Interpolant Tree Automata and their Application in Horn Clause Verification *

Bishoksan Kafle 
Roskilde University, Denmark 
kafle@ruc.dk

John P. Gallagher 
Roskilde University, Denmark 
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.
A constrained Horn clause (CHC) is a first order predicate logic formula of the form $\phi \land p_1(X_1) \land \ldots \land p_k(X_k) \rightarrow p(X)$ (where $\phi$ 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 $\phi$ is expressed in terms of $X_i, X$, which can be achieved by adding equalities to $\phi$. $p(X)$ is the head of the clause and $\phi \land p_1(X_1) \land \ldots \land p_k(X_k)$ 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 $H, B_1, \ldots, B_k$ stand for atomic formulas (atoms) $p(X), p_1(X_1), \ldots, p_k(X_k)$. 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 $\phi$ 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 ($\theta$ 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)** An FTA $\mathcal{A}$ is a tuple $(Q, Q_f, \Sigma, \Delta)$, where $Q$ is a finite set of states, $Q_f \subseteq Q$ is a set of final states, $\Sigma$ is a set of function symbols, and $\Delta$ 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 $\Sigma$ are disjoint.

We assume that each CHC in a program $P$ is associated with an identifier by a mapping $id_P : P \rightarrow \Sigma$. An identifier (an element of $\Sigma$) 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 $\mathcal{A}_P = (Q, Q_f, \Sigma, \Delta)$ where

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

Figure 1: Example CHCs (Fib) defining a Fibonacci function.
Example 1 Let Fib be the set of CHCs in Figure 1. Let \( \text{id}_{\text{Fib}} \) map the clauses to identifiers \( c_1, c_2, c_3 \) respectively. Then \( \mathcal{A}_{\text{Fib}} = (Q, Q_f, \Sigma, \Delta) \) where:

\[
\begin{align*}
Q &= \{ \text{fib}, \text{false} \} \\
Q_f &= \{ \text{false} \} \\
\Sigma &= \{ c_1, c_2, c_3 \} \\
\Delta &= \{ c_1 \to \text{fib}, c_2(\text{fib}, \text{fib}) \to \text{fib}, \\
& \quad c_3(\text{fib}) \to \text{false} \}
\end{align*}
\]

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 \( t \in \mathcal{L}(\mathcal{A}_P) \). There exists an FTA \( \mathcal{A}_t \) such that \( \mathcal{L}(\mathcal{A}_t) = \{ t \} \). We illustrate the construction with an example.

Example 2 (Trace FTA) Consider the FTA in Example 7. Let \( \mathcal{A}_t = c_3(c_2(c_1, c_1)) \in \mathcal{L}(\mathcal{A}_P) \). Then \( \mathcal{A}_t = (Q, Q_f, \Sigma, \Delta) \) is defined as:

\[
\begin{align*}
Q &= \{ e_1, e_2, e_3, e_4 \} \\
Q_f &= \{ e_1 \} \\
\Sigma &= \{ c_1, c_2, c_3, c_4 \} \\
\Delta &= \{ c_1 \to e_3, c_1 \to e_4, c_2(e_3, e_4) \to e_2, \\
& \quad c_3(e_2) \to e_1 \}
\end{align*}
\]

where \( \Sigma \) is the same as in \( \mathcal{A}_P \) and the states \( e_i \) \( (i = 1 \ldots 4) \) represent the nodes in the trace-tree, with root node \( e_1 \) as the final state.

A trace-term is a representation of a derivation trees, called an AND-tree 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 \( t \in \mathcal{L}(\mathcal{A}_P) \). 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 \( c_j = \text{id}_P(p(X) \leftarrow \phi, p_1(X_1), \ldots, p_k(X_k)) \), and a constraint \( \phi \); 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.
The formula represented by an AND-tree $T$, represented by $F(T)$ is
1. $\phi$, if $T$ is a single leaf node labelled by a constraint $\phi$; or
2. $\phi \land \bigwedge_{i=1..n}(F(T_i))$ if the root node of $T$ is labelled by the constraint $\phi$ and has subtrees $T_1, \ldots, T_n$.

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

$$A > 5 \land B < A \land A > 1 \land A2 = A - 2 \land A1 = A - 1 \land B = B1 + B2 \land A2 \geq 0 \land A2 \leq 1 \land B2 = 1 \land A1 \geq 0 \land A1 \leq 1 \land B1 = 1$$

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-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 $\phi_1, \phi_2$ such that $\phi_1 \land \phi_2$ is unsatisfiable, a (Craig) interpolant is a formula $I$ with (1) $\phi_1 \rightarrow I$; (2) $I \land \phi_2 \rightarrow \text{false}$; and (3) $\text{vars}(I) \subseteq \text{vars}(\phi_1) \cap \text{vars}(\phi_2)$. An interpolant of $\phi_1$ and $\phi_2$ is represented by $I(\phi_1, \phi_2)$.

The existence of an interpolant implies that $\phi_1 \land \phi_2$ 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 $\phi_1 \equiv A2 \leq 1 \land A > 1 \land A2 = A - 2 \land A1 = A - 1 \land B = B1 + B2$ and $\phi_2 \equiv A > 5 \land B < A$ such that $\phi_1 \land \phi_2$ is unsatisfiable. Since the formula $I \equiv A \leq 3$ fulfills all the conditions of Definition 4, it is an interpolant of $\phi_1$ and $\phi_2$.

Given a node $i$ in an AND-tree $T$, we call $T_i$ the sub-tree rooted at $i$, $\phi_i$ 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 $\phi$ and the node is $i$; or
2. $\phi$, if $T$ is a single leaf node labelled by constraint $\phi$ and the node is different from $i$; or
3. $true$, if the root node of $T$ is labelled by the constraint $\phi$ and the node is $i$; or
4. $\phi \land \bigwedge_{i=1,n}(G(T_i))$ if the root node of $T$ is labelled by the constraint $\phi$, and the node is different from $i$ and has subtrees $T_1, \ldots, T_n$.

Definition 5 (Tree Interpolant of an AND-tree) 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 $F_i$ as $(\phi_i \land \bigwedge_{k=1,n}I_k)$ 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 $I(F_i, G(T_i))$.

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

![Figure 3](image_url)

Figure 3: AND tree of Figure 2(left) and its tree interpolant (right). Let $I_j$ represents an interpolant of the node $j$. Then $I_1 \equiv false$; $I_2 \equiv I(\phi_3, \phi_3 \land \phi_1 \land \phi_2); I_3 \equiv I(\phi_3, \phi_3 \land \phi_2 \land I_4); I_2 \equiv I(I_5 \land I_4 \land \phi_2, \phi_1)$.

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 $\Pi_{TI}$) Given a tree interpolant $TI$ for some tree, $\Pi_{TI}$ 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 $\psi$ is the formula label at node $j$. 
For our example program $\Pi_{TI}$ is the following:

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

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

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

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

- $Q = \{\sigma(A,i) : A$ is the atom label of the node $i$ of $T_I(t)\}$;
- $F = \{\sigma(A,i) : A$ is the atom label of the root of $T_I(t)\}$;
- $\Sigma$ is a set of function symbols of $P$;
- $\Delta = \{c(p^h_1, \ldots, p^h_k) \rightarrow p^I \mid cl = p(X) \leftarrow \phi, p_1(X_1), \ldots, p_k(X_k) \in P, c = \text{id}_P(cl), \rho(p^I) = p, \rho(p^h_m) = p_m$ for $m = 1..k$ and $\Pi_{TI}(p^I)(X) \leftarrow \phi, \Pi_{TI}(p^h_1(X_1), \ldots, \Pi_{TI}(p^h_k)(X_k))\}$.

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

$$Q = \{\text{fib}^2,\text{fib}^3,\text{fib}^4,\text{error}\}$$

$$Q_f = \{\text{error}\}$$

$$\Sigma = \{c_1,c_2,c_3\}$$

$$\Delta = \{c_1 \rightarrow \text{fib}^2,c_1 \rightarrow \text{fib}^3,c_1 \rightarrow \text{fib}^4,$n$$

$$c_2(\text{fib}^2,\text{fib}^2) \rightarrow \text{fib}^4,c_2(\text{fib}^2,\text{fib}^3) \rightarrow \text{fib}^2,$n$$

$$c_2(\text{fib}^3,\text{fib}^2) \rightarrow \text{fib}^4,c_2(\text{fib}^3,\text{fib}^2) \rightarrow \text{fib}^4,$n$$

$$c_2(\text{fib}^3,\text{fib}^3) \rightarrow \text{fib}^2,c_2(\text{fib}^3,\text{fib}^3) \rightarrow \text{fib}^4,$n$$

$$c_2(\text{fib}^4,\text{fib}^2) \rightarrow \text{fib}^4,c_2(\text{fib}^4,\text{fib}^3) \rightarrow \text{fib}^2,$n$$

$$c_2(\text{fib}^4,\text{fib}^3) \rightarrow \text{fib}^4,c_2(\text{fib}^4,\text{fib}^4) \rightarrow \text{fib}^4,$n$$

$$c_3(\text{fib}^2) \rightarrow \text{error},c_3(\text{fib}^3) \rightarrow \text{error}\}$$

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 $t \in \mathcal{L}(\mathcal{P})$ be any infeasible trace-term. Then the interpolant automaton $\mathcal{A}_t^I$ recognizes only infeasible trace-terms of $P$. 
Definition 9 (Conjunctive interpolant mapping) Given an interpolant mapping $\Pi_{T I}$ of a tree interpolant $T I$, we define a conjunctive interpolant mapping for an atom label $A$ of any node in $T I$, represented as $\Pi^c_{T I}(A)$, to be the following formula $\Pi^c_{T I}(A) = \land_j \Pi_{T I}(A^j)$, where $j$ ranges over the nodes of $T I$. It is the conjunction of interpolants of all the nodes of $T I$ with atom label $A$. The conjunctive interpolant mapping of $T I$ is represented is $\Pi^c_{T I} = \{ \Pi^c_{T I}(A) \mid A$ is the atom label of $T I$\}.

It is desirable that the interpolant tree automaton of a trace $t \in L(\mathcal{A}_P)$ 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 $t \in L(\mathcal{A}_P)$ be any infeasible trace-term. If $\Pi^c_{T I(i)}$ is a model of $P$, then the interpolant automaton of $t$ recognizes 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 $\mathcal{A}_P^M$ (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 $t \in L(\mathcal{A}_P^M)$ 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 $\mathcal{A}_P'$ which recognizes all traces in $L(\mathcal{A}_P^M) \setminus L(\mathcal{A}_i)$ and generating a refined set of clauses from $P$ and $\mathcal{A}_P'$. 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. $\mathcal{A}_i^I$ 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 $\mathcal{A}_P'$ now recognizes all traces in $L(\mathcal{A}_P^M) \setminus L(\mathcal{A}_i^I)$. 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, $\mathcal{A}_P^M$, 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 $\phi$ is a constraint over $X$. Given such an set $M$, define $\gamma_M$ 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. $\mathcal{A}_P^M$ recognizes all traces in $\mathcal{L}(\mathcal{A}_P^M) \setminus \mathcal{L}(\mathcal{A}_f)$.

$M$ 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 $P$, $\phi \land \bigwedge_{i=1}^n \gamma_M(p_i(X_i)) \rightarrow \gamma_M(p(X))$.

Given such an $M$, we construct an FTA corresponding to $P$, which is the same as $\mathcal{A}_P$ (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 $\mathcal{A}_P^M = (Q, Q_f, \Sigma, \Delta_M)$ where $Q, Q_f$ and $\Sigma$ are the same as for $\mathcal{A}_P$ (Definition 2) and $\Delta_M$ is the following set of transitions.

$$\Delta_M = \{ c(p_1, \ldots, p_n) \rightarrow p \mid \text{id}_P(c) = p(X) \leftarrow \phi, p_1(X_1), \ldots, p_n(X_n),$$

$$\text{SAT}(\phi \land \bigwedge_{i=1}^n \gamma_M(p_i(X_i))) \}$$

**Lemma 1** Let $P$ be a set of clauses and $M$ be a model of $P$ then $\mathcal{L}(\mathcal{A}_P^M)$ 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 7 the convex polyhedral abstraction produces an over-approximation $M$ which is represented as

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

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 7 except the transition $c_3(\text{fib}) \rightarrow \text{false}$, 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\(^1\)\(\text{[3]}\) (recursive category) and translated them to Horn clauses using inter-procedural encoding of SeaHorn \(\text{[18]}\text{[17]}\) 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 \(\text{[23]}\). This set consists of problems, among others, from the NECLA static analysis suite, from the paper \(\text{[25]}\). These tasks are also considered in \(\text{[33]}\) 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 \(\text{[33]}\) (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 4.

**Implementation:** Most of the tools in our tool-chain depicted in Figure 4 are implemented in Ciao Prolog \(\text{[22]}\) except the one for determinisation of FTA, which is implemented in Java following the algorithm described in \(\text{[14]}\). 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 \(\text{[1]}\) and the Yices SMT solver \(\text{[12]}\) over linear real arithmetic. The construction of tree interpolation uses constrained based algorithm presented in \(\text{[30]}\) for computing interpolant of two formulas.

**Description:** In Table 4, 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 4) 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 cgmp2005_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_expanded.c and cousot.correct (which are supposed to be interpreted over integers). In them, if we replace strict inequalities (> , <) with non-strict inequalities (≥, ≤) over integers (for example replace X > Y with X ≥ 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.
<table>
<thead>
<tr>
<th>Program</th>
<th>Time (secs) RAHFT</th>
<th>#Itr. RAHFT</th>
<th>Time (secs) TAR</th>
<th>#Itr. TAR</th>
<th>Time (secs) RAHFT</th>
<th>#Itr. RAHFT</th>
<th>Time (secs) TAR</th>
<th>#Itr. TAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>addition</td>
<td>0</td>
<td>0</td>
<td>0.26</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1.72</td>
<td>9</td>
</tr>
<tr>
<td>anubisv.correct</td>
<td>2</td>
<td>0</td>
<td>1.57</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>5</td>
</tr>
<tr>
<td>bft</td>
<td>2</td>
<td>0</td>
<td>0.43</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>5</td>
</tr>
<tr>
<td>binarysearch</td>
<td>5</td>
<td>1</td>
<td>8.93</td>
<td>69</td>
<td>11</td>
<td>0.36</td>
<td>19</td>
<td>-</td>
</tr>
<tr>
<td>blastasser.c</td>
<td>TO</td>
<td>-</td>
<td>53.65</td>
<td>193</td>
<td>TO</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>blastasser_expanded.c</td>
<td>TO</td>
<td>-</td>
<td>69.06</td>
<td>340</td>
<td>TO</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>buildheap</td>
<td>11</td>
<td>0</td>
<td>12.79</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td>countZero</td>
<td>1</td>
<td>0</td>
<td>7.67</td>
<td>34</td>
<td>1</td>
<td>0.21</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>cousot.correct</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td>gopan.c</td>
<td>1</td>
<td>0</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td>ntlwatches.c</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td>identity</td>
<td>1</td>
<td>0</td>
<td>0.36</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.72</td>
<td>5</td>
</tr>
<tr>
<td>id1.error</td>
<td>4</td>
<td>1</td>
<td>0.51</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>5</td>
</tr>
<tr>
<td>id2.error</td>
<td>5</td>
<td>1</td>
<td>1.96</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>5</td>
</tr>
<tr>
<td>id.error</td>
<td>2</td>
<td>0</td>
<td>0.36</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>1</td>
</tr>
<tr>
<td>listcounter.correct</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>TO</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td>listcounter.error</td>
<td>9</td>
<td>1</td>
<td>1.21</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.36</td>
<td>1</td>
</tr>
<tr>
<td>listreverse.correct</td>
<td>4</td>
<td>0</td>
<td>35.79</td>
<td>149</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>1</td>
</tr>
<tr>
<td>listreverse.error</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>TO</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td>loop.error</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>TO</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td>loo1.error</td>
<td>8</td>
<td>0</td>
<td>10.87</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>1</td>
</tr>
<tr>
<td>mc91.pl</td>
<td>139</td>
<td>24</td>
<td>0.57</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>1</td>
</tr>
<tr>
<td>merge</td>
<td>2</td>
<td>0</td>
<td>0.36</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>1</td>
</tr>
<tr>
<td>mergetsort.error</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td>palindrome</td>
<td>2</td>
<td>0</td>
<td>0.61</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>6</td>
</tr>
<tr>
<td>parity</td>
<td>3</td>
<td>1</td>
<td>0.62</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>7</td>
</tr>
<tr>
<td>rate_limiter.c</td>
<td>3</td>
<td>0</td>
<td>49.36</td>
<td>130</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>130</td>
</tr>
<tr>
<td>remainder</td>
<td>1</td>
<td>0</td>
<td>1.5</td>
<td>17</td>
<td>1</td>
<td>0</td>
<td>0.36</td>
<td>17</td>
</tr>
<tr>
<td>running</td>
<td>3</td>
<td>1</td>
<td>0.4</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>5</td>
</tr>
<tr>
<td>scan.error</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>TO</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td>string_concat.error</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>TO</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td>string_concat1.error</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td>string_copy.error</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>TO</td>
<td>TO</td>
<td>-</td>
<td>TO</td>
<td>-</td>
</tr>
<tr>
<td>substring.error</td>
<td>5</td>
<td>0</td>
<td>0.55</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>1</td>
</tr>
<tr>
<td>substring1.error</td>
<td>15</td>
<td>0</td>
<td>2.84</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>5</td>
</tr>
<tr>
<td>triple</td>
<td>27</td>
<td>10</td>
<td>0.36</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0.36</td>
<td>6</td>
</tr>
<tr>
<td>average (over 38)</td>
<td>8.78</td>
<td>0.93</td>
<td>9.52</td>
<td>38.64</td>
<td>0</td>
<td>0</td>
<td>0.36</td>
<td>38.64</td>
</tr>
<tr>
<td>solved/total</td>
<td>33/38</td>
<td>-</td>
<td>28/38</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

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


Interpolant Tree Automata and their Application in Horn Clause Verification


