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 [2,3,4] 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.

* This research was supported the Belgian FNRS grant 2.4530.02 of the FRFC project "Centre Fédéré en Vérification" and by the project "MoVES", an Interuniversity Attraction Poles Programme of the Belgian Federal Government.

** Pierre Ganty is supported by the FNRS under a FRIA grant.

© Springer-Verlag Berlin Heidelberg 2007

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 distinguishes 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 bound to use values which are more precise than 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 [5]) 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 [6] 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 ([7] as generalized by [8]) 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 [9] but, for efficiency, restricted to states reachable within the invariant to be checked. In [10] the authors define a restricted abstract domain completion. However since we reuse all the information computed so far our completion is much more finer than theirs. In [11] 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. Sect. 5 concludes the paper and proposes some future works.

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 $(f \circ g) = \lambda X.\, f(g(X))$. Let X be any set and let $f \in X \mapsto X$ be a function on this set. 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: $\lambda x.\, f(x) \mathrel{\dot{\subseteq}} \lambda x.\, g(x)$ iff $\forall y \in L\colon f(y) \subseteq g(y)$. 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-tuple $\mathcal{T} = (C, I, T)$ 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 $\widetilde{\mathit{pre}}[T'](C') = \neg \mathit{pre}[T'](\neg C') = \neg \mathit{post}[T'^{-1}](\neg C')$ $= \{c \in C \mid \forall c'\colon (c, c') \in T' \Rightarrow c' \in C'\}$. 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 [12], 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 $s \in \mathit{post}^*(I)$. 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 [12], 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 $\langle A, \sqsubseteq, \sqcap, \sqcup, \top, \bot\rangle$, respectively. The two lattices are related by abstraction and concretization maps α and γ forming a *Galois connection* $\forall c \in L : \forall a \in A : \alpha(c) \sqsubseteq a \Leftrightarrow c \subseteq \gamma(a)$ [7]. 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. We recall here a well-known (see [13] for instance) Galois connection that will be used later on: given a transition system (C, I, T) we have $\langle C, \subseteq\rangle \underset{\lambda X.post[T](X)}{\overset{\lambda X.\widetilde{pre}[T](X)}{\leftrightarrows}} \langle C, \subseteq\rangle$. 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 A_i, we write $\gamma(A_i)$ for the subset of concrete sets $X \in L$ that can be represented by abstract values in A_i.

The set $\gamma(A_i) \subseteq L$ of concrete values that the abstract domain represents must be closed by intersection if there is a Galois connection between 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*[1].
- 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 $\gamma(\mathcal{A}_P) \subseteq L$ is the smallest set which is Boolean closed and contains the sets $\{[\![p]\!] \mid p \in P\}$, i.e. $\gamma(\mathcal{A}_P) = \mathcal{B}(\{[\![p]\!] \mid p \in P\})$. The elements of $\mathcal{A}_P$ are equivalent to propositional formulas built from the predicates in P.

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

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.

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 [14] 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. $\mathcal{S}$, to be the abstract forward, resp. backward, semantics on A_i as $\mathit{lfp}^{\sqsubseteq}\lambda X.\, \alpha_i((I \cup \mathit{post}(\gamma(X))) \cap Z)$, resp. $\mathit{gfp}^{\sqsubseteq}\lambda X.\, \alpha_i(S \cap \widetilde{\mathit{pre}}(\gamma(X)))$.*

$$\left.\begin{aligned} \mathit{lfp}^{\subseteq}\lambda X.\, (I \cup \mathit{post}(X)) \cap Z &\subseteq \gamma(\mathcal{R}) \\ \mathit{gfp}^{\subseteq}\lambda X.\, S \cap \widetilde{\mathit{pre}}(X) &\subseteq \gamma(\mathcal{S}) \end{aligned}\right\} \text{ We call this inclusion the overapproximation of the abstract semantics.}$$

The Fixpoint Checking Problem.
Instance: a transition system (C, I, T) and a set of states $S \subseteq C$.
Question: Does the inclusion $\mathit{lfp}^{\subseteq}\lambda X.\, I \cup \mathit{post}(X) \subseteq S$ holds ?

3 Abstract Fixpoint Checking Algorithm

Alg. 1 has been inspired and is a generalization of what we have done previously in [15,16,17]. We review here its main characteristics.

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 A_i can represent (this set is $\gamma(A_i)$). The new

Algorithm 1. The abstract fixpoint checking algorithm

```
Data: An instance of the fixpoint checking problem such that $I \subseteq S$ and an
      abstract domain $A_0$ such that $S \in \gamma(A_0)$
1  $Z_0 = S$
2  for $i = 0, 1, 2, 3, \ldots$ do
3      Compute $\mathcal{R}_i$ given by $\mathit{lfp}^{\sqsubseteq} \lambda X.\, \alpha_i\Big((I \cup \mathit{post}(\gamma(X))) \cap Z_i\Big)$
4      if $\alpha_i(I \cup \mathit{post}(\gamma(\mathcal{R}_i))) \sqsubseteq \alpha_i(Z_i)$ then
5          return OK
6      else
7          Compute $\mathcal{S}_i$ given by $\mathit{gfp}^{\sqsubseteq} \lambda X.\, \alpha_i\Big(\gamma(\mathcal{R}_i) \cap \widetilde{\mathit{pre}}(\gamma(X))\Big)$
8          if $\alpha_i(I) \sqsubseteq \mathcal{S}_i$ then
9              Let $Z_{i+1} = \gamma(\mathcal{S}_i) \cap \widetilde{\mathit{pre}}(\gamma(\mathcal{S}_i))$
10             Let $A_{i+1}$ be s.t. $\gamma(A_{i+1}) = \mathcal{M}\big(\{Z_{i+1}\} \cup \gamma(A_i)\big)$
11         else
12             return KO
13         end
14     end
15 end
```

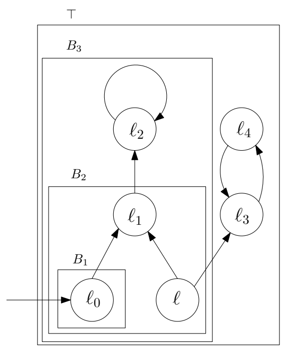
abstract domain is given by A_{i+1}. It is worth pointing that we actually add more than the single value Z_{i+1} to the abstract domain since working in the framework of abstract interpretation requires that $\gamma(A_{i+1})$ is a Moore family. We will see later that Moore closure is sufficiently powerful 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 [18,19], see [16,17] 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



$\mathcal{R}_0 = B_3$ 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 $A_1 = A_0 \cup \{B_4, B_5\}$ 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

the following positive instance of the fixpoint checking problem where $A_0 = \{B_1, B_2, B_3, \top\}$, $I = \{\ell_0\}$, 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.

3.1 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, if $post^*(I) \subseteq S$ then $post^*(I) \subseteq \gamma(\mathcal{R}_i)$ for any $i \in \mathbb{N}$.*

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. For the inductive case we prove the contrapositive. 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, $\mathit{post}^*(I) \not\subseteq S$ 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 $s \to^* s'$ and $s' \notin \gamma(\mathcal{R}_{i-1})$. First, note that as $s \in \mathit{post}^*(I)$, we conclude that $s' \in \mathit{post}^*(I)$. But as $s' \notin \gamma(\mathcal{R}_{i-1})$, we know that $\mathit{post}^*(I) \not\subseteq \gamma(\mathcal{R}_{i-1})$ and by induction hypothesis we conclude that $\mathit{post}^*(I) \not\subseteq S$.
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 $\gamma(\mathcal{S}_{i-1}) \cap \widetilde{\mathit{pre}}(\gamma(\mathcal{S}_{i-1}))$ that there exists $s' \notin \gamma(\mathcal{S}_{i-1})$ such that $s \to s'$. Either $s' \notin \gamma(\mathcal{R}_{i-1})$ or $s' \in \gamma(\mathcal{R}_{i-1})$ and by the previous case, we know that $s' \to^* s''$ and $s'' \notin \gamma(\mathcal{R}_{i-1})$. In the two cases, we conclude that $\mathit{post}^*(I) \not\subseteq \gamma(\mathcal{R}_{i-1})$ and by induction hypothesis that $\mathit{post}^*(I) \not\subseteq S$.
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 $\mathit{post}^*(I) \not\subseteq S$. □

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* $\mathit{post}^*(I) \subseteq S$.

Proof.

$$\begin{array}{lr}
\text{Algorithm says “OK”} & \\
\Leftrightarrow \alpha_i(I \cup \mathit{post}(\gamma(\mathcal{R}_i))) \sqsubseteq \alpha_i(Z_i) & \text{line } 4\\
\Leftrightarrow \alpha_i(I) \sqsubseteq \alpha_i(Z_i) \mathbin{\&} \alpha_i \circ \mathit{post} \circ \gamma(\mathcal{R}_i) \sqsubseteq \alpha_i(Z_i) & \alpha_i \text{ additivity}\\
\Leftrightarrow I \sqsubseteq \gamma \circ \alpha_i(Z_i) \mathbin{\&} \mathit{post}(\gamma(\mathcal{R}_i)) \sqsubseteq \gamma \circ \alpha_i(Z_i) & \underset{\alpha_i}{\overset{\gamma}{\leftrightarrows}}\\
\Leftrightarrow I \subseteq Z_i \mathbin{\&} \mathit{post}(\gamma(\mathcal{R}_i)) \subseteq Z_i & Z_i \in \gamma(A_i) \text{ line } 10
\end{array}$$

Then,

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

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

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 $I \nsubseteq \gamma(\mathcal{S}_i)$ by $\overset{\gamma}{\underset{\alpha_i}{\leftrightarrows}}$. We conclude from Lem. 2 that $\gamma(\mathcal{R}_{i+1}) \subseteq \gamma(\mathcal{S}_i)$, hence that $I \nsubseteq \gamma(\mathcal{R}_{i+1})$ and finally that $post^*(I) \nsubseteq S$ using the contrapositive of Prop. 1. □

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 $R \subseteq T^*$. In fact for correction to hold Z_{i+1} must be such that $\widetilde{pre}[T^*](\gamma(\mathcal{S}_i)) \subseteq Z_{i+1} \subseteq \gamma(\mathcal{S}_i)$. Later we will see how we can benefit from acceleration techniques which build a relation R such that $T \subseteq R \subseteq T^*$. This alternative refinement using R including T yields to stronger termination properties of the algorithm.

3.2 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 $I \nsubseteq Z_i$ 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,

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

□

Proposition 5. *If $post^*(I) \nsubseteq S$ 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 $s \to^k s'$. 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 $I \nsubseteq \bigcap_{j=0}^{k} \widetilde{pre}[T^j](S)$, hence that $I \nsubseteq \gamma(\mathcal{R}_{k-1})$ by transitivity and finally that $I \nsubseteq Z_k$ by Lem. 2. The last step uses Prop. 4.2 to show that the algorithm terminates. □

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

Proposition 6. *If there exists a poset $Y \subseteq L$ such that the descending chain condition holds on $\langle Y, \subseteq \rangle$ and $Z_i \in Y$ for all $i \in \mathbb{N}$ 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 existence of a poset satisfying the above hypothesis. □

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 that 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 steps, then Alg. 1 terminates.*

3.3 Termination of the Algorithm Enhanced by Acceleration Techniques

In this section we will study an enhancement of Alg. 1 which relies on acceleration techniques (we refer the interested reader to [20] and the references given there). Roughly speaking, acceleration techniques allow us to compute underapproximations of the transitive closure of some binary relation as, for instance the transition relation.

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 $\underset{post}{\overset{\widetilde{pre}}{\leftrightarrows}}$, hence that the enhanced algorithm terminates at iteration where $i = 1$ 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. Discs depicts some reachable states, which are given by $\{(x,y) \mid y \leqslant x \mathbin{\&} 0 \leqslant x\}$. 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 A_0 is such that $\gamma(A_0) = \mathcal{M}(S)$.

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

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

We now turn to the evaluation of the *gfp*.

$$\begin{aligned}
X^0 &= \top & \\
X^1 &= \alpha_0(\gamma(\mathcal{R}_0) \cap \widetilde{pre}[T](\gamma(X_0))) & \\
&= \alpha_0(S) & \gamma(\mathcal{R}_0) = S, \top \subseteq \widetilde{pre}[T](\top)\\
&= S & S \in \gamma(A_0)\\
X^2 &= \alpha_0(S \cap \widetilde{pre}[T](\gamma(X_1))) & \\
&= \alpha_0\big(S \cap \widetilde{pre}[t_1](\gamma(X_1)) \cap \widetilde{pre}[t_2](\gamma(X_1))\big) & \text{def. } \widetilde{pre}\\
&= \alpha_0\big(S \cap \widetilde{pre}[t_2](\gamma(X_1))\big) & S \cap \widetilde{pre}[t_1](S) = S
\end{aligned}$$

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 escapes 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 fails 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 be obtained using acceleration techniques as shown below.

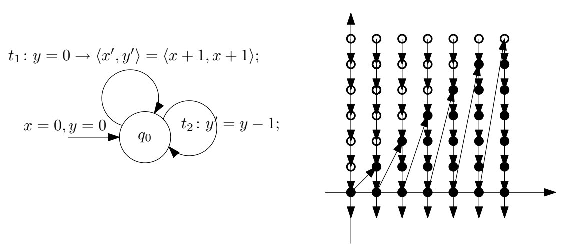

$t_1: y = 0 \to \langle x', y' \rangle = \langle x+1, x+1 \rangle;$
$x = 0, y = 0$
q_0
$t_2: y' = y - 1;$

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

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 $S \cap \widetilde{pre}[t_1 \cup t_2^*](S)$.

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

The latter fixpoint evaluates to $\{(x, y) \mid y \leqslant x\}$ and so the new abstract domain A_1 is such that $\gamma(A_1) = \mathcal{M}(\gamma(A_0) \cup Z_1)$. 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. ■

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

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

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 $\mathcal{S}_i$ have their iterated function given by $\lambda X.\, \alpha_i(I \cup post(\gamma(X)) \cap Z_i)$ and $\lambda X.\, \alpha_i(\gamma(\mathcal{R}_i) \cap \widetilde{pre}(\gamma(X)))$, respectively. For various reasons we may be constrained to use a less precise approximations f, g that is to say $\lambda X.\, \alpha_i(I \cup post(\gamma(X)) \cap Z_i) \dot{\subseteq} \lambda X.\, f(X)$ and $\lambda X.\, \alpha_i(\gamma(\mathcal{R}_i) \cap \widetilde{pre}(\gamma(X))) \dot{\subseteq} \lambda X.\, g(X)$. In this context provided the additional mild requirements $\lambda X.\, f(X) \dot{\subseteq} \lambda X.\, Z_i$ and $\lambda X.\, g(X) \dot{\subseteq} \lambda X.\, \gamma(\mathcal{R}_i)$ hold we find that all the results of Sect. 3.1, Sect. 3.2 and Sect. 3.3 remain valid.

4 Relationships with Other Approaches

4.1 Counterexample Guided Abstraction Refinement

We first recall here the main ingredients of the CEGAR approach [21, §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 $C_{i_n} \nsubseteq S$. 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 $I \nsubseteq \widetilde{pre}[t_1 \circ \ldots \circ t_{n-1}](S)$ (by monotonicity we have $I \nsubseteq \widetilde{pre}[T^*](S)$), 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}; or which are not initial if $j = 0$. 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 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 counterexample (identifying a positive instance of the fixpoint 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:*

$$\exists A \in \gamma(A_0) \colon I \subseteq \underbrace{gfp^{\subseteq} \lambda X.\, A \cap \widetilde{pre}(X)}_{V} \subseteq S \;\&\; V = A \cap \bigcap_{i \in I} \widetilde{pre}[w_i](S) \ .$$

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 $\overset{\widetilde{pre}}{\underset{post}{\leftrightarrows}}$. 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 $\gamma(A_{k+1}) \subseteq \gamma(A_0)$) 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. $\square$

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 holds 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 $\mathcal{B}(\gamma(A_0)) = \gamma(A_0)$, 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, pp. 238–252. ACM Press, New York (1977)
2. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Software verification with blast. In: SPIN '03: Model Checking Software, 10th International SPIN Workshop. LNCS, vol. 2648, pp. 235–239. Springer, Heidelberg (2003)
3. Chaki, S., Clarke, E.M., Groce, A., Jha, S., Veith, H.: Modular verification of software components in c. In: ICSE '03: Proc. 25th International Conference on Software Engineering, pp. 385–395. IEEE Computer Society Press, Los Alamitos (2003)
4. Ball, T., Rajamani, S.K.: The slam project: debugging system software via static analysis. In: POPL '02: Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pp. 1–3. ACM Press, New York (2002)

5. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003)
6. Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. In: POP '02: Proc. 29th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, pp. 58–70. ACM Press, New York (2002)
7. 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)
8. Massé, D.: Combining forward and backward analyses of temporal properties. In: PADO '01: Programs as Data Objects, 2nd Symp. LNCS, vol. 2053, pp. 103–116. Springer, Heidelberg (2001)
9. Giacobazzi, R., Quintarelli, E.: Incompleteness, counterexamples and refinements in abstract model-checking. In: Cousot, P. (ed.) SAS '01: Proc. 8th Int. Static Analysis Symp. LNCS, vol. 2126, pp. 356–373. Springer, Heidelberg (2001)
10. 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. LNCS, vol. 2280, pp. 158–172. Springer, Heidelberg (2002)
11. 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. LNCS, vol. 2031, pp. 98–112. Springer, Heidelberg (2001)
12. Cousot, P., Cousot, R.: Refining model checking by abstract interpretation. Automated Software Engineering 6(1), 69–95 (1999)
13. Cousot, P.: Partial completeness of abstract fixpoint checking, invited paper. In: SARA '2000: Proc. 4th Int. Symp. on Abstraction, Reformulations and Approximation. LNCS (LNAI), vol. 1864, pp. 1–25. Springer, Heidelberg (2000)
14. Cousot, P.: Semantic foundations of program analysis. In: Muchnick, S.S, Jones, N.D (eds.) Program Flow Analysis: Theory and Applications, pp. 303–342. Prentice-Hall, Englewood Cliffs (1981)
15. Esparza, J., Ganty, P., Schwoon, S.: Locality-based abstractions. In: SAS '05: Proc. 12th Int. Symp. on Static Analysis. LNCS, vol. 3672, pp. 118–134. Springer, Heidelberg (2005)
16. 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. LNCS, vol. 3855, pp. 49–64. Springer, Heidelberg (2006)
17. 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, Heidelberg (2007)
18. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.-K.: General decidability theorems for infinite-state systems. In: Proc.1th Annual IEEE Symp. on Logic in Computer Science (LICS), pp. 313–321. IEEE Computer Society Press, Los Alamitos (1996)
19. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theoretical Computer Science 256(1-2), 1–2 (1997)
20. Boigelot, B.: On iterating linear transformations over recognizable sets of integers. Theoretical Computer Science 309(2), 413–468 (2003)
21. Dams, D.: Comparing abstraction refinement algorithms. Electr. Notes Theor. Comput. Sci 89(3) (2003)