergesort.error	TO	-	TO	-	0.32	1
57	palindrome	2	0	2	0	0.61	6
58	parity	3	1	4	1	0.62	7
59	rate_limiter.c	3	0	3	0	49.96	130
60	remainder	1	0	1	0	1.5	17
61	running	3	1	8	2	0.4	5
62	scan.error	3	0	3	0	TO	-
63	string_concat.error	6	0	6	0	TO	-
64	string_concat1.error	TO	-	TO	-	TO	-
65	string_copy.error	3	0	3	0	TO	-
66	substring.error	5	0	5	0	0.55	1
67	substring1.error	15	0	15	0	2.84	5
68	triple	27	10	13	1	0.86	6
	average (over 68)	**10.55**	**2.32**	**11.40**	**2.08**	-	-
	average (over 38)	-	-	**8.78**	**0.93**	**9.52**	**38.64**
	solved/total	**56/68**	-	**61/68, 33/38**	-	**28/38**	

Table 1: Experiments on software verification problems. In the table "TO" means time out which is set for five minutes, "-" means not relevant and "NA" means not available.

[18, 20, 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* [8] 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.

6 Conclusion

We used the notion of *interpolant tree automata* for generalising a spurious counterexample in an *abstraction-refinement* scheme for Horn clause verification. Experimental results on a set of software verification benchmarks using this scheme demonstrated their usefulness in practice; showing 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 the tool-chain.

References

[1] R. Bagnara, P. M. Hill & E. 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. Available at `http://dx.doi.org/10.1016/j.scico.2007.08.001`.

[2] C. Baier & C. Tinelli, editors (2015): *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*. *Lecture Notes in Computer Science* 9035, Springer, doi:10.1007/978-3-662-46681-0. Available at `http://dx.doi.org/10.1007/978-3-662-46681-0`.

[3] D. 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. Available at `http://dx.doi.org/10.1007/978-3-662-46681-0_31`.

[4] N. Bjørner, F. Fioravanti, A. Rybalchenko & V. 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. Available at `http://dx.doi.org/10.4204/EPTCS.169`.

[5] N. Bjørner, A. Gurfinkel, K. L. McMillan & A. Rybalchenko (2015): *Horn Clause Solvers for Program Verification*. In L. D. Beklemishev, A. Blass, N. Dershowitz, B. Finkbeiner & W. Schulte, editors: *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday*, *Lecture Notes in Computer Science* 9300, Springer, pp. 24–51, doi:10.1007/978-3-319-23534-9_2. Available at `http://dx.doi.org/10.1007/978-3-319-23534-9_2`.

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

[7] 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.

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

[9] W. Craig (1957): *Linear Reasoning. A New Form of the Herbrand-Gentzen Theorem*. *J. Symb. Log.* 22(3), pp. 250–268, doi:10.2307/2963593. Available at `http://dx.doi.org/10.2307/2963593`.

[10] E. De Angelis, F. Fioravanti, A. Pettorossi & M. Proietti (2014): *VeriMAP: A Tool for Verifying Programs through Transformations*. In E. Ábrahám & K. Havelund, editors: *TACAS*, *LNCS* 8413, Springer, pp. 568–574. Available at http://dx.doi.org/10.1007/978-3-642-54862-8_47.

[11] B. Dutertre (2014): *Yices 2.2*. In A. Biere & R. Bloem, editors: *Computer-Aided Verification (CAV'2014)*, *Lecture Notes in Computer Science* 8559, Springer, pp. 737–744.

[12] 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 Lecture Notes in Computer Science* 1110, pp. 115–136. Available at http://dx.doi.org/10.1007/3-540-61580-6_7.

[13] J. P. Gallagher, M. Ajspur & B. 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.

[14] J. P. Gallagher & B. 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.

[15] S. Grebenshchikov, N. P. Lopes, C. Popeea & A. Rybalchenko (2012): *Synthesizing software verifiers from proof rules*. In J. Vitek, H. Lin & F. 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://doi.acm.org/10.1145/2254064.2254112.

[16] A. Gurfinkel, T. Kahsai, A. Komuravelli & J. A. Navas (2015): *The SeaHorn Verification Framework*. In D. Kroening & C. S. Pasareanu, editors: *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, *Lecture Notes in Computer Science* 9206, Springer, pp. 343–361, doi:10.1007/978-3-319-21690-4_20. Available at http://dx.doi.org/10.1007/978-3-319-21690-4_20.

[17] A. Gurfinkel, T. Kahsai & J. 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. Available at http://dx.doi.org/10.1007/978-3-662-46681-0_41.

[18] M. Heizmann, J. Hoenicke & A. Podelski (2009): *Refinement of Trace Abstraction*. In J. Palsberg & Z. Su, editors: *Static Analysis, 16th International Symposium, SAS 2009*, *LNCS* 5673, Springer, pp. 69–85, doi:10.1007/978-3-642-03237-0_7. Available at http://dx.doi.org/10.1007/978-3-642-03237-0_7.

[19] M. Heizmann, J. Hoenicke & A. Podelski (2010): *Nested interpolants*. In M. V. Hermenegildo & J. 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://doi.acm.org/10.1145/1706299.1706353.

[20] M. Heizmann, J. Hoenicke & A. 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. Available at http://dx.doi.org/10.1007/978-3-642-39799-8_2.

[21] K. S. Henriksen, G. Banda & J. P. Gallagher (2007): *Experiments with a Convex Polyhedral Analysis Tool for Logic Programs*. In: *Workshop on Logic Programming Environments, Porto, 2007*. Available at http://arxiv.org/abs/0712.2737.

[22] M. V. Hermenegildo, F. Bueno, M. Carro, P. López-García, E. Mera, J. F. Morales & G. Puebla (2012): *An overview of Ciao and its design philosophy*. *TPLP* 12(1-2), pp. 219–252, doi:10.1017/S1471068411000457. Available at http://dx.doi.org/10.1017/S1471068411000457.

[23] H. Hojjat, F. Konecný, F. Garnier, R. Iosif, V. Kuncak & P. Rümmer (2012): *A Verification Toolkit for Numerical Transition Systems - Tool Paper*. In D. Giannakopoulou & D. Méry, editors: *FM 2012: Formal Methods - 18th International Symposium, Paris, France, August 27-31, 2012. Proceedings*, *Lecture Notes in Computer Science* 7436, Springer, pp. 247–251, doi:10.1007/978-3-642-32759-9_21. Available at http://dx.doi.org/10.1007/978-3-642-32759-9_21.

[24] J. Jaffar, V. Murali, J. A. Navas & A. E. Santosa (2012): *TRACER: A Symbolic Execution Tool for Verification*. In P. Madhusudan & S. 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] R. Jhala & K. L. McMillan (2006): *A Practical and Complete Approach to Predicate Refinement*. In H. Hermanns & J. Palsberg, editors: *TACAS*, *LNCS* 3920, Springer, pp. 459–473. Available at `http://dx.doi.org/10.1007/11691372_33`.

[26] B. Kafle & J. P. Gallagher (2015): *Constraint Specialisation in Horn Clause Verification*. In K. Asai & K. Sagonas, editors: *Proceedings of the 2015 Workshop on Partial Evaluation and Program Manipulation, PEPM, Mumbai, India, January 15-17, 2015*, ACM, pp. 85–90, doi:10.1145/2678015.2682544. Available at `http://doi.acm.org/10.1145/2678015.2682544`.

[27] B. Kafle & J. 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. Available at `http://dx.doi.org/10.1016/j.cl.2015.11.001`.

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

[29] P. Rümmer, H. Hojjat & V. Kuncak (2013): *Disjunctive Interpolants for Horn-Clause Verification*. In Sharygina & Veith [31], pp. 347–363, doi:10.1007/978-3-642-39799-8. Available at `http://dx.doi.org/10.1007/978-3-642-39799-8`.

[30] A. Rybalchenko & V. 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] N. Sharygina & H. Veith, editors (2013): *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*. *Lecture Notes in Computer Science* 8044, Springer, doi:10.1007/978-3-642-39799-8. Available at `http://dx.doi.org/10.1007/978-3-642-39799-8`.

[32] R. F. Stärk (1989): *A Direct Proof for the Completeness of SLD-Resolution*. In E. Börger, H. K. Büning & M. 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] W. Wang & L. 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.