Fixpoint-Guided Abstraction Refinements

Patrick Cousot[1], Pierre Ganty[2], and Jean-François Raskin[2]

[1] Département d'informatique, École normale supérieure
45 rue d'Ulm, 75230 Paris cedex 05, France
Patrick.Cousot@ens.fr
[2] Département d'informatique, Université Libre de Bruxelles
Campus de la Plaine, CP212, 1050 Bruxelles, Belgium
{pganty,jraskin}@ulb.ac.be

Abstract. In this paper, we present an abstract fixpoint checking algorithm with automatic refinement by backward completion in Moore closed abstract domains. We study the properties of our algorithm and prove it to be more precise than the counterexample guided abstract refinement algorithm (CEGAR). Contrary to several works in the literature, our algorithm does not require the abstract domains to be partitions of the state space. We also show that our automatic refinement technique is compatible with so-called acceleration techniques. Furthermore, the use of Boolean closed domains does not improve the precision of our algorithm. The algorithm is illustrated by proving properties of programs with nested loops.

1 Introduction

Techniques for the automatic verification of program's invariants is an active research subject since the early days of computer science. Invariant verification for a program P can be reduced to a *fixpoint checking problem*: given a monotone function $post$ over sets of program states, a set of initial states I, and a set S of states, S is an *invariant* of P if and only if the reachable states $\bigcup_{i\geqslant 0} post^i(I)$ from I that is the least fixpoint of $\lambda X.\, I \cup post(X)$ is a subset of S. We call this fixpoint the forward semantics of P.

For fundamental reasons (undecidability of the invariant checking problem for Turing complete models of computation), or for practical reasons (limitations of the computing power of computers), the forward semantics is usually not evaluated in the domain of the function $\lambda X.\, I \cup post(X)$, the so-called *concrete domain*, but in a simpler domain of values, a so-called *abstract domain*. Abstract interpretation has been proposed in [1] as a general theory to abstract fixpoint checking problems. The design of effective abstract interpretation algorithms relies on the definition of useful abstract domains and semantics. The design of good abstractions for a programming language is a difficult and time consuming tasks. Recently, research efforts [] have been devoted to find automatic techniques that are able to discover and refine abstract domains for a given program. This work proposes new results in this line.

In this paper, we propose a new *abstract algorithm* for fixpoint checking with built-in *abstract domain refinements*. The automatic refinement of abstract domains is used to improve the precision of the algorithm when it is inconclusive. Our algorithm has several properties that departs it from the existing algorithms proposed in the literature.

First, it computes not only overapproximations of least fixpoints but also *overapproximations of greatest fixpoints*. The two analyses improve each other: the current fixpoint is limited to the values that are computed by the previous fixpoint. Second, it is not bound to consider refinements related to spurious abstract counterexamples. The refinement principle that we propose is guided by the abstract fixpoint computations. Our refinement method is more robust and systematic. Third, our refinement principle is compatible with acceleration techniques: acceleration techniques can be used to discover new interesting abstract values which can be used by subsequent abstract computations. This is an important characteristic as this allows us to compute new abstract values that are useful to capture the behavior of loops. This hinders the application of the CEGAR approach. Fourth, in the abstract interpretation framework the subset of concrete values given by the abstract domain is a Moore family. Intuitively it means that the set is closed for the meet operation of the concrete lattice. This property is weaker than the property enforced by the use of partitions of the state space as in so-called *predicate abstractions*. In the paper we show that requiring the use of partitions instead of Moore families does not add power to our algorithm. If it terminates using partitions then it terminates using Moore families. Fifth we show that whenever an invariant can be proved using the CEGAR approach then our algorithm is able to prove the invariant as well. And last we show that the abstract algorithm is guaranteed to terminate under various conditions like for instance the descending chain condition on the concrete domain or if the refinement adds a value for which the concrete greatest fixpoint is computable.

Related works. In the following pages we relate our approach with the CEGAR approach (see [2]) where the refinement is done by a backward traversal of the abstract counterexample. Recently new refinement techniques based on the proof of unsatisfiability of the counterexample emerged (see [3] and the references given there). Seen differently, the refinement picks non deterministically the new values to add to the abstract domain among a set of values defined declaratively. In our case the value is unique and defined operationally. For this reason we think that an empirical comparison would make more sense.

The abstract fixpoint checking algorithm we propose is an extension of the classical combination of forward and backward static analysis in abstract interpretation ([4] as generalized by [5]) to include abstract domain completion that is the extension of the abstract domain to avoid loss of precision in abstract fixpoints. This abstract domain completion is a backward completion in the classical sense of abstract interpretation [6] but, for efficiency, restricted to reachable states stuck in the invariant to be checked. In [7] the authors define a restricted abstract domain completion. However since we reused all the information computed so far our completion is much more finer than theirs. In [8] the authors consider a set of proof rules to establish invariant properties of the system and they propose abstractions to show the premises of some rule hold. Moreover they give a method to exclude spurious counterexamples based on acceleration techniques.

Structure of the paper. The paper is organized as follows. Sect. 2 introduces some preliminary results that are useful for the rest of the paper. In Sect. 3, we present our algorithm and prove its main properties related to correctness and termination, we also

show that our approach can be easily combined with acceleration techniques. Sect. 4 compares our algorithm to the CEGAR approach and predicate abstraction. Finally let us mention that omitted proofs are in the appendix together with an illustration of the behavior of the algorithm on two representative examples (viz. heapsort and bubblesort).

2 Preliminaries

Notations and notions of lattice theory. We use Church's lambda notation (so that F is $\lambda X.\, F(X)$) and use the composition operator $\circ$ on functions given by $(f \circ g)(x) = f(g(x))$. Let X be any set and let $f \in X \mapsto X$ be a function on this set. We extend the function f to subsets in X in a natural way: given $S \subseteq X$, $f(S) = \{f(s) \mid s \in S\}$. The reflexive transitive closure f^* of a function f such that its domain and co-domain coincide is given by $\bigcup_{i \geq 0} f^i$ where f^0 is the identity and $f^{i+1} = f^i \circ f$. The reflexive transitive closure R^* of a relation R is defined in the same way. A function f on a complete lattice is said to be *additive* (resp. *coadditive*) if f distributes the join (resp. the meet) operator. Given two functions f, g on a poset $(L, \subseteq)$, we define the pointwise comparison $\dot{\subseteq}$ between functions as follows: $\lambda x.\, f(x) \;\dot{\subseteq}\; \lambda x.\, g(x)$ iff $\forall y \in (L, \subseteq): f(y) \subseteq g(y)$. Given a set S, $\wp(S)$ denote the set of all subsets of S. Sometimes we write s instead of the singleton $\{s\}$.

We denote by $\mathit{lfp}(f)$ and $\mathit{gfp}(f)$, respectively, the least and greatest fixpoint, when they exist, of a function f on a poset. The well-known Knaster-Tarski's theorem states that any monotone function $f \in L \mapsto L$ on a complete lattice $\langle L, \leqslant, \wedge, \vee, \top, \bot \rangle$ admits a least fixpoint and the following characterization holds: $\mathit{lfp}(f) = \bigwedge \{x \in L \mid f(x) \leqslant x\}$. Dually, f also admits a greatest fixpoint and the following characterization holds: $\mathit{gfp}(f) = \bigvee \{x \in L \mid x \leqslant f(x)\}$.

Transition systems and predicate transformers. A *transition system* is a 3-uple $\mathcal{T} = (C, I, T)$ where C is the set of *states*, $I \subseteq C$ is the subset of *initial states*, and $T \subseteq C \times C$ is the *transition relation*. Often, we write $s \rightarrow s'$ if $(s, s') \in T$, $s \rightarrow^* s'$ if $(s, s') \in T^*$ and $s \rightarrow^k s'$ if $(s, s') \in T^k$ for $k \in \mathbb{N}$.

To manipulate sets of states, we use *predicate transformers*. The *forward image operator* is a function that given a relation $T' \subseteq C \times C$ and a set of states $C' \subseteq C$, returns the set $\mathit{post}[T'](C') = \{c' \in C \mid \exists c \in C' : (c, c') \in T'\}$. When the forward image is used with the transition relation T, it is called the *post operator* and it returns, given a set of states C' all its one step successors in the transition system, we simply write it $\mathit{post}(C')$. The *backward image operator* is a function given a relation $T' \subseteq C \times C$ and set of states $C' \subseteq C$, returns the set $\widetilde{\mathit{pre}}[T'](C') = \neg \mathit{pre}[T'](\neg C') = \neg \mathit{post}[T'^{-1}](\neg C') = \{c \in C \mid \forall c' : (c, c') \in T' \Rightarrow c' \in C'\}$. When the backward image operator is used with the transition relation T, it is called the *unavoidable operator* and it returns, given a set of states C' all the states which have all their successors in the set C', we simply write it $\widetilde{\mathit{pre}}(C')$.

Given a set I of states the set of *reachable states* is given by the following least fixpoint $\mathit{lfp}^{\subseteq} \lambda X.\, I \cup \mathit{post}[T](X)$. As shown in [9], this fixpoint coincides with $\mathit{post}[T^*](I)$ also written $\mathit{post}^*(I)$ when the transition relation is clear from the context. So a state

s is said to be reachable if $s \in post^*(I)$. Dually, given a set S of states, the set of states that *are stuck in* S (or also that *cannot escape from* S) is given by the following greatest fixpoint $gfp^{\subseteq}\lambda X.\, S \cap \widetilde{pre}[T](X)$. As shown in [9], this fixpoint coincides with $\widetilde{pre}[T^*](S)$ also written $\widetilde{pre}^*(S)$ when the transition relation is clear from the context.

Given two sets I, Z of states we call $lfp^{\subseteq}\lambda X.\, (I \cup post(X)) \cap Z$ the set of *reachable states within* Z. Finally given a set S of states, the set of *states that cannot escape from* S *in less than* 1 *steps* is given by $S \cap \widetilde{pre}(S)$.

Abstract interpretation. We use abstract interpretation to abstract the semantics of transition systems. We assume standard abstract interpretation where, *concrete* and *abstract domains*, L given by $\wp(C)$ and A, are Boolean complete lattice $\langle L, \subseteq, \cap, \cup, C, \emptyset, \neg\rangle$ and complete lattice $\langle A, \sqsubseteq, \sqcap, \sqcup, \top_A, \bot_A\rangle$, respectively. The two lattices are related by abstraction and concretization maps α and γ forming a *Galois connection* $\forall c \in L : \forall a \in A : \alpha(c) \sqsubseteq a \Leftrightarrow c \subseteq \gamma(a)$ [4]. We write this fact as follows: $\langle L, \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle A, \sqsubseteq\rangle$, or simply $\underset{\alpha}{\overset{\gamma}{\leftrightarrows}}$ when the concrete and abstract domains are clear from the context. In this paper, we use a family of finite abstract domains that are subset of A.

Definition 1 (Family of abstract domains). *Let $\{A_i\}_{i\in I}$ be a family of finite sets such that: (i) $A = \bigcup_{i\in I} A_i$, (ii) $\langle A_i, \sqsubseteq\rangle$ is a complete lattice, and (iii) $\exists \alpha_i \colon \langle L, \subseteq\rangle \underset{\alpha_i}{\overset{\gamma}{\leftrightarrows}} \langle A_i, \sqsubseteq\rangle$.*

Given an abstract domain $\mathcal{A}_i$, we write $\gamma(\mathcal{A}_i)$ for the subset of concrete sets $X \in L$ that can be represented by abstract values in A_i.

The set $\gamma(\mathcal{A}_i) \subseteq L$ of concrete values that the abstract domain represents must be closed by intersection if there is a Galois connection between $\mathcal{A}_i$ and L. Our abstract domains are thus Moore closed. This notion, and the stronger notion of Boolean closure are defined as follows.

Definition 2 (Moore and Boolean closure). *A finite subset $X \subseteq L$ is said to be:*

- Moore closed *iff* $\forall x_1, x_2 \in X : x_1 \wedge x_2 \in X$ *and X contains the topmost element of L. We define the function $\lambda X.\, \mathcal{M}(X)$ which returns the Moore closure of its argument, i.e. the smallest set $M \subseteq L$ such that $X \subseteq M$ and M is Moore closed*[3].
- Boolean closed *iff* $\forall x_1, x_2 \in X$: (i) $x_1 \wedge x_2 \in X$, (ii) $x_1 \vee x_2 \in X$, *and* (iii) $C \setminus x \in X$. *We define the function $\lambda X.\, \mathcal{B}(X)$ which returns the Boolean closure of its argument, i.e. the smallest set B such that $X \subseteq B$ and B is Boolean closed.*

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of predicates and let $[\![p_i]\!] \subseteq C$ be the subset of states that satisfy the predicate p_i. The set of predicates P implicitly defines a Boolean closed abstract domain, noted $\mathcal{A}_P$, such that $\gamma(\mathcal{A}_P) \subseteq L$ which is the smallest set which is Boolean closed and contains the sets $\{[\![p]\!] \mid p \in P\}$, i.e. $\mathcal{A}_P = \mathcal{B}(\{[\![p]\!] \mid p \in P\})$. The elements of $\mathcal{A}_P$ are equivalent to propositional formulas build from the predicates in P. Elements of $\mathcal{A}_P$ can also be viewed as union of equivalence classes of states: two states $c_1, c_2 \in C$ are equivalent whenever they satisfy exactly the same subset of predicates in P.

[3] A Moore closed set is also called a Moore family.

The following Lemma contains well-known results of abstract interpretation that we recall here so that the paper is self contained. We refer the interested reader to [10] and the references given there for more details.

Lemma 1. *Let $I, S, Z \in L$ be sets of states. Given an abstract domain A_i, we define $\mathcal{R}$, resp. $\mathcal{S}$, be the abstract forward, resp. backward, semantics on A_i as $lfp^{\sqsubseteq}\lambda X.\, \alpha_i((I \cup post(\gamma(X))) \cap Z)$, resp. $gfp^{\sqsubseteq}\lambda X.\, \alpha_i(S \cap \widetilde{pre}(\gamma(X)))$.*

$$\left.\begin{array}{r} lfp^{\subseteq}\lambda X.\, (I \cup post(X)) \cap Z \subseteq \gamma(\mathcal{R}) \\ gfp^{\subseteq}\lambda X.\, S \cap \widetilde{pre}(X) \subseteq \gamma(\mathcal{S}) \end{array}\right\}$$

We call this inclusion the overapproximation of the abstract semantics.

The Fixed point Checking Problem. Given a transition system $\mathcal{T} = (C, I, T)$ and $S \subseteq C$ a set of states. The *fixpoint checking problem* asks if $lfp^{\subseteq}\lambda X.\, I \cup post(X) \subseteq S$

3 Abstract Fixed-point Checking Algorithm

3.1 The algorithm

Alg. 1 has been inspired and is a generalization of what we have done previously in [11–13]. We review here its main characteristics.

Algorithm 1: The algorithm

Data: An instance of the fixpoint checking problem such that $I \subseteq S$ and an abstract domain $\mathcal{A}_0$ such that $S \in \gamma(\mathcal{A}_0)$

$Z_0 = S$
for $i = 0, 1, 2, 3, \ldots$ **do**
  Compute $\mathcal{R}_i$ given by $lfp^{\sqsubseteq}\lambda X.\, \alpha_i\Big((I \cup post(\gamma(X))) \cap Z_i\Big)$
  if $\alpha_i(I \cup post(\gamma(\mathcal{R}_i))) \sqsubseteq \alpha_i(Z_i)$ **then**
    return *OK*
  else
    Compute $\mathcal{S}_i$ given by $gfp^{\sqsubseteq}\lambda X.\, \alpha_i\Big(\gamma(\mathcal{R}_i) \cap \widetilde{pre}(\gamma(X))\Big)$
    if $\alpha_i(I) \sqsubseteq \mathcal{S}_i$ **then**
      Let $Z_{i+1} = \gamma(\mathcal{S}_i) \cap \widetilde{pre}(\gamma(\mathcal{S}_i))$
      Let $\mathcal{A}_{i+1}$ be s.t. $\gamma(\mathcal{A}_{i+1}) = \mathcal{M}\big(\{Z_{i+1}\} \cup \gamma(\mathcal{A}_i)\big)$
    else
      return *KO*
    end
  end
end

It computes overapproximations of *least* and *greatest* fixpoints. Line 3 computes an abstract least fixpoint. As we will see in Prop. 1, when executed on a positive instance of the fixpoint checking problem, every set $\gamma(\mathcal{R}_i)$ overapproximates the reachable states of the transition system. Line 7 computes an abstract greatest fixpoint. As we will see in Lem. 2, and Lem. 3, $\gamma(\mathcal{S}_i)$ underapproximates the set of states that cannot escape from S in less than $i+1$ steps. As we can see from line 3 and line 7, the two fixpoints share all the information that has been computed so far. In fact the abstract least fixpoint of line 3 overapproximates the reachable states within Z_i which gathers all the information computed so far. Similarly, the abstract greatest fixpoint of line 7 starts with the least fixpoint computed previously. Parts of the state space that have already been proved unreachable within S or stuck in S are not explored during the next iterations.

The refinement that we propose is applied on the entire abstract fixpoint and is not bound to individual counterexamples. The value Z_i contains states that cannot escape from $\gamma(S_i)$ in one step, all concrete states that are stuck within S have this property. So, this set is interesting as it adds information about concrete states in the abstract domain, this information will be used by subsequent abstract fixpoint computation. We will see later in the paper that line 9 can be modified in a way to incorporate information computed by acceleration techniques. The results that we first prove with line 9 are still valid when accelerations are used. The possibility of combining our algorithm with acceleration techniques is very interesting as accelerations may allow to discover interesting abstract values related to loops in programs. Loops usually hinder the application of the CEGAR approach.

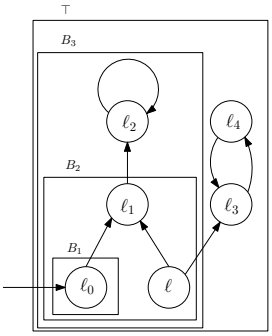
In line 10 we see that the new value Z_{i+1} computed at line 9 is added to the set of values the current abstract domain $\mathcal{A}_i$ can represent (this set is $\gamma(\mathcal{A}_i)$). The new abstract domain is given by $\mathcal{A}_{i+1}$. It is worth pointing that we actually add more than the single value Z_{i+1} to the abstract domain since working in the framework of abstract interpretation requires that $\gamma(\mathcal{A}_{i+1})$ is a Moore family. We will see later that Moore closure is sufficiently powerful in the following precise sense: considering the Boolean closure instead does not improve the precision of our algorithm. This interesting result is established in Th. 2. This contrasts with several approaches in the literature that use predicate abstraction which induce more complex Boolean closed domains. The most precise abstract $post$ operation is usually more difficult to compute on Boolean closed domains.

Our algorithm also enjoys nice termination properties. Prop. 6 shows that our algorithm terminates whenever the concrete domain enjoys the descending chain condition. This result allows us to conclude that our algorithm will always terminate for the important class of Well-structured transition systems [14, 15], see [12, 13] for the details. Th. 1 of Sect. 4 also shows that whenever CEGAR terminates, then our algorithm terminates. We also establish in Prop. 5 that whenever our algorithm is submitted a negative instance, it always terminates.

Finally it is worth pointing out that all the operations in the algorithm, with the exception of the refinement operation of line 9, are abstract operations, and the only concrete operation is used outside of any of the fixpoint computations.

Before giving a formal characterization of Alg. 1, let us give more insights by running the algorithm on a toy example.

Example 1. The toy example is a finite state system given at Fig. 1. The set of states given by the initial abstract domain are given by the boxes. We submit to our algorithm the following positive instance of the fixpoint checking problem where $\mathcal{A}_0 = \{B_1, B_2, B_3, \top\}$, $I = \{\ell_0\}$, and $S = \gamma(B_3)$. So note that $Z_0 = \gamma(B_3) = S$. In the right side of Fig. 1 the algorithm is executed step by step. Since the fixpoints converge in very few steps we invite the interested reader to verify them by hand.



$\mathcal{R}_0 = B_3$	line 3
$\alpha_0(I \cup post(\gamma(B_3))) \not\sqsubseteq Z_0$ (so not "OK")	line 4
$\mathcal{S}_0 = B_3$	line 7
$\alpha_0(I) \sqsubseteq \mathcal{S}_0$ (cannot say "KO")	line 8
$Z_1 = \gamma(B_3) \cap \widetilde{pre}(\gamma(B_3))$	line 9
$= \{\ell_0, \ell_1, \ell_2\}$	
The new domain is $\mathcal{A}_1 = \mathcal{A}_0 \cup \{B_4, B_5\}$	line 10
with $\gamma(B_4) = Z_1$ and $\gamma(B_5) = Z_1 \cap \gamma(B_2)$	
$= \{\ell_0, \ell_1\}$	
$\mathcal{R}_1 = B_4$	line 3
$\alpha_1(I \cup post(\gamma(B_4))) \sqsubseteq Z_1$	line 4
Alg. 1 terminates saying "OK"	

Fig. 1. A finite state system and the result of evaluating Alg. 1 on it.

3.2 Correctness of the algorithm

In what follows we assume that Alg. 1 reaches enough iteration to compute the sets appearing in the statements. For instance, if $\gamma(\mathcal{R}_i)$ appears in the statement then the algorithm has not yet concluded at iteration $i-1$ or if Z_{i+1} appears in the statement then the algorithm has not yet concluded at iteration i.

We start with a technical lemma that states that our algorithm computes sets of states that are decreasing.

Lemma 2. *In Alg. 1 we have*

$$Z_{i+1} \subseteq \gamma(\mathcal{S}_i) \subseteq \gamma(\mathcal{R}_i) \subseteq Z_i \subseteq \cdots \subseteq Z_1 \subseteq \gamma(\mathcal{S}_0) \subseteq \gamma(\mathcal{R}_0) \subseteq Z_0 \subseteq S \ .$$

The next proposition characterizes the sets of states that are computed by the Algorithm in the presence of positive instances.

Proposition 1. *In Alg. 1,* $\forall i \in \mathbb{N}$ *if* $post^*(I) \subseteq S$ *then* $post^*(I) \subseteq \gamma(\mathcal{R}_i)$.

Proof. Our proof is by induction on i.

Base case. Lem. 1 tells us that $\gamma(\mathcal{R}_0)$ overapproximates the following least fixpoint $lfp^{\subseteq}\lambda X.\ (I \cup post(X)) \cap S$. Provided the system respects the invariant S (i.e. $post^*(I) \subseteq S$), this fixpoint is equal to $lfp^{\subseteq}\lambda X.\ (I \cup post(X))$. So, $post^*(I) \subseteq \gamma(\mathcal{R}_0)$.

Inductive case. By induction hypothesis, the property is true for $i-1$. Suppose that there exists $s \in post^*(I)$ and $s \notin \gamma(\mathcal{R}_i)$. We recall Lem. 2 which shows that $\gamma(\mathcal{R}_{i-1}) \supseteq \gamma(\mathcal{S}_{i-1}) \supseteq Z_i \supseteq \gamma(\mathcal{R}_i)$. We now consider several cases.

1. $s \notin \gamma(\mathcal{R}_{i-1})$. Then by induction hypothesis, $post^*(I) \not\subseteq S$ and we are done.
2. $s \in \gamma(\mathcal{R}_{i-1})$ and $s \notin \gamma(\mathcal{S}_{i-1})$. We conclude from Lem. 1 that $\gamma(\mathcal{S}_{i-1})$ overapproximates the states stuck in $\gamma(\mathcal{R}_{i-1})$. Since $s \notin \gamma(\mathcal{S}_{i-1})$ there exists a state s' such that $s \rightarrow^* s'$ and $s' \notin \gamma(\mathcal{R}_{i-1})$. First, note that as $s \in post^*(I)$, we conclude that $s' \in post^*(I)$. But as $s' \notin \gamma(\mathcal{R}_{i-1})$, we know that $post^*(I) \not\subseteq \gamma(\mathcal{R}_{i-1})$ and by induction hypothesis we conclude that $post^*(I) \not\subseteq S$.
3. $s \in \gamma(\mathcal{R}_{i-1})$, $s \in \gamma(\mathcal{S}_{i-1})$ and $s \notin Z_i$. We conclude from the definition of Z_i which is given by $\gamma(\mathcal{S}_{i-1}) \cap \widetilde{pre}(\gamma(\mathcal{S}_{i-1}))$ that there exists $s' \notin \gamma(\mathcal{S}_{i-1})$ such that $s \rightarrow s'$. Either $s' \notin \gamma(\mathcal{R}_{i-1})$ or $s' \in \gamma(\mathcal{R}_{i-1})$ and by the previous case, we know that $s' \rightarrow^* s''$ and $s'' \notin \gamma(\mathcal{R}_{i-1})$. In the two cases, we conclude that $post^*(I) \not\subseteq \gamma(\mathcal{R}_{i-1})$ and by induction hypothesis that $post^*(I) \not\subseteq S$.
4. $s \in \gamma(\mathcal{R}_{i-1})$, $s \in \gamma(\mathcal{S}_{i-1})$, $s \in Z_i$, and $s \notin \gamma(\mathcal{R}_i)$. By overapproximation of the abstract semantics, we know that s is not reachable from I within Z_i. Otherwise stated, all paths starting form I and ending in s leaves Z_i. As s is reachable from I, we know that there exists some $s' \notin Z_i$ which is reachable form I. We can apply the same reasoning as above and conclude that $post^*(I) \not\subseteq S$. □

We are now in position to prove that, when the algorithm terminates and returns OK, it has been submitted a positive instance of the fixpoint checking problem, and when the algorithm terminates and returns KO, it has been submitted a negative instance of the fixpoint checking problem.

Proposition 2 (Correctness – positive instances). *If Alg. 1 says "OK" then we have* $post^*(I) \subseteq S$.

Proof.

Algorithm says "OK"	
$\Leftrightarrow \alpha_i(I \cup post(\gamma(\mathcal{R}_i))) \sqsubseteq \alpha_i(Z_i)$	line 4
$\Leftrightarrow \alpha_i(I) \sqsubseteq \alpha_i(Z_i)\ \&\ \alpha_i \circ post \circ \gamma(\mathcal{R}_i) \sqsubseteq \alpha_i(Z_i)$	α_i additivity
$\Leftrightarrow I \sqsubseteq \gamma \circ \alpha_i(Z_i)\ \&\ post(\gamma(\mathcal{R}_i)) \sqsubseteq \gamma \circ \alpha_i(Z_i)$	$\frac{\gamma}{\alpha_i}$
$\Leftrightarrow I \subseteq Z_i\ \&\ post(\gamma(\mathcal{R}_i)) \subseteq Z_i$	$Z_i \in \gamma(\mathcal{A}_i)$ line 10

Then,

$$
\begin{array}{lr}
\alpha_i((I \cup post(\gamma(\mathcal{R}_i))) \cap Z_i) \sqsubseteq \mathcal{R}_i & \text{def. of } \mathcal{R}_i\text{, prop. of } \mathit{lfp} \\
\Leftrightarrow (I \cup post(\gamma(\mathcal{R}_i))) \cap Z_i \subseteq \gamma(\mathcal{R}_i) & \stackrel{\gamma}{\underset{\alpha_i}{\leftrightarrows}} \\
\Rightarrow I \cup post(\gamma(\mathcal{R}_i)) \subseteq \gamma(\mathcal{R}_i) & I \subseteq Z_i \,\&\, post(\gamma(\mathcal{R}_i)) \subseteq Z_i \\
\Rightarrow \mathit{lfp}^{\subseteq} \lambda X.\, I \cup post(X) \subseteq \gamma(\mathcal{R}_i) & \text{prop. of } \mathit{lfp} \\
\Rightarrow post^*(I) \subseteq S & \gamma(\mathcal{R}_i) \subseteq S \text{ by Lem. 2} \quad \square
\end{array}
$$

Proposition 3 (Correctness – negative instances). *If Alg. 1 says "KO" then we have* $post^*(I) \not\subseteq S$.

Proof. If at iteration i the algorithm says "KO" then we find that $\alpha_i(I) \not\sqsubseteq \mathcal{S}_i$ (line 8) which is equivalent to $I \not\subseteq \gamma(\mathcal{S}_i)$ by $\stackrel{\gamma}{\underset{\alpha_i}{\leftrightarrows}}$. We conclude from Lem. 2 that $\gamma(\mathcal{R}_{i+1}) \subseteq \gamma(\mathcal{S}_i)$, hence that $I \not\subseteq \gamma(\mathcal{R}_{i+1})$ and finally that $post^*(I) \not\subseteq S$ using the contrapositive of Prop. 1. $\square$

Remark 1. The proofs of the above results remain correct if in line 9 of Alg. 1 instead of $\lambda X.\, \widetilde{pre}[T](X)$ we take $\lambda X.\, \widetilde{pre}[R](X)$ where $T \subseteq R \subseteq T^*$. This property will be used later when we propose alternative refinement operations based on acceleration techniques.

3.3 Termination of the Algorithm

To reason about the termination of the algorithm, we need the following technical proposition and its corollary.

Proposition 4. *In Alg. 1 the following holds:*

1. *if* $Z_{i+1} = Z_i$ *then* $post(Z_i) \subseteq Z_i$*;*
2. *if* $I \not\subseteq Z_i$ *then the algorithm terminates at iteration* i *and returns "KO";*
3. *if* $I \cup post(Z_i) \subseteq Z_i$ *then the algorithm terminates at iteration* i *and return "OK".*

Corollary 1. *In Alg. 1 if* $Z_i = Z_{i+1}$ *then the algorithm terminates.*

Alg. 1 terminates when submitted a negative instance as proved below in Lem. 3 and Prop. 5.

Lemma 3. *In Alg. 1,* $\gamma(\mathcal{R}_i)$ *underapproximates the set* $\widetilde{pre}[\bigcup_{j=0}^{i} T^j](S)$ *of states which cannot escape from* S *in less than* $i+1$ *steps.*

Proof. The result is shown by induction on the number i of steps. For the base case, Lem. 2 shows that $\gamma(\mathcal{R}_0) \subseteq S = \widetilde{pre}[T^0](S)$. For the inductive case,

$$\begin{aligned}
&\widetilde{pre}[\bigcup_{j=0}^{i+1} T^j](S) = \widetilde{pre}[\bigcup_{j=0}^{i} T^j \cup \bigcup_{j=1}^{i+1} T^j](S) && \text{def. } \cup \\
&= \widetilde{pre}[\bigcup_{j=0}^{i} T^j](S) \cap \widetilde{pre}[T](\widetilde{pre}[\bigcup_{j=0}^{i} T^j](S)) && \text{def. } \widetilde{pre} \\
&\supseteq \gamma(\mathcal{R}_i) \cap \widetilde{pre}[T](\gamma(\mathcal{R}_i)) && \text{ind. hyp.} \\
&\supseteq \gamma(\mathcal{S}_i) \cap \widetilde{pre}[T](\gamma(\mathcal{S}_i)) && \text{by Lem. 2} \\
&= Z_{i+1} && \text{by line 9} \\
&\supseteq \gamma(\mathcal{R}_{i+1}) && \text{by Lem. 2} \qquad \square
\end{aligned}$$

Proposition 5. *If $post^*(I) \not\subseteq S$ then Alg. 1 terminates.*

Proof. Hypothesis shows that there exists states s, s' and a value $k \in \mathbb{N}$ such that $s \in I$, $s' \notin S$ and $s \rightarrow^k s'$. Lem. 3 shows that $\gamma(\mathcal{R}_{k-1}) \subseteq \bigcap_{j=0}^{k} \widetilde{pre}[T^j](S)$. So we conclude from above that $I \not\subseteq \bigcap_{j=0}^{k} \widetilde{pre}[T^j](S)$, hence that $I \not\subseteq \gamma(\mathcal{R}_{k-1})$ by transitivity and finally that $I \not\subseteq Z_k$ by Lem. 2. The last step uses Prop. 4.2 to show that the algorithm terminates. $\square$

The following proposition states that our algorithm terminates under the descending chain condition in the concrete domain.

Proposition 6. *Assuming the descending chain condition holds on $\langle L, \subseteq \rangle$ then Alg. 1 terminates.*

Proof. We prove the contrapositive. Assume the algorithm does not terminate. We thus obtain that $Z_0 \supset Z_1 \supset \cdots \supset Z_n \supset \cdots$ by Cor. 1 and Lem. 2 which contradicts the descending chain condition. $\square$

Below Prop. 7 establishes a stronger termination result of our algorithm which states that if the algorithm computes a value Z_i from which the evaluation of the greatest fixpoint $gfp^{\subseteq}\lambda X.\, Z_i \cap \widetilde{pre}(X)$ terminates after a finite number of iterations then our algorithm terminates. We use classical fixpoint evaluation techniques to compute the set $gfp^{\subseteq}\lambda X.\, Z_i \cap \widetilde{pre}(X)$. First we start with the set Z_i and then we remove the states escape from Z_i in 1 step. The set obtained is formally given by $Z_i \cap \widetilde{pre}(Z_i)$. Then we iterate this process until no state is removed.

Proposition 7. *If in Alg. 1 there is a value for i such that $gfp^{\subseteq}\lambda X.\, Z_i \cap \widetilde{pre}(X)$ stabilizes after a finite number of step, then Alg. 1 terminates.*

3.4 Termination of the Algorithm Enhanced by Acceleration Techniques

In this section we will study a enhancement of Alg. 1 which relies on acceleration techniques (see [16] and the references given there). Roughly speaking, acceleration

techniques allows us to compute underapproximations of the transitive closure of some binary relation as, for instance the transition relation. We refer the interested reader to the extensive literature on this topic.

Assume we are given some binary relation R such that $T \subseteq R \subseteq T^*$. The enhancement we propose replaces line 9 (viz. $Z_{i+1} = \gamma(\mathcal{S}_i) \cap \widetilde{pre}[T](\gamma(\mathcal{S}_i))$) by the following: $Z_{i+1} = \gamma(\mathcal{S}_i) \cap \widetilde{pre}[R](\gamma(\mathcal{S}_i))$. The definition of R suggests that the value added using R should be at least as precise as the one given using T. A very favorable situation is when R equals T^* but Prop. 7 is not applicable at any iteration. We conclude from $Z_1 = gfp^{\subseteq}\lambda X.\, \gamma(\mathcal{S}_0) \cap \widetilde{pre}(X)$ that $post(Z_1) \subseteq Z_1$ by $\xleftarrow[post]{\widetilde{pre}}\!\!\rightarrow$, hence that the enhanced algorithm terminates at iteration one by Prop. 4 while the normal algorithm might not since Prop. 7 is never applicable. Below we illustrate this situation using a toy example.

Example 2. Fig. 2 shows a two counters automaton and its associated semantics. The domain of the counters is the set of integers. In the automaton x, y refer to the current value of the counters while x', y' refer to the next value (namely the value after firing the transition). Transition t_1 is given by a simultaneous assignment. In green are the reachable states, which are given by $\{(x, y) \mid y \leqslant x \,\&\, 0 \leqslant x\}$. We will submit to Alg. 1 a positive instance of the fixpoint checking problem such that I and S are given by $\{(0, 0)\}$ and $\{(x, y) \mid y \neq x + 1\}$ respectively. Our initial abstract domain $\mathcal{A}_0$ is such that $\gamma(\mathcal{A}_0) = \mathcal{M}(S)$.

It is routine to check $\mathcal{R}_0$, computed at line 3, is such that $\gamma(\mathcal{R}_0) = S$, hence that the test of line 4 fails. It follows that we have to compute $\mathcal{S}_0$ given at line 7. Let X^δ, δ be the sequence of iterates for $\lambda X.\, \alpha_0(\gamma(\mathcal{R}_0) \cap \widetilde{pre}[T](\gamma(X)))$ which converges to $\mathcal{S}_0$. First let us compute

$$
\begin{aligned}
& S \cap \widetilde{pre}[t_2](S) \\
&= S \cap \neg \circ pre[t_2] \circ \neg(S) && \text{def. of } \widetilde{pre} \\
&= S \cap \neg \circ pre[t_2](\{(x, y) \mid y = x + 1\}) && \text{def. of } \neg, S \\
&= S \cap \neg(\{(x, y) \mid y = x + 2\}) && \text{see Fig. 2} \\
&= S \cap \{(x, y) \mid y \neq x + 2\} \\
&= \{(x, y) \mid y \neq x + 1\} \cap \{(x, y) \mid y \neq x + 2\} && \text{def. of } S
\end{aligned}
$$

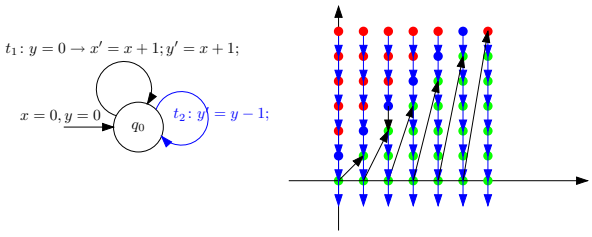


Fig. 2. A two counters automata and its associated semantics.

We now turn to the evaluation of the gfp.

$$
\begin{aligned}
X^0 &= \top \\
X^1 &= \alpha_0(\gamma(\mathcal{R}_0) \cap \widetilde{pre}[T](\gamma(X_0))) \\
&= \alpha_0(S) && \gamma(\mathcal{R}_0) = S, \top \subseteq \widetilde{pre}[T](\top) \\
&= S && S \in \gamma(\mathcal{A}_0) \\
X^2 &= \alpha_0(S \cap \widetilde{pre}[T](\gamma(X_1))) \\
&= \alpha_0\big(S \cap \widetilde{pre}[t_1](\gamma(X_1)) \cap \widetilde{pre}[t_2](\gamma(X_1))\big) && \text{def. } \widetilde{pre} \\
&= \alpha_0\big(S \cap \widetilde{pre}[t_2](\gamma(X_1))\big) && S \cap \widetilde{pre}[t_1](S) = S
\end{aligned}
$$

By above we find that $\alpha_0(S \cap \widetilde{pre}[t_2](S)) = S$, hence that $\gamma(\mathcal{S}_0) = S$. Since the test of line 8 succeeds the next step (line 9) is to compute Z_1. We use acceleration techniques to compute Z_1 for otherwise the algorithm does not converge. Without resorting to acceleration techniques each Z_i escape from S in $i+1$ steps by firing transition t_2. This clearly indicates that the CEGAR approach considers counterexamples of increasing length and thus fail on this toy example. By considering the limit instead of the Z_i's we obtain a value that is stuck in S. That value stuck in S can obtained using acceleration techniques as shown below.

Our candidate relation to show termination is given by $t_1 \cup t_2^*$ which is computable using acceleration technique. It is routine to check that $T \subseteq t_1 \cup t_2^* \subseteq T^*$. Let us compute Z_1 which is given by $S \cap \widetilde{pre}[t_1 \cup t_2^*](S)$.

$$
\begin{aligned}
S \cap \widetilde{pre}[t_1 \cup t_2^*](S) &= && \text{def. } \widetilde{pre} \\
S \cap \widetilde{pre}[t_1](S) \cap \widetilde{pre}[t_2^*](S) &= && S \cap \widetilde{pre}[t_1](S) = S \\
\widetilde{pre}[t_2^*](S) &= \\
gfp^{\subseteq} \lambda X.\, S \cap \widetilde{pre}[t_2](X)
\end{aligned}
$$

The latter fixpoint evaluates to $\{(x, y) \mid y \leqslant x\}$ (details are given in the appendix) and so the new abstract domain $\mathcal{A}_1$ is such that $\gamma(\mathcal{A}_1) = \mathcal{M}(\gamma(\mathcal{A}_0) \cup Z_1)$. At iteration 1, we find at line 3 that $\gamma(\mathcal{R}_1) = Z_1$, hence that the test of line 4 succeeds since there is no outgoing transition of Z_1 (see Fig. 2), and finally that Alg. 1 terminates with the right answer.

It is worth pointing that the forward abstract semantics is conclusive. However algorithms using acceleration techniques to compute the forward concrete semantics do not terminate. Basically acceleration techniques identify regular expressions over the transition alphabet and then compute underapproximation of the transitive closure of the transition relation. For the automaton of Fig. 2 acceleration techniques fail because there is no finite regular expression that describes all the possible executions of the counter automaton. Additional examples can be found in the appendix. ■

The rest of this section is devoted to establish some termination properties of the enhanced algorithm. In fact, as we said in Rem. 1 our correctness proofs remains valid for the enhancement. Thus below we focus on termination properties.

By definition of R it is routine to check that

$$\lambda X.\, \widetilde{pre}[T^*](X) \mathrel{\dot\subseteq} \lambda X.\, \widetilde{pre}[R](X) \mathrel{\dot\subseteq} \lambda X.\, \widetilde{pre}[T](X) \ . \tag{1}$$

Proposition 8. *Let R_2 such that $T \subseteq R_2 \subseteq T^*$ and gfp$^{\subseteq}\lambda X.\, S \cap \widetilde{pre}[R_2](X)$ stabilizes after a finite number of step, then Alg. 1 when using any R_1 such that $R_2 \subseteq R_1 \subseteq T^*$ at line 9 terminates as well.*

Remark 2. In Alg. 1 the fixpoint $\mathcal{R}_i$ and $\mathcal{S}_i$ are defined according to the best approximation of the predicate transformer $post$ and $\widetilde{pre}$, respectively. For various reasons we may be constrained to use a less precise approximation. In this context provided Lem. 2 holds all the result of Sect. 3.2, 3.3 and 3.4 remain valid.

4 Relationships with Other Approaches

4.1 Counterexample Guided Abstraction Refinement

We first recall here the main ingredients of the CEGAR approach [17, §4.2]. Given a transition system $\mathcal{T} = (C, T, I)$, called the *concrete transition system*, and a partition of C into a finite number of equivalence classes $\mathcal{C} = \{C_1, \ldots C_k\}$, the abstract transition system is a transition system $\mathcal{T}^\alpha = (C^\alpha, T^\alpha, I^\alpha)$ where:

- $C^\alpha = \mathcal{C}$, i.e. abstract states are the equivalence classes;
- $T^\alpha = \{(C_i, C_j) \mid \exists c \in C_i, c' \in C_j \colon (c, c') \in T\}$, i.e. there is a transition from an equivalence class C_i to an equivalence class C_j whenever there is a state of C_i which has a successor in C_j by the transition relation;
- $I^\alpha = \{C_i \in \mathcal{C} \mid C_i \cap I \neq \emptyset\}$, i.e. a class is initial whenever it contains an initial state.

A path in the abstract transition system is a finite sequence of abstract states related by T^α that starts in an initial state. An abstract state C_i is reachable if there exists a path in $\mathcal{T}^\alpha$ that ends in C_i. The set of states within the equivalence classes that are reachable in the abstract transition system, is an overapproximation of the reachable states in the concrete transition system.

An abstract counterexample to $S \subseteq C$ is a path $C_{i_1} C_{i_2} \ldots C_{i_n}$ in the abstract transition system such that $C_{i_n} \not\subseteq S$. An abstract counterexample is *spurious* if it does not match a concrete path in $\mathcal{T}$. We define this formally as follows. To an abstract counterexample $C_{i_1}, \ldots, C_{i_n}$, we associate a sequence $t_1, t_2, \ldots, t_{n-1}$ of subsets of T (the transition relation of $\mathcal{T}$) such that $t_j = T \cap (C_{i_j} \times C_{i_{j+1}})$ (the projection of T on successive classes).

An abstract counterexample is an *error trace*, only if $I \not\subseteq \widetilde{pre}[t_1 \circ \ldots \circ t_{n-1}](S)$ (by monotonicity we have $I \not\subseteq \widetilde{pre}[T^*](S)$), otherwise it is called *spurious* and, so $I \subseteq \widetilde{pre}[t_1 \circ \ldots \circ t_{n-1}](S)$. Eliminating a spurious counterexample is done by splitting a class C_j where $1 \leqslant j \leqslant n$. The class C_j contains *bad states* (written bad) that can reach $\neg S$ but which are not reachable from C_{j-1}. Accordingly the class C_j split in $C_j \cap \mathrm{bad}$ and $C_j \cap \neg\mathrm{bad}$. From the above definition, we can deduce that $\mathrm{bad} = pre[t_j \circ \ldots \circ t_{n-1}](\neg S)$, hence that $\neg\mathrm{bad} = \neg \circ pre[t_j \circ \ldots \circ t_{n-1}] \circ \neg(S)$, and, finally that $\neg\mathrm{bad} = \widetilde{pre}[t_j \circ \ldots \circ t_{n-1}](S)$. Hence the splitting of C_j is given by $C_j \cap \widetilde{pre}[t_j \circ \ldots \circ t_{n-1}](S)$ and $C_j \cap \neg \circ \widetilde{pre}[t_j \circ \ldots \circ t_{n-1}](S)$. When the spurious counterexample has been removed, by splitting an equivalence class, a new

abstract transition transition system, based on the refined partition, is considered and the method is iterated.

CEGAR approach concludes when it either finds an error trace (identifying a negative instance of the fixpoint checking problem) or when it does not find any new abstract counter example (identifying a positive instance of the fixed point problem).

We now relate the abstract model used by CEGAR with the abstract interpretation of the system. The initial abstract domain A_0, that our algorithm uses, is such that for all equivalence classes C_i in the initial partition used by the CEGAR algorithm, there exists an abstract value $a \in A_0$ such that $\gamma(a) = C_i$.

Lemma 4. *Assume that CEGAR terminates on a positive instance of the fixpoint checking problem. So CEGAR produced a finite set $\{w_i\}_{i\in I}$ of counterexamples such that the following holds:*

$$\exists A \in \gamma(\mathcal{A}_0) \colon I \subseteq \underbrace{gfp^{\subseteq}\lambda X.\, A \cap \widetilde{pre}(X)}_{V} \subseteq S \;\&\; V = A \cap \bigcap_{i\in I} \widetilde{pre}[w_i](S) \ .$$

We need one more auxiliary result before presenting Th. 1.

Proposition 9. *In Alg. 1, $\forall k \in \mathbb{N}$ if $post^*(\gamma(\mathcal{R}_k)) \subseteq S$ then $post(\gamma(\mathcal{R}_k)) \subseteq Z_k$.*

Theorem 1. *Assume a positive instance of the fixpoint checking problem, if CEGAR terminates so does Alg. 1.*

Proof. Let k be the size of the longest w_i for $i \in I$. Lem. 3 shows that $\gamma(\mathcal{R}_{k+1})$ is an underapproximation of the states that cannot escape S in less than k steps. Formally, we have $\gamma(\mathcal{R}_{k+1}) \subseteq \bigcap_{j=0}^{k} \widetilde{pre}[T^j](S)$. This implies that

$$\gamma(\mathcal{R}_{k+1}) \subseteq \bigcap_{i\in I} \widetilde{pre}[w_i](S) \tag{2}$$

Our next step will be to show that $post[T^*](\gamma(\mathcal{R}_{k+1})) \subseteq S$ which intuitively says that $\gamma(\mathcal{R}_{k+1})$ cannot escape S. First, note that if $\gamma(\mathcal{R}_{k+1})$ can escape from S then it cannot be with the counterexamples produced by CEGAR since $\gamma(\mathcal{R}_{k+1}) \subseteq \bigcap_{i\in I} \widetilde{pre}[w_i](S)$ which is equivalent to $\bigcup_{i\in I} post[w_i](\gamma(\mathcal{R}_{k+1})) \subseteq S$ by $\frac{\widetilde{pre}}{post}$. Let A be defined as in Lem. 4. Our proof falls into two parts:

1. $\gamma(\mathcal{R}_{k+1}) \cap A$ cannot escape from S, i.e. $post[T^*](\gamma(\mathcal{R}_{k+1}) \cap A) \subseteq S$, as shown as follows. From (2), we know that $\gamma(\mathcal{R}_{k+1}) \subseteq \bigcap_{i\in I} \widetilde{pre}[w_i](S)$, and by definition of V, we have that $\gamma(\mathcal{R}_{k+1}) \cap A \subseteq V$. As V is inductive for $post$ and $V \subseteq S$, we conclude that $post[T^*](\gamma(\mathcal{R}_{k+1}) \cap A) \subseteq S$.
2. $\gamma(\mathcal{R}_{k+1}) \cap \neg A$ cannot escape from S. For that, we show that $\gamma(\mathcal{R}_{k+1}) \cap \neg A = \emptyset$. Prop. 1 and definition of A show that $I \subseteq \gamma(\mathcal{R}_{k+1}) \cap A$ and so $\gamma(\mathcal{R}_{k+1}) \cap A \neq \emptyset$. We also know that in any state $s \in \gamma(\mathcal{R}_{k+1}) \cap A$ for $post[T^*](\{s\}) \cap \neg A \neq \emptyset$ to hold s has to be such that $s \notin \bigcap_{i\in I} \widetilde{pre}[w_i](S)$. However since $\gamma(\mathcal{R}_{k+1}) \subseteq \bigcap_{i\in I} \widetilde{pre}[w_i](S)$ and since $\mathcal{R}_{k+1}$ is given by $lfp^{\subseteq}\lambda X.\, \alpha_{k+1}(I \cup post(\gamma(X)) \cap Z_{k+1})$ over A_{k+1} (with $\gamma(\mathcal{A}_{k+1}) \subseteq \gamma(\mathcal{A}_0)$) we find that $\gamma(\mathcal{R}_{k+1}) \cap \neg A = \emptyset$. It follows that $post[T^*](\gamma(\mathcal{R}_{k+1})) \subseteq S$.

We conclude from Prop. 9 that $post(\gamma(\mathcal{R}_{k+1})) \subseteq Z_{k+1}$, hence that the test of line 4 succeeds by α_{k+1} monotonicity and $I \subseteq \gamma(\mathcal{R}_{k+1})$, and finally that Alg. 1 terminates. □

If we consider the converse result, namely that CEGAR terminates if Alg. 1 terminates we find that this does not hold for the enhanced algorithm as shown in Ex. 2.

4.2 Predicate Abstraction versus Moore Closed Abstract Domains

Below we prove that Alg. 1 does not take any advantage maintaining a Boolean closed abstract domain instead of a Moore closed one: Moore closure is as strong as Boolean closure.

The following Lemma shows that every "interesting" value added by the Boolean closure is added by the Moore closure as well. By extension we obtain that (see Th. 2) if Alg. 1 extended with the Boolean closure terminates then Alg. 1 terminates. Our result hold basically because both $\mathcal{R}_i$ and $\mathcal{S}_i$ are such that $\gamma(\mathcal{R}_i) \subseteq Z_i$ and $\gamma(\mathcal{S}_i) \subseteq Z_i$ by Lem. 2.

Lemma 5. *Let A be a finite subset of L such that $\mathcal{B}(A) = A$ and let $Z_0, Z_1, \ldots, Z_k$ be elements of L such that $Z_k \subseteq \cdots \subseteq Z_1 \subseteq Z_0$. Given $e \in \mathcal{B}(A \cup \{Z_0, Z_1, \ldots, Z_k\})$ such that $e \subseteq Z_k$ we have $e \in \mathcal{M}(A \cup \{Z_0, Z_1, \ldots, Z_k\})$.*

Theorem 2. *Provided $\mathcal{B}(\gamma(\mathcal{A}_0)) = \gamma(\mathcal{A}_0)$, if Alg. 1 with the Moore closure (viz. $\mathcal{M}$) replaced by the Boolean closure (viz $\mathcal{B}$) terminates then Alg. 1 terminates as well.*

In the context of predicate abstraction, there is no polynomial algorithm to compute the best approximation. In fact the result of applying α to value V is given by the strongest Boolean combination of predicates approximating V. Moreover the computation of the best approximation is required at each iterate of each fixpoint computation. So in the worst case the time to compute a fixpoint is given by the height of the abstract lattice times an exponential in the number of predicates. It is generally admitted that this cost is not affordable and this is why approximations in time linear in the number of predicates are preferred instead. For our algorithm the situation is pretty much better: as shown in Lem. 5 we can compute the best approximation in time linear in the number of predicates. However we need the initial set of predicates to be Boolean closed.

5 Conclusion and Future Works

We have presented a new abstract fixpoint refinement algorithm for the fixpoint checking problem. Our systematic refinement uses the information computed so far which is given by two fixpoints computed in the abstract domain. As a future work, we can consider two variants of this algorithm. First, the dual algorithm for the inverted transition system T^{-1} can be used to discover necessary correct termination conditions. A second dual algorithm where we use the inverted inclusion order $\supseteq$ on states leading to underapproximation of fixpoints. In this settings the *lfp* allows to conclude on negative instances and the *gfp* on positive instances. Also the refinement step uses the $post$ predicate transformer instead of $\widetilde{pre}$. Finally we will consider more complicated properties like properties defined by nested fixpoint expressions.

References

1. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL '77: Proc. 4th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages, ACM Press (1977) 238–252
2. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5) (2003) 752–794
3. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: POPL '02: Proc. 29th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, ACM Press (2002) 58–70
4. Cousot, P.: Méthodes Itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French). Thèse d'État ès sciences mathématiques, Université scientifique et médicale de Grenoble (1978)
5. Massé, D.: Combining forward and backward analyses of temporal properties. In: PADO'01: Programs as Data Objects, 2nd Symp. Volume 2053., Springer-Verlag (2001) 103–116
6. Giacobazzi, R., Quintarelli, E.: Incompleteness, counterexamples and refinements in abstract model-checking. In Cousot, P., ed.: SAS '01: Proc. 8th Int. Static Analysis Symp. Volume 2126 of LNCS., Springer-Verlag (2001) 356–373
7. Ball, T., Podelski, A., Rajamani, S.K.: Relative completeness of abstraction refinement for software model checking. In: TACAS '02: Tools and Algorithms for the Construction and Analysis of Systems, 8th Int. onf. Volume 2280 of LNCS., Springer-Verlag (2002) 158–172
8. Lakhnech, Y., Bensalem, S., Berezin, S., Owre, S.: Incremental verification by abstraction. In: TACAS '01: Proc. 7th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Volume 2031 of LNCS., Springer-Verlag (2001) 98–112
9. Cousot, P., Cousot, R.: Refining model checking by abstract interpretation. Automated Software Engineering **6**(1) (1999) 69–95
10. Cousot, P.: Semantic foundations of program analysis. In Muchnick, S., Jones, N., eds.: Program Flow Analysis: Theory and Applications. Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1981) 303–342
11. Esparza, J., Ganty, P., Schwoon, S.: Locality-based abstractions. In: SAS '05: Proc. 12th Int. Symp. on Static Analysis. Volume 3672 of LNCS., Springer-Verlag (2005) 118–134
12. Ganty, P., Raskin, J.F., Van Begin, L.: A complete abstract interpretation framework for coverability properties of WSTS. In: VMCAI '06: Proc. Verification, Model Checking and Abstract Interpretation. Volume 3855 of LNCS., Springer-Verlag (2006) 49–64
13. Ganty, P., Raskin, J.F., Van Begin, L.: From many places to few: Automatic abstraction refinement for Petri nets. In: Application and Theory of Petri Nets (ATPN'07). LNCS, Springer-Verlag (2007)
14. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.K.: General decidability theorems for infinite-state systems. In: Proc. 11th Annual IEEE Symp. on Logic in Computer Science (LICS), IEEE Computer Society (1996) 313–321
15. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theoretical Computer Science **256**(1-2) (1997)
16. Boigelot, B.: On iterating linear transformations over recognizable sets of integers. Theoretical Computer Science **309**(2) (2003) 413–468
17. Dams, D.: Comparing abstraction refinement algorithms. Electr. Notes Theor. Comput. Sci **89**(3) (2003)
18. Henzinger, T.A., Ho, P.H., Toi, H.W.: Hytech: A model checker for hybrid systems. Int. Journal on Software Tools for Technology Transfer **1**(1-2) (1997) 110–122
19. Bardin, S., Finkel, A., Leroux, J., Petrucci, L.: FAST: Fast acceleration of symbolic transition systems. In: CAV '03: Proc. Int. Conf. on Computer Aided Verification. Volume 2725 of LNCS., Springer-Verlag (2003) 118–121
20. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: POPL '78: Proc. 5th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, ACM Press (1978) 84–97

21. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. MIT Press and McGraw-Hill (1990)
22. Miné, A.: The octagon abstract domain. Higher-Order and Symbolic Computation **19** (2006) 31–100

A Missing proofs

Proof (of Lem. 2). First, we prove that

$$\gamma(\mathcal{S}_{i+1}) \subseteq \underbrace{\gamma(\mathcal{S}_i) \cap \widetilde{pre}(\gamma(\mathcal{S}_i))}_{Z_{i+1}} \subseteq \gamma(\mathcal{S}_i) \ .$$

First, consider the case $\gamma(\mathcal{S}_{i+1}) \subseteq Z_{i+1}$.

$$\begin{aligned}
&\mathcal{S}_{i+1} = \mathit{gfp}^{\sqsubseteq} \lambda X.\, \alpha_{i+1}\Big(\gamma(\mathcal{R}_{i+1}) \cap \widetilde{pre}(\gamma(X))\Big) && \text{def. of } \mathcal{S}_{i+1}\\
&\Rightarrow \mathcal{S}_{i+1} \sqsubseteq \alpha_{i+1} \circ \gamma(\mathcal{R}_{i+1}) && \text{prop. of } \mathit{gfp}\\
&\Rightarrow \mathcal{S}_{i+1} \sqsubseteq \mathcal{R}_{i+1} && \tfrac{\gamma}{\longleftarrow\!\!\!\!\!\!\longrightarrow_{\alpha_{i+1}}}\\
&\Rightarrow \mathcal{S}_{i+1} \sqsubseteq \alpha_{i+1}(Z_{i+1}) && \text{def. of } \mathcal{R}_{i+1}\\
&\Rightarrow \gamma(\mathcal{S}_{i+1}) \subseteq \gamma \circ \alpha_{i+1}(Z_{i+1}) && \gamma \text{ monotonicity}\\
&\Rightarrow \gamma(\mathcal{S}_{i+1}) \subseteq Z_{i+1} && Z_{i+1} \in \gamma(\mathcal{A}_{i+1}) \text{ line 10}
\end{aligned}$$

Second, from line 9, $Z_{i+1} = \gamma(\mathcal{S}_i) \cap \widetilde{pre}(\gamma(\mathcal{S}_i)) \subseteq \gamma(\mathcal{S}_i)$

Finally the result is obtained by the above reasoning, the definition of $\mathcal{R}_i$, $\mathcal{S}_i$ and Z_i, the fact that $Z_i \in \gamma(\mathcal{A}_i)$ for any i and the inclusion $Z_0 \subseteq S$ which holds by definition of Z_0. □

Proof (of Prop. 4). (1) By Lem. 2 and line 9, $Z_{i+1} = Z_i$ implies $Z_{i+1} = \gamma(\mathcal{S}_i) \cap \widetilde{pre}(\gamma(\mathcal{S}_i)) \subseteq \gamma(\mathcal{S}_i) \subseteq Z_i = Z_{i+1}$ so $\gamma(\mathcal{S}_i) \cap \widetilde{pre}(\gamma(\mathcal{S}_i)) = \gamma(\mathcal{S}_i) = Z_i$ proving $Z_i \subseteq \widetilde{pre}(Z_i)$ whence $post(Z_i) \subseteq Z_i$ by definition of Galois connection.
(2) The hypothesis and the monotonicity of α_i show that the test of line 4 fails and the algorithm computes $\mathcal{S}_i$ which is such that $\gamma(\mathcal{S}_i) \subseteq Z_i$ by Lem. 2. Then the hypothesis again shows that $I \not\subseteq \gamma(\mathcal{S}_i)$ which is equivalent to $\alpha_i(I) \not\sqsubseteq \mathcal{S}_i$ by the Galois connection $\tfrac{\gamma}{\longleftarrow\!\!\!\!\!\!\longrightarrow_{\alpha_i}}$ and thus the test of line 8 fails and the algorithm terminates at iteration i returning "KO".
(3) Lem. 2 shows that $\gamma(\mathcal{R}_i) \subseteq Z_i$, so since $post(Z_i) \subseteq Z_i$ we obtain that $post(\gamma(\mathcal{R}_i)) \subseteq Z_i$ by monotonicity of $post$. Finally monotonicity of α_i shows that $\alpha_i(I \cup post(\gamma(\mathcal{R}_i))) \sqsubseteq \alpha_i(Z_i)$ and thus the test of line 4 succeeds and the algorithm terminates. □

Proof (of Cor. 1). The proof falls naturally into two parts. If $I \subseteq Z_i$ then it is a logical consequence of Prop. 4.1 and 4.3; Otherwise termination follows from Prop. 4.2.

Lemma 6. *If $\mathit{gfp}^{\subseteq} \lambda X.\, Z_i \cap \widetilde{pre}(X)$ is computable in k steps, so is $\mathit{gfp}^{\subseteq} \lambda X.\, \gamma(\mathcal{R}_i) \cap \widetilde{pre}(X)$. Moreover the following equality holds:*

$$\gamma(\mathcal{R}_i) \cap \mathit{gfp}^{\subseteq} \lambda X.\, Z_i \cap \widetilde{pre}(X) = \mathit{gfp}^{\subseteq} \lambda X.\, \gamma(\mathcal{R}_i) \cap \widetilde{pre}(X) \ .$$

Proof. Let s be a state such that $s \in Z_i$ but not in the set of states stuck in Z_i (recall that this set is given by $\mathit{gfp}^{\subseteq} \lambda X.\, Z_i \cap \widetilde{pre}(X)$). We find that there exists a state $s' \notin Z_i$

and a value $k' \leqslant k$ such that $s \rightarrow^{k'} s'$ for otherwise the set of states stuck in Z_i is not computable in k steps.

Now, let s_1 be such that $s_1 \in \gamma(\mathcal{R}_i)$ but not in the set of states stuck in $\gamma(\mathcal{R}_i)$. Lem. 2 shows that $\gamma(\mathcal{R}_i) \subseteq Z_i$ and hence that $s_1 \in Z_i$. We conclude from $post(\gamma(\mathcal{R}_i)) \cap Z_i \subseteq \gamma(\mathcal{R}_i)$ that s_1 escape from Z_i and hence that, according to the above reasoning, there exists $s_1' \notin Z_i$ and $k' \leqslant k$ such that $s_1 \rightarrow^{k'} s_1'$, and finally that $\mathit{gfp}^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_i) \cap \widetilde{pre}(X)$ is computable in k steps.

The proof of the equality follows from the following observation: the states of $\gamma(\mathcal{R}_i)$ removed during the computation of $\mathit{gfp}^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_i) \cap \widetilde{pre}(X)$ are also removed by the computation of $\mathit{gfp}^{\subseteq}\lambda X.\, Z_i \cap \widetilde{pre}(X)$. □

Proof (of Prop. 7). We conclude from the stabilization of $\mathit{gfp}^{\subseteq}\lambda X.\, Z_i \cap \widetilde{pre}(X)$ at step k (i.e. iterate k equals iterate $k+1$ and $k \in \mathbb{N}$) that $\mathit{gfp}^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_i) \cap \widetilde{pre}(X)$ stabilizes at step k by Lem. 6. Then,

$$\begin{aligned} Z_{i+1} &= \gamma(\mathcal{S}_i) \cap \widetilde{pre}(\gamma(\mathcal{S}_i)) && \text{def. of } Z_{i+1} \\ &\subseteq \gamma(\mathcal{R}_i) \cap \widetilde{pre}(\gamma(\mathcal{R}_i)) && \gamma(\mathcal{S}_i) \subseteq \gamma(\mathcal{R}_i) \text{ by Lem. 2} \end{aligned} \tag{3}$$

$$\begin{aligned} &\mathit{gfp}^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_i) \cap \widetilde{pre}(X) \subseteq \gamma(\mathcal{S}_i) && \text{def. of } \mathcal{S}_i\text{, Lem. 1} \\ &\Rightarrow \mathit{gfp}^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_i) \cap \widetilde{pre}(X) \subseteq \gamma(\mathcal{S}_i) \cap \widetilde{pre}(\gamma(\mathcal{S}_i)) && \text{prop. of } \mathit{gfp} \\ &\Rightarrow \mathit{gfp}^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_i) \cap \widetilde{pre}(X) \subseteq Z_{i+1} && \text{def. of } Z_{i+1} \\ &\Rightarrow \mathit{gfp}^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_i) \cap \widetilde{pre}(X) = \mathit{gfp}^{\subseteq}\lambda X.\, Z_{i+1} \cap \widetilde{pre}(X) && \text{by (3)} \end{aligned} \tag{4}$$

We have shown above that $\mathit{gfp}^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_i) \cap \widetilde{pre}(X)$ stabilizes at step k. By (3) and (4) we find that $\mathit{gfp}^{\subseteq}\lambda X.\, Z_{i+1} \cap \widetilde{pre}(X)$ stabilizes at step $k-1$.

Repeated application of the above reasoning shows that $\mathit{gfp}^{\subseteq}\lambda X.\, Z_{i+k} \cap \widetilde{pre}(X)$ stabilizes at step 0. We thus obtain that

$$\begin{aligned} &\mathit{gfp}^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_{i+k}) \cap \widetilde{pre}(X) \\ &= \gamma(\mathcal{R}_{i+k}) \cap \mathit{gfp}^{\subseteq}\lambda X.\, Z_{i+k} \cap \widetilde{pre}(X) && \text{Lem. 6} \\ &= \gamma(\mathcal{R}_{i+k}) \cap Z_{i+k} && \text{stabilizes at step 0} \\ &= \gamma(\mathcal{R}_{i+k}) && \gamma(\mathcal{R}_{i+k}) \subseteq Z_{i+k} \text{ by Lem. 2} \end{aligned}$$

This property allows us to conclude that $\gamma(\mathcal{R}_{i+k}) = \gamma(\mathcal{S}_{i+k})$, hence that $Z_{i+k+1} = \gamma(\mathcal{R}_{i+k})$ and finally that $\gamma(\mathcal{A}_{i+k+1}) = \gamma(\mathcal{A}_{i+k})$. So it is routine to check that $Z_{i+k+1} = Z_{i+k}$ and so the algorithm terminates by Cor. 1. □

Proposition 10. *Let R be such that $T \subseteq R \subseteq T^*$. If $\mathit{gfp}^{\subseteq}\lambda X.\, Z_i \cap \widetilde{pre}[R](X)$ is computable in k steps, so is $\mathit{gfp}^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_i) \cap \widetilde{pre}[R](X)$. Moreover the following equality holds:*

$$\gamma(\mathcal{R}_i) \cap \mathit{gfp}^{\subseteq}\lambda X.\, Z_i \cap \widetilde{pre}[R](X) = \mathit{gfp}^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_i) \cap \widetilde{pre}[R](X)\ .$$

Proof. The proof of Prop. 7 can be straightforwardly generalized to any binary relation R such that $T \subseteq R \subseteq T^*$. □

Proof (of Prop. 8). As $Z_0 = S$, by hypothesis we have $gfp^{\subseteq}\lambda X.\, Z_0 \cap \widetilde{pre}[R_2](X)$ stabilizes after at most k steps (i.e. iterate k equals iterate $k+1$ and $k \in \mathbb{N}$), hence we deduce that $gfp^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_0) \cap \widetilde{pre}[R_2](X)$ stabilizes at most after k steps by Prop. 10. Then,

$$\begin{aligned} Z_1 &= \gamma(\mathcal{S}_0) \cap \widetilde{pre}[R_1](\gamma(\mathcal{S}_0)) && \text{def. of } Z_{i+1} \\ &\subseteq \gamma(\mathcal{R}_0) \cap \widetilde{pre}[R_1](\gamma(\mathcal{R}_0)) && \gamma(\mathcal{S}_0) \subseteq \gamma(\mathcal{R}_0) \text{ by Lem. 2} \\ &\subseteq \gamma(\mathcal{R}_0) \cap \widetilde{pre}[R_2](\gamma(\mathcal{R}_0)) && \widetilde{pre}[R_1] \dot{\subseteq} \widetilde{pre}[R_2] \end{aligned} \tag{5}$$

$$\begin{aligned} & gfp^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_0) \cap \widetilde{pre}(X) \subseteq \gamma(\mathcal{S}_0) && \text{def. of } \mathcal{S}_0, \text{ Lem. 1} \\ \Rightarrow\ & gfp^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_0) \cap \widetilde{pre}(X) \subseteq \gamma(\mathcal{S}_0) \cap \widetilde{pre}[R_1](\gamma(\mathcal{S}_0)) && \text{def. (1), prop. of } gfp \\ \Rightarrow\ & gfp^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_0) \cap \widetilde{pre}(X) \subseteq Z_1 && \text{def. of } Z_1 \\ \Rightarrow\ & gfp^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_0) \cap \widetilde{pre}(X) = gfp^{\subseteq}\lambda X.\, Z_1 \cap \widetilde{pre}(X) && \text{by (5)} \end{aligned} \tag{6}$$

We have shown above that $gfp^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_0) \cap \widetilde{pre}[R_2](X)$ stabilizes at step k, by (5) and (6) we find that $gfp^{\subseteq}\lambda X.\, Z_1 \cap \widetilde{pre}[R_2](X)$ stabilizes at step $k-1$.

Repeated application of the above reasoning shows that $gfp^{\subseteq}\lambda X.\, Z_k \cap \widetilde{pre}[R_2](X)$ stabilizes at step 0 and so does $gfp^{\subseteq}\lambda X.\, Z_k \cap \widetilde{pre}(X)$. We thus obtain that

$$\begin{aligned} & gfp^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_k) \cap \widetilde{pre}(X) \\ &= \gamma(\mathcal{R}_k) \cap gfp^{\subseteq}\lambda X.\, Z_k \cap \widetilde{pre}(X) && \text{Lem. 6} \\ &= \gamma(\mathcal{R}_k) \cap Z_k && \text{stabilizes at step 0} \\ &= \gamma(\mathcal{R}_k) && \gamma(\mathcal{R}_k) \subseteq Z_k \text{ by Lem. 2} \end{aligned}$$

This property allows us to conclude that $\gamma(\mathcal{R}_k) = \gamma(\mathcal{S}_k)$, hence that $Z_{k+1} = \gamma(\mathcal{R}_k)$ and finally that $\gamma(\mathcal{A}_{k+1}) = \gamma(\mathcal{A}_k)$. So it is routine to check that $Z_{k+1} = Z_k$ and so the algorithm terminates by Cor. 1. □

Proof (of Lem. 4). Let $\mathcal{T}^\alpha = (C^\alpha, T^\alpha, I^\alpha)$ be the abstract transition system where C^α is the partition that is obtained when the spurious counterexamples from $\{w_i\}_{i \in I}$ has been considered.

Let $F^\alpha \subseteq C^\alpha$ be subset of reachable classes in $\mathcal{T}^\alpha$. Let F be $\bigcup_{C_i \in F^\alpha} C_i$, i.e. F contains the set of states that are within reachable classes in $\mathcal{T}^\alpha$. As the abstract analysis is conclusive, we know that $I \subseteq F$, $F \subseteq S$, and $post(F) \subseteq F$. As F is inductive for $\lambda X.\, I \cup post(X)$, we know that $F \cap \bigcup_{i \in I} pre[w_i](\neg S)$ is emtpy, i.e. $F \subseteq \bigcap_{i \in I} \widetilde{pre}[w_i](S)$. The classes in C^α are either classes that were present in the initial partition (defined by A_0) or classes that were refined and does not contain bad states, so F is composed of classes of the initial partition and refined classes of the initial partition. None of these classes intersect $\bigcup_{i \in I} pre[w_i](\neg S)$. □

Proof (of Prop. 9). Our proof is by induction on k.
Base case. The result follows immediately since in Alg. 1 we have $Z_0 = S$.
Inductive case. We show the contrapositive of the implication. We first relate Z_{k+1} with the set of states that cannot escape from $\gamma(\mathcal{R}_k)$ (i.e. $gfp^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_k) \cap \widetilde{pre}(X)$) as follows

$$\begin{aligned}
& gfp^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_k) \cap \widetilde{pre}(X) \subseteq \gamma(\mathcal{S}_k) && \text{Lem. 1}\\
\Rightarrow\ & gfp^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_k) \cap \widetilde{pre}(X) \subseteq \gamma(\mathcal{S}_k) \cap \widetilde{pre}(\gamma(\mathcal{S}_k)) && \text{fixpoint property}\\
\Leftrightarrow\ & gfp^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_k) \cap \widetilde{pre}(X) \subseteq Z_{k+1} && \text{def. of } Z_{k+1}
\end{aligned}$$

We conclude from the contrapositive hypothesis given by $post(\gamma(\mathcal{R}_{k+1})) \not\subseteq Z_{k+1}$ and by above that $post(\gamma(\mathcal{R}_{k+1})) \not\subseteq gfp^{\subseteq}\lambda X.\, \gamma(\mathcal{R}_k) \cap \widetilde{pre}(X)$. Intuitively this means that some states of $post(\gamma(\mathcal{R}_{k+1}))$ can escape $\gamma(\mathcal{R}_k)$ or more formally that $post^*(\gamma(\mathcal{R}_{k+1})) \not\subseteq \gamma(\mathcal{R}_k)$. So consider the sequence $s_0, s_1, \ldots, s_n$ such that $(s_i, s_{i+1}) \in T$ for $1 \leqslant i < n$ and $s_0 \in \gamma(\mathcal{R}_{k+1})$ and $s_n \notin \gamma(\mathcal{R}_k)$. Since, by Lem. 2, $\gamma(\mathcal{R}_{k+1}) \subseteq \gamma(\mathcal{R}_k)$ the sequence can be partitioned into a prefix (from 0 to i) where the states belong to $\gamma(\mathcal{R}_k)$ and a suffix (from $i+1$ to n) where the states does not belong to $\gamma(\mathcal{R}_k)$. We have that $\{s_{i+1}\} \not\subseteq Z_k$ for otherwise $post(\gamma(\mathcal{R}_k)) \cap Z_k \subseteq \gamma(\mathcal{R}_k)$ does not hold. Lem. 2 shows that $\gamma(\mathcal{R}_{k+1}) \subseteq \gamma(\mathcal{R}_k) \subseteq Z_k$. We conclude from $s_i \in \gamma(\mathcal{R}_k)$, $s_{i+1} \notin Z_k$ and $s_i \rightarrow s_{i+1}$ that $post(s_i) \not\subseteq Z_k$, hence that $post(\gamma(\mathcal{R}_k)) \not\subseteq Z_k$ and finally that $post^*(\gamma(\mathcal{R}_k)) \not\subseteq S$ using the induction hypothesis. Finally since, by definition of the sequence, s_{i+1} is reachable from $\gamma(\mathcal{R}_{k+1})$ we find that $post^*(\gamma(\mathcal{R}_{k+1})) \not\subseteq S$. □

Proof (of Lem. 5). We first notice that the value can be expressed in a form similar to the Conjunctive Normal From (CNF) used in propositional logic. Moreover since $e \subseteq Z_k$ we have that $e \cap Z_0 \cap Z_1 \cap \cdots \cap Z_k = e$. So e can be expressed as follows:

$$e = \bigcap_{i \in I} (a_1 \cup \cdots \cup a_{n_i}) \cap \bigcap_{j=0}^{k} Z_j$$

such that the a_i's belong to A and I is a finite set since A is finite subset of L.

We now give two syntactic transformations of the above e that preserves its semantics.

- Remove from e each union of the form $(Z_j \cup \psi)$. This rule does not modify the value of e since $e \subseteq Z_j \subseteq (Z_j \cup \psi)$.
- Replace in e any union of the form $\neg Z_j \cup \psi$ by ψ. This rule does not modify the value of e as shown below.

$$\begin{aligned}
& Z_j \cap (\neg Z_j \cup \psi) && \text{subexpression of } e\\
&= (Z_j \cap \neg Z_j) \cup (Z_j \cap \psi) && \text{set theory}\\
&= \emptyset \cup (Z_j \cap \psi) && \text{set theory}\\
&= Z_j \cap \psi
\end{aligned}$$

Since e has finitely many unions expressions the two rules can be applied finitely many times because the size of e decrease after applying any rule. It follows that the repeated application of these two rules stabilizes after a finite number of steps.

Moreover after stabilization no value $Z_0, \ldots, Z_i$ appears in a union of 2 or more values which means since $\mathcal{B}(A) = A$ that $e \in \mathcal{M}(A)$. □

In Ex. 2 let X^δ, δ be the sequences of iterates for $\lambda X.\, S \cap \widetilde{pre}[t_2](X)$ which converges to $gfp^{\subseteq} \lambda X.\, S \cap \widetilde{pre}[t_2](X)$ We have:

$$
\begin{array}{lll}
X^0 = \top & & \\
X^1 = S \cap \widetilde{pre}[t_2](X_0) & & \text{def. of the iterates} \\
\quad = S & & \top \subseteq \widetilde{pre}[t_2](\top) \\
\quad = \top \setminus \{(x,y) \mid y = x+1\} & & \\
X^2 = S \cap \widetilde{pre}[t_2](X_1) & & \text{def. of the iterates} \\
\quad = S \cap \widetilde{pre}[t_2](S) & & X_1 = S \\
\quad = \top \setminus \{(x,y) \mid y = x+1 \text{ or } y = x+2\} & & \text{from above} \\
\quad \vdots & & \\
X^\delta = \{(x,y) \mid y \leqslant x\} & &
\end{array}
$$

B Examples

In this section we will show that Alg. 1 terminates on two well-known array sorting algorithms. The property we prove are safety properties which states that the array to be sorted is never accessed out of its bound. We do not analyze directly the program code of those algorithms but an abstraction instead. Our abstraction forgets about the content of the array and so we replace the tests based on array's values by non deterministic choices. Our model is sound in the sense that it contains at least all the behaviors of the program. So if the abstract model satisfies the safety property so does the program. The abstract model we use is given by counter automata where each counter corresponds to an array index. The safety property is naturally reduced to a reachability property on the counter automaton. Prop. 5 shows that when submitted a negative instance Alg. 1 terminates. Consequently the instances considered below are positive instances.

At the present time, no implementation of Alg. 1 is available but, as shown in the previous sections, the algorithm is correct and moreover we identified some conditions that, if satisfied, guarantee its termination. We thus rely on these conditions to show that our algorithm is going to conclude with the right answer. These conditions are non trivial but they can be evaluated using available tools. We choose to rely on the Hytech model checker (see [18]) to prove that the condition of Prop. 7 is satisfied and hence that Alg. 1 terminates.

Besides Hytech we also rely on the FAST tool (see [19]). FAST is a tool that uses acceleration techniques. If the FAST tool terminates when evaluating $gfp^{\subseteq} \lambda X.\, S \cap \widetilde{pre}[T](X)$ it returns an acceleration scheme R such that $T \subseteq R \subseteq T^*$. Then Prop. 8 is used to show that for any R' such that $R \subseteq R' \subseteq T^*$ Alg. 1 terminates provided line 9 is replaced by $Z_{i+1} = \gamma(\mathcal{S}_i) \cap \widetilde{pre}[R'](\gamma(\mathcal{S}_i))$.

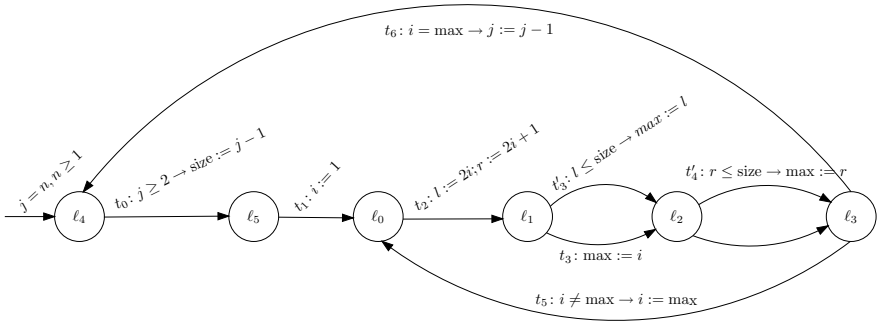


Fig. 3. Counters automata modeling of the Heapsort algorithm

The Heapsort Algorithm. Heapsort is a classical example in static analysis (e.g. [20] using the polyhedral abstraction). We shall prove that the array to be sorted is never accessed out of its bound given by 1 and n. The counter automaton is given Fig. 3. The model has been derived manually from the code given in [21]. Since the array is accessed through $V = \{\ell, i, r, \max\}$ we want to prove that each access is legal. Formally the set S of states representing legal access is given by the following formulas ψ_1 to ψ_4 associated to the locations with the same index.

$$\begin{aligned}
\psi_1 &= \ell \leqslant j - 1 \to (1 \leqslant \ell \leqslant n \,\&\, 1 \leqslant i \leqslant n)\\
\psi_2 &= r \leqslant j - 1 \to (1 \leqslant r \leqslant n \,\&\, 1 \leqslant \max \leqslant n)\\
\psi_3 &= i \neq \max \to (1 \leqslant i \leqslant n \,\&\, 1 \leqslant \max \leqslant n)\\
\psi_4 &= 1 \leqslant j \leqslant n
\end{aligned}$$

The set I of initial states is given by $\begin{cases} j = n \,\&\, n \geqslant 1 & \text{at } \ell_4 \\ \bot & \text{elsewhere.} \end{cases}$

Let $\mathcal{P}_0$ be the set of predicates appearing in the text of the program. Formally, $\mathcal{P}_0$ is given by $\{j \geqslant 2, j = n, n \geqslant 1, i = \max, i \neq \max, r \leqslant j - 1, \ell \leqslant j - 1, \psi_1, \psi_2, \psi_3, \psi_4\}$. The initial abstract domain is given by $A_0 = \mathcal{M}(\mathcal{P}_0)$.

We are going to show that Alg. 1 terminates on the Heapsort algorithm. Hytech terminates for

$$\begin{aligned}
\psi'_1 &= \ell \leqslant j - 1 \to (1 \leqslant \ell \,\&\, 1 \leqslant i \leqslant n)\\
\psi'_2 &= r \leqslant j - 1 \to (1 \leqslant r \,\&\, 1 \leqslant \max \leqslant n)\\
\psi_3 &= i \neq \max \to (1 \leqslant i \leqslant n \,\&\, 1 \leqslant \max \leqslant n)\\
\psi_4 &= 1 \leqslant j \leqslant n
\end{aligned}$$

and thus Alg. 1 terminates by Prop. 7.

Notice that in ψ'_1 and ψ'_2 we do not check for $\ell \leqslant n$ (recall that component ℓ of the array is accessed). However since j is not modified in locations $\ell_4, \ell_0, \ell_1, \ell_2$ and by ψ_4 we can deduce that whenever the array is accessed through ℓ, the inequality $\ell \leqslant n$ holds.

Now assume you do not want this ad hoc reasoning to convince yourself that the array is never accessed out of its bounds. We can still manage this situation since FAST terminates for

$$\psi_1 = \ell \leqslant j - 1 \rightarrow (1 \leqslant \ell \leqslant n \,\&\, 1 \leqslant i \leqslant n)$$
$$\psi_2 = r \leqslant j - 1 \rightarrow (1 \leqslant r \leqslant n \,\&\, 1 \leqslant \max \leqslant n)$$
$$\psi_3 = i \neq \max \rightarrow (1 \leqslant i \leqslant n \,\&\, 1 \leqslant \max \leqslant n)$$
$$\psi_4 = 1 \leqslant j \leqslant n$$

and thus Alg. 1 terminates by Prop. 8.

The Bubble sort algorithm. The necessary termination condition $n \geqslant 0$ is found in [4] by an iterated forward-backward non-relational interval analysis. ASTRÉE proves the absence of out of array bound error in 0.8 s thanks to the octagonal abstraction [22].

We shall prove that the array to be sorted is never accessed out of its bound given by 0 and n. Since the array is accessed through variable j only we want to prove that $0 \leqslant j \leqslant n$ holds for each reachable state. The counter automaton given in Fig. 4 has been extracted from [4].

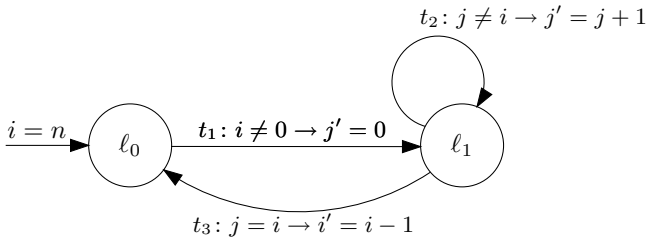


Fig. 4. Our two counters automata modeling the Bubblesort algorithm

In our model we have variables i and j and a non negative parameter n representing the array's size. Let I and S be given by $\{(i, j, n) \mid i = n\}$ and $\{(i, j, n) \mid 0 \leqslant j \leqslant n\}$ respectively. Let $\mathcal{P}_0$ be the set of predicates appearing in the text of the program plus the formula representing S. Formally, $\mathcal{P}_0$ is given by $\{i = n, i = 0, i = j, 0 \leqslant j \leqslant n\}$. The abstract domain A_0 is given by $\mathcal{M}(\mathcal{P}_0)$.

Finally we have that since the FAST tool terminates then Alg. 1 terminates by Prop. 8.