A Forward-Backward Abstraction Refinement Algorithm

Francesco Ranzato, Olivia Rossi Doria, and Francesco Tapparo

Dipartimento di Matematica Pura ed Applicata
Università di Padova, Italy

Abstract. Abstraction refinement-based model checking has become a standard approach for efficiently verifying safety properties of hardware/software systems. Abstraction refinement algorithms can be guided by counterexamples generated from abstract transition systems or by fixpoints computed in abstract domains. Cousot, Ganty and Raskin recently put forward a new fixpoint-guided abstraction refinement algorithm that is based on standard abstract interpretation and improves the state-of-the-art, also for counterexample-driven methods. This work presents a new fixpoint-guided abstraction refinement algorithm that enhances the Cousot-Ganty-Raskin's procedure. Our algorithm is based on three main ideas: (1) within each abstraction refinement step, we perform multiple forward-backward abstract state space traversals; (2) our abstraction is a disjunctive abstract domain that is used both as an overapproximation and an underapproximation; (3) we maintain and iteratively refine an overapproximation M of the set of states that belong to some minimal (i.e. shortest) counterexample to the given safety property so that each abstract state space traversal is limited to the states in M.

1 Introduction

Abstraction techniques are widely used in model checking to blur some properties of the concrete model and then to design a reduced abstract model where to run the verification algorithm [3]. Abstraction provides a successful solution to the state-explosion problem that arises in model checking systems with parallel components [4]. CounterExample-Guided Abstraction Refinement (CEGAR), pioneered by Clarke et al. [5], is become the standard methodology for applying abstraction to model checking. The basic idea of the CEGAR approach is as follows: if the abstract model checker return "YES" then the system satisfies the property; otherwise the abstract model checker returns an abstract counterexample to the property, that is checked to determine whether it corresponds to a real counterexample or not; it it does then return "NO" otherwise refine the abstract model in order to remove that spurious counterexample. Many different algorithms that implement the CEGAR approach have been suggested. Most CEGAR-based model checkers — like BLAST [16,17], MAGIC [5,2] and SLAM [1] — deal with counterexamples that are paths of abstract states, i.e. paths in an abstract transition system defined by an abstract state space and an abstract transition relation. Most often, model checkers aim at verifying so-called safety properties, i.e., states that can be reached from an initial state are always safe. Hence, safety verification consists in automatically proving that systems cannot go wrong.

F. Logozzo et al. (Eds.): VMCAI 2008, LNCS 4905, pp. 248–262, 2008.
© Springer-Verlag Berlin Heidelberg 2008

Recently, Cousot, Ganty and Raskin [10] (more details are given by the PhD thesis [12]) put forward a new fixpoint-guided abstraction refinement algorithm, here called CGR, for checking safety properties. The CGR algorithm is based on a number of interesting features. (1) CGR maintains and refines generic abstract domains that are defined within the standard abstract interpretation framework, as opposed to most other CEGAR-based algorithms that consider as abstract models a partition of the state space. (2) The refinement of the current abstract domain A is driven by the abstract fixpoint computed within A and not by a path-based counterexample. (3) CGR computes overapproximations of both least and greatest fixpoints, and these two analyses are made iteratively synergic since the current abstract fixpoint computation is limited by the abstract value provided by the previous abstract fixpoint computation.

We isolated a number of examples where the behavior of the CGR algorithm could be improved, in particular where one could abstractly conclude that the system is safe or unsafe without resorting to abstraction refinements. This work puts forward a new fixpoint-guided abstraction refinement algorithm for safety verification, called FBAR (Forward-Backward Abstraction Refinement), that is designed as an enhancement of the CGR procedure that integrates some new ideas.

(i) FBAR maintains and refines a disjunctive abstract domain μ that overapproximates any set S of states by $\mu(S) \supseteq S$. While a generic abstract domain can be viewed as a set of subsets of states that is closed under arbitrary intersections, a disjunctive abstract domain must also be closed under arbitrary unions. The advantage of dealing with a disjunctive abstraction μ is given by the fact that μ can be simultaneously used both as an over- and under-approximating abstraction. As an additional advantage, it turns out that disjunctive abstractions can be efficiently represented and refined, as shown in [20].

(ii) FBAR computes and maintains an overapproximation M of the set of states that occur in some *minimal* safety counterexample. A safety counterexample is simply a path from an initial state to an unsafe state. However, counterexamples may be redundant, namely may contain shorter sub-counterexamples. A counterexample is thus called minimal when it cannot be reduced. It can be therefore helpful to focus on minimal counterexamples rather than on generic counterexamples. In FBAR, abstract fixpoints are always computed within the overapproximation M and other than being used for safety checking they are also used for refining M.

(iii) Each abstraction refinement step in FBAR consists of two loops that check whether the system can be proved safe/unsafe by using the current abstraction. The safety loop is based on a combined forward-backward abstract exploration of the portion of the state space limited by M. This combined forward-backward abstract computation was first introduced by Cousot [6]. The unsafety loop relies on an iterated combination of two abstract fixpoints: the first one is an overapproximation of the states in M that are globally safe and is computed by using the current abstraction μ as an overapproximation; the second one is instead an underapproximation of the states in M that can reach an unsafe state and is computed by viewing μ as an underapproximating abstraction.

We prove that FBAR is a correct algorithm for safety verification and that, analogously to CGR, it terminates when the concrete domain satisfies the descending chain

condition. We formally compare FBAR and CGR by showing that FBAR improves CGR in the following sense: if CGR terminates on a given disjunctive abstraction μ with no refinement then FBAR also terminates on μ, while the converse is not true.

Related Work. We discussed above the relationship with the CGR algorithm in [10]. Gulavani and Rajamani [15] also describe a fixpoint-driven abstraction refinement algorithm for safety verification. This algorithm relies on using widening operators during abstract fixpoint computations. When the abstract fixpoint is inconclusive, this algorithm does not refine the abstract domain but determines which iteration of the abstract fixpoint computation was responsible of the loss of precision so that widening is replaced with a concrete union and the abstract computation is re-started from that iteration. Manevich et al. [18] put forward an abstraction refinement algorithm for safety verification that runs over disjunctive abstract domains. However, this algorithm does not compute abstract fixpoints but instead computes paths of abstract values so that the abstraction refinement is based on counterexamples defined as sequences of abstract values.

2 Background

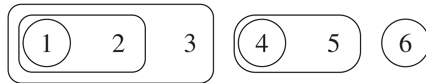
Notation and Orders. Let Σ be any set. If $S \subseteq \Sigma$ then $\neg S$ denotes the complement set $\Sigma \smallsetminus S$ when Σ is clear from the context. A set S of one-digit integers is often written in a compact form without brackets and commas like $S = 1357$ that stands for $\{1, 3, 5, 7\}$. $\mathrm{Part}(\Sigma)$ denotes the set of partitions of Σ. If $R \subseteq \Sigma \times \Sigma$ is any relation then $R^* \subseteq \Sigma \times \Sigma$ denotes the reflexive and transitive closure of R. Posets and complete lattices are denoted by $P_\leq$ where $\leq$ is the partial order. A function f between complete lattices is additive (co-additive) when f preserves least upper (greatest lower) bounds. If $f : P \to P$ then $\mathrm{lfp}(f)$ and $\mathrm{gfp}(f)$ denote, resp., the least and greatest fixpoints of f, when they exist.

Abstract Domains. In standard Cousot and Cousot's abstract interpretation, abstract domains (or abstractions) can be equivalently specified either by Galois connections/insertions through α/γ abstraction/concretization maps or by upper closure operators (uco's) [7]. These two approaches are equivalent, modulo isomorphic representations of domain's objects. The closure operator approach has the advantage of being independent from the representation of domain's objects and is therefore appropriate for reasoning on abstract domains independently from their representation. Given a state space Σ, the complete lattice $\wp(\Sigma)_\subseteq$, i.e. the powerset of Σ ordered by the subset relation, plays the role of concrete domain. Let us recall that an operator $\mu : \wp(\Sigma) \to \wp(\Sigma)$ is a uco on $\wp(\Sigma)$, that is an overapproximating abstract domain of $\wp(\Sigma)$, when μ is monotone, idempotent and extensive (i.e., overapproximating: $X \subseteq \mu(X)$). Each closure μ is uniquely determined by its image $\mathrm{img}(\mu) = \{\mu(X) \in \wp(\Sigma) \mid X \in \wp(\Sigma))\}$ as follows: for any $X \subseteq \Sigma$, $\mu(X) = \cap\{Y \in \mathrm{img}(\mu) \mid X \subseteq Y\}$. On the other hand, a set of subsets $A \subseteq \wp(\Sigma)$ is the image of some closure on $\wp(\Sigma)$ iff A is closed under arbitrary intersections, i.e. $A = \mathrm{Cl}_\cap(A) \stackrel{\text{def}}{=} \{\cap S \mid S \subseteq A\}$ (in particular, note that $\mathrm{Cl}_\cap(A)$ always contains $\Sigma = \cap\varnothing$). This makes clear that an abstract domain μ guarantees that for any concrete set of states X, $\mu(X)$ is the best (i.e., more precise) overapproximation of X

in μ. We denote by $\mathrm{Abs}(\Sigma)$ the set of abstract domains of $\wp(\Sigma)_\subseteq$. Capital letters like $A, A' \in \mathrm{Abs}(\Sigma)$ are sometimes used for denoting abstract domains. By a slight abuse of notation, a given abstract domain $A \in \mathrm{Abs}(\Sigma)$ can be viewed and used both as a set of subsets of Σ and as an operator on $\wp(\Sigma)$ when the context allows us to disambiguate this use. If $A_1, A_2 \in \mathrm{Abs}(\Sigma)$ then A_1 is more precise than (or is a refinement of) A_2 when $A_1 \supseteq A_2$. $\mathrm{Abs}(\Sigma)_\supseteq$ is called the (complete) lattice of abstract domains of $\wp(\Sigma)$.

Let $f : \wp(\Sigma) \rightarrow \wp(\Sigma)$ be a concrete semantic function, like a predicate transformer, and, given an abstraction $\mu \in \mathrm{Abs}(\Sigma)$, let $f^\sharp : \mu \rightarrow \mu$ be a corresponding abstract function on μ. Then, $f^\sharp$ is a correct approximation of f in μ when for any $X \in \wp(\Sigma)$, $\mu(f(X)) \subseteq f^\sharp(\mu(X))$. The abstract function $f^\mu : \mu \rightarrow \mu$ defined as $f^\mu \stackrel{\text{def}}{=} \mu \circ f$ is called the best correct approximation of f in μ because for any correct approximation $f^\sharp$, for any $X \in \mu$, $f^\mu(X) \subseteq f^\sharp(X)$ always holds.

Disjunctive Abstract Domains. An abstract domain $\mu \in \mathrm{Abs}(\wp(\Sigma))$ is disjunctive (or additive or a powerset abstract domain) when μ is additive, i.e. μ preserves arbitrary unions. This happens exactly when the image $\mathrm{img}(\mu)$ is closed under arbitrary unions, i.e., $\mu = \mathrm{Cl}_\cup(\mu) \stackrel{\text{def}}{=} \{\cup S \mid S \subseteq \mu\}$ (in particular, note that $\mathrm{Cl}_\cup(\mu)$ always contains $\varnothing = \cup\varnothing$). Hence, a disjunctive abstract domain is a set of subsets of states that is closed under both arbitrary intersections and unions. The intuition is that a disjunctive abstract domain does not loose precision in approximating concrete set unions. We denote by $\mathrm{dAbs}(\wp(\Sigma)) \subseteq \mathrm{Abs}(\wp(\Sigma))$ the set of disjunctive abstract domains. A disjunctive abstraction μ can be specified just by defining how any singleton $\{x\} \subseteq \Sigma$ is approximated by $\mu(\{x\})$, because the approximation of a generic subset $X \subseteq \Sigma$ can be obtained through set unions as $\mu(X) = \cup_{x \in X} \mu(\{x\})$. We exploit this property for representing disjunctive abstractions through diagrams. As an example, the following diagram:



denotes the disjunctive abstract domain μ that is determined by the following behaviour on singletons: $\mu(1) = 1$, $\mu(2) = 12$, $\mu(3) = 123$, $\mu(4) = 4$, $\mu(5) = 45$, $\mu(6) = 6$, so that μ is the closure under unions of the set $\{1, 12, 123, 4, 45, 6\}$.

Underapproximating Abstract Domains. It turns out that a disjunctive abstract domain $\mu \in \mathrm{Abs}(\wp(\Sigma)_\subseteq)$ can be also viewed as an underapproximating abstract domain, namely an abstraction of the concrete domain $\wp(\Sigma)_\supseteq$ where the approximation order is reversed. Formally, this is specified by the closure $\widetilde{\mu} \in \mathrm{Abs}(\wp(\Sigma)_\supseteq)$ that is defined as the adjoint of μ as follows: for any $X \subseteq \Sigma$, $\widetilde{\mu}(X) \stackrel{\text{def}}{=} \cup\{Y \subseteq \Sigma \mid Y = \mu(Y) \subseteq X\}$. An underapproximating abstraction is thus determined by the behaviour on the sets $\neg x \stackrel{\text{def}}{=} \Sigma \smallsetminus \{x\}$ because, for any $X \subseteq \Sigma$, $\widetilde{\mu}(X) = \cap_{x \notin X} \widetilde{\mu}(\neg x)$. For example, for the above disjunctive abstraction μ, we have that $\widetilde{\mu}(\neg 1) = 456$, $\widetilde{\mu}(\neg 2) = 1456$, $\widetilde{\mu}(\neg 3) = 12456$, $\widetilde{\mu}(\neg 4) = 1236$, $\widetilde{\mu}(\neg 5) = 12346$ and $\widetilde{\mu}(\neg 6) = 12345$.

Transition Systems. A transition system $\mathcal{T} = (\Sigma, R)$ consists of a set Σ of states and a transition relation $R \subseteq \Sigma \times \Sigma$, that is also denoted in infix notation by $\rightarrow$. As usual

in model checking, we assume that the relation R is total, i.e., for any $s \in \Sigma$ there exists some $t \in \Sigma$ such that $s \to t$. The set of finite and infinite paths in $\mathcal{T}$ is denoted by $\mathrm{Path}(\mathcal{T})$. For any $\Pi \subseteq \mathrm{Path}(\mathcal{T})$, $\mathrm{states}(\Pi) \subseteq \Sigma$ denotes the set of states that occur in some path $\pi \in \Pi$. If $\pi \in \mathrm{Path}(\mathcal{T})$ is finite then $\mathrm{first}(\pi), \mathrm{last}(\pi) \in \Sigma$ denote, resp., the first and last states of π.

Standard predicate transformers $\mathrm{pre}, \widetilde{\mathrm{pre}}, \mathrm{post}, \widetilde{\mathrm{post}} : \wp(\Sigma) \to \wp(\Sigma)$ are defined as usual:

- $\mathrm{pre}(X) \stackrel{\mathrm{def}}{=} \{a \in \Sigma \mid \exists b \in X.\ a \to b\}$;
- $\mathrm{post}(X) \stackrel{\mathrm{def}}{=} \{b \in \Sigma \mid \exists a \in X.\ a \to b\}$;
- $\widetilde{\mathrm{pre}}(X) \stackrel{\mathrm{def}}{=} \neg\, \mathrm{pre}(\neg X) = \{a \in \Sigma \mid \forall b.\ a \to b \;\Rightarrow\; b \in X\}$;
- $\widetilde{\mathrm{post}}(X) \stackrel{\mathrm{def}}{=} \neg\, \mathrm{post}(\neg X) = \{b \in \Sigma \mid \forall a.\ a \to b \;\Rightarrow\; a \in X\}$.

Let us remark that pre and post are additive while $\widetilde{\mathrm{pre}}$ and $\widetilde{\mathrm{post}}$ are co-additive. We use the notation $\mathrm{pre}^*, \widetilde{\mathrm{pre}}^*, \mathrm{post}^*, \widetilde{\mathrm{post}}^*$ when the reflexive-transitive closure R^* is considered instead of R. Let us recall the following standard fixpoint characterizations:

$$\mathrm{pre}^*(X) = \mathrm{lfp}(\lambda Z.X \cup \mathrm{pre}(Z)); \qquad \mathrm{post}^*(X) = \mathrm{lfp}(\lambda Z.X \cup \mathrm{post}(Z));$$
$$\widetilde{\mathrm{pre}}^*(X) = \mathrm{gfp}(\lambda Z.X \cap \widetilde{\mathrm{pre}}(Z)); \qquad \widetilde{\mathrm{post}}^*(X) = \mathrm{gfp}(\lambda Z.X \cap \widetilde{\mathrm{post}}(Z)).$$

Safety Verification Problems. Let $\mathit{Init} \subseteq \Sigma$ be a set of initial states and $\mathit{Safe} \subseteq \Sigma$ a set of safe states. We denote by $\mathit{NInit} \stackrel{\mathrm{def}}{=} \neg \mathit{Init}$ the set of noninitial states and by $\mathit{Bad} = \neg \mathit{Safe}$ the set of bad (i.e. unsafe) states. The set of reachable states is $\mathrm{post}^*(\mathit{Init})$. The set of states that are globally safe is $\widetilde{\mathrm{pre}}^*(\mathit{Safe})$. The set of states that can reach a bad state is $\mathrm{pre}^*(\mathit{Bad})$. The set of states that can be reached only from noninitial states is $\widetilde{\mathrm{post}}^*(\mathit{NInit})$. Note that $\mathrm{pre}^*(\mathit{Bad}) = \neg \widetilde{\mathrm{pre}}^*(\mathit{Safe})$ and $\widetilde{\mathrm{post}}^*(\mathit{NInit}) = \neg\, \mathrm{post}^*(\mathit{Init})$.

A system $\mathcal{T}$ is safe when any reachable state is safe, i.e. $\mathrm{post}^*(\mathit{Init}) \subseteq \mathit{Safe}$, or, equivalently, when one of the following equivalent conditions holds: $\mathit{Init} \subseteq \widetilde{\mathrm{pre}}^*(\mathit{Safe})$ $\Leftrightarrow \mathrm{pre}^*(\mathit{Bad}) \subseteq \mathit{NInit} \Leftrightarrow \mathit{Bad} \subseteq \widetilde{\mathrm{post}}^*(\mathit{NInit})$. A safety verification problem is then specified by a transition system $\mathcal{T} = \langle \Sigma, R, \mathit{Init}, \mathit{Safe} \rangle$ that also defines initial and safe states and consists in checking whether $\mathcal{T}$ is safe (OK) or not (KO).

3 Cousot-Ganty-Raskin's Algorithm

The Cousot-Ganty-Raskin's algorithm, here denoted by CGR, is recalled in Fig. 1. In each abstraction refinement step $i \geq 0$, CGR abstractly computes two overapproximations R_i and S_i of least/greatest fixpoints and a concrete value Z_{i+1} that is added to the current abstract domain μ_i for the purpose of refining it. The correctness of CGR follows from the following three main invariants: for all $i \geq 0$: (1) $Z_{i+1} \subseteq S_i \subseteq R_i \subseteq Z_i$; (2) if the system is safe then $\mathrm{post}^*(\mathit{Init}) \subseteq R_i$, i.e. R_i overapproximates the reachable states; (3) $R_i \subseteq \widetilde{\mathrm{pre}}^i(\mathit{Safe})$, i.e. R_i underapproximates the states that remain inside Safe along paths of length $\leq i$.

CGR admits a dual version, denoted by $\mathrm{CGR}^{\leftarrow}$, where the transition relation is reversed, namely where R is replaced by R^{-1}, Init by Bad and Safe by NInit (so that

```
Data: Init initial states, Safe safe states such that Init ⊆ Safe
Data: μ0 ∈ uco(℘(Σ)) initial abstract domain such that Safe ∈ μ0
1  begin
2    Z0 := Safe;
3    for i := 0, 1, 2, ... do
4      Ri := lfp(λX.μi(Zi ∩ (Init ∪ post(X))));
5      if μi(Init ∪ post(Ri)) ⊆ Zi then return OK;
6      else
7        Si := gfp(λX.μi(Ri ∩ p̃re(X));
8        if μi(Init) ⊈ Si then return KO;
9        else
10         Zi+1 := Si ∩ p̃re(Si);
11         μi+1 := Cl∩(μi ∪ {Zi+1});
12 end
```

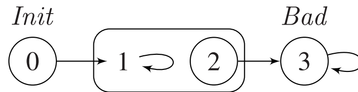
Fig. 1. CGR Algorithm

post and $\widetilde{\mathrm{pre}}$ become, respectively, pre and $\widetilde{\mathrm{post}}$). Thus, while CGR performs a forward abstract exploration of the state space through post, CGR$^{\leftarrow}$ proceeds instead backward through pre.

3.1 Where CGR Could Be Improved

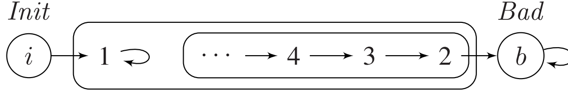
We isolated a number of examples where the CGR algorithm could be improved.

Example 3.1. Let us consider the safety problem represented by the following diagram that also specifies a disjunctive abstract domain $\mu_0 \in \mathrm{Abs}(\wp(\Sigma))$.



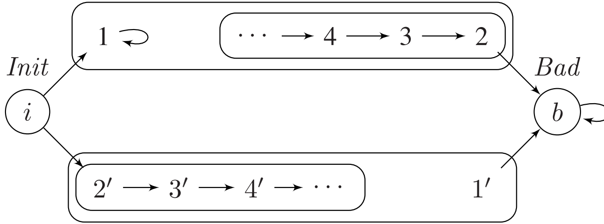
CGR computes the following sequence: $Z_0 = 012$, $R_0 = 012$, $S_0 = 012$, $Z_1 = 01$, $R_1 = 01$ and then outputs OK. Let us observe that $S_0 = 012$ because $\mu_0(R_0 \cap \widetilde{\mathrm{pre}}(R_0)) = \mu_0(01) = 012$. Thus, CGR needs to refine the abstraction μ_0 by adding the singleton $\{1\}$ to μ_0. However, one could abstractly conclude that the system is safe already through the abstraction μ_0. In fact, one could abstractly explore backward the state space by computing the following fixpoint: $T_0 = \mathrm{lfp}(\lambda X.\mu_0(\mathit{NInit} \cap (\mathit{Bad} \cup \mathrm{pre}(X))))$. Thus, $T_0 = 23$ and since $\mu_0(\mathit{Bad} \cup \mathrm{pre}(T_0)) \subseteq T_0$ one can conclude that the system is safe. Thus, the dual algorithm CGR$^{\leftarrow}$ is able to conclude that the system is safe with no abstraction refinement.

Along the same lines, it turns out that CGR even does not terminate when applied to the following infinite state system, although the abstraction is finite, while a backward abstract exploration would allow to conclude that the system is safe.



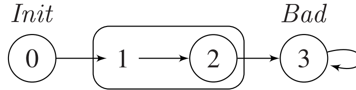
In fact, CGR does not terminate because it would compute the following infinite sequence: $Z_0 = i \cup 12345...$, $R_0 = i \cup 12345...$, $S_0 = i \cup 12345...$, $Z_1 = i \cup 1345...$, $R_1 = i \cup 1345...$, $S_1 = i \cup 1345...$, $Z_2 = i \cup 145....$ Instead, one could proceed backward by computing the following abstract fixpoint: $T_0 = \mathrm{lfp}(\lambda X.\mu_0(\mathit{NInit} \cap (\mathit{Bad} \cup \mathrm{pre}(X)))) = b \cup 2345....$ Hence, since $\mu_0(\mathit{Bad} \cup \mathrm{pre}(T_0)) \subseteq T_0$ we can conclude that the system is safe. Thus, here again, $\mathrm{CGR}^{\leftarrow}$ is able to infer that the system is safe with no abstraction refinement.

Let us consider now the following infinite state system.



In this case, it turns out that neither CGR nor $\mathrm{CGR}^{\leftarrow}$ terminate. In fact, similarly to the above examples, it is simple to check that both CGR and $\mathrm{CGR}^{\leftarrow}$ would compute infinite sequences of abstract values. However, it is still possible to derive that the system is safe with no abstraction refinement. In fact, we can first compute the following forward abstract fixpoint $U_0 = \mathrm{lfp}(\lambda X.\mu_0(\mathit{Safe} \cap (\mathit{Init} \cup \mathrm{post}(X)))) = i \cup 1234... \cup 2'3'4'....$ Then, we can explore backward starting from Bad but remaining inside $U_0 \cup \mathit{Bad}$, namely we compute the following backward abstract fixpoint $V_0 = \mathrm{lfp}(\lambda X.\mu_0((U_0 \cup \mathit{Bad}) \cap (\mathit{Bad} \cup \mathrm{pre}(X)))) = b \cup 234....$ We can now conclude that the system is safe because $\mu_0(\mathit{Bad} \cup \mathrm{pre}(V_0)) \subseteq V_0$. □

Example 3.2. Let us consider the following safety problem and disjunctive abstract domain $\mu_0 \in \mathrm{Abs}(\wp(\Sigma))$.



In this case, CGR computes the following sequence: $Z_0 = 012$, $R_0 = 012$, $S_0 = 012$, $Z_1 = 01$, $R_1 = 01$, $S_1 = \varnothing$ and then outputs KO. Thus, CGR needs to refine the abstraction μ_0 by adding the singleton $\{1\}$ to μ_0. However, one could abstractly conclude that the system is not safe already through the abstraction μ_0 by viewing μ_0 as an underapproximating abstraction, i.e. by considering the underapproximating abstraction $\widetilde{\mu}_0$. In fact, one could abstractly explore the state space backward by computing the following abstract fixpoint: $T_0 = \mathrm{lfp}(\lambda X.\widetilde{\mu}_0(\mathit{Bad} \cup \mathrm{pre}(X))) = 0123$. Since T_0 is an underapproximation of the set of states that can reach a bad state and T_0 contains some initial state we can conclude that the system is unsafe. □

These examples suggested us to design an abstraction refinement algorithm that improves the CGR algorithm by integrating a combined forward-backward abstract exploration of the state space and by using disjunctive abstract domains that can be exploited both as overapproximating and underapproximating abstractions.

4 Restricted Predicate Transformers

Let $M \subseteq \Sigma$ be a fixed set of states of interest. In our context, M will play the role of a portion of the state space that limits the abstract search space of our abstraction refinement algorithm. Let us define the following *restricted* (to the states in M) predicate transformers $\mathrm{pre}_M, \mathrm{post}_M, \widetilde{\mathrm{pre}}_M, \widetilde{\mathrm{post}}_M : \wp(\Sigma) \to \wp(\Sigma)$ as follows:

- $\mathrm{pre}_M(X) \stackrel{\text{def}}{=} M \cap \mathrm{pre}(M \cap X)$;
- $\mathrm{post}_M(X) \stackrel{\text{def}}{=} M \cap \mathrm{post}(M \cap X)$;
- $\widetilde{\mathrm{pre}}_M(X) \stackrel{\text{def}}{=} \neg\, \mathrm{pre}_M(\neg X) = \neg M \cup \widetilde{\mathrm{pre}}(\neg M \cup X)$ $= \{a \in \Sigma \mid \forall b.\, (a \to b \;\&\; a, b \in M) \;\Rightarrow\; b \in X\}$;
- $\widetilde{\mathrm{post}}_M(X) \stackrel{\text{def}}{=} \neg\, \mathrm{post}_M(\neg X) = \neg M \cup \widetilde{\mathrm{post}}(\neg M \cup X)$ $= \{b \in \Sigma \mid \forall a.\, (a \to b \;\&\; a, b \in M) \;\Rightarrow\; a \in X\}$.

Thus, M-restricted predicate transformers only consider states that belong to M. In fact, if $\mathrm{pre}_M, \mathrm{post}_M, \widetilde{\mathrm{pre}}_M, \widetilde{\mathrm{post}}_M$ are viewed as mappings $\wp(M) \to \wp(M)$ — i.e., both the argument and the image of the M-restricted transformers are taken as subsets of M — then they coincide with the corresponding standard predicate transformers on the M-restricted transition system $\mathcal{T}_{/M} = \langle M, R_{/M} \rangle$. Let us remark that, analogously to the unrestricted case, pre_M, post_M are additive functions and $\widetilde{\mathrm{pre}}_M$, $\widetilde{\mathrm{post}}_M$ are co-additive functions. We also consider the following fixpoint definitions:

- $\mathrm{pre}^*_M(X) \stackrel{\text{def}}{=} \mathrm{lfp}(\lambda Z.\, (X \cap M) \cup \mathrm{pre}_M(Z))$ $= \{x \in \Sigma \mid \exists y \in X.\, x \to^* y \;\&\; \mathrm{states}(x \to^* y) \subseteq M\}$;
- $\mathrm{post}^*_M(X) \stackrel{\text{def}}{=} \mathrm{lfp}(\lambda Z.\, (X \cap M) \cup \mathrm{post}_M(Z))$ $= \{y \in \Sigma \mid \exists x \in X.\, x \to^* y \;\&\; \mathrm{states}(x \to^* y) \subseteq M\}$;
- $\widetilde{\mathrm{pre}}^*_M(X) \stackrel{\text{def}}{=} \mathrm{gfp}(\lambda Z.\, (X \cup \neg M) \cap \widetilde{\mathrm{pre}}_M(Z))$ $= \{x \in \Sigma \mid \forall y.\, (x \to^* y \;\&\; \mathrm{states}(x \to^* y) \subseteq M) \;\Rightarrow\; y \in X\}$;
- $\widetilde{\mathrm{post}}^*_M(X) \stackrel{\text{def}}{=} \mathrm{gfp}(\lambda Z.\, (X \cup \neg M) \cap \widetilde{\mathrm{post}}_M(Z))$ $= \{y \in \Sigma \mid \forall x.\, (x \to^* y \;\&\; \mathrm{states}(x \to^* y) \subseteq M) \;\Rightarrow\; x \in X\}$.

Hence, we have that $x \in \mathrm{pre}^*_M(X)$ iff x may reach X through a path inside M, while $x \in \widetilde{\mathrm{pre}}^*_M(X)$ iff x inside M can only reach states in X. Let us note that, analogously to the unrestricted case, $\widetilde{\mathrm{pre}}^*_M(\neg X) = \neg\, \mathrm{pre}^*_M(X)$ and $\widetilde{\mathrm{post}}^*_M(\neg X) = \neg\, \mathrm{post}^*_M(X)$. Moreover, $\mathrm{pre}^*_M(X) \subseteq \mathrm{pre}^*(X)$ and $\mathrm{post}^*_M(X) \subseteq \mathrm{post}^*(X)$ while $\widetilde{\mathrm{pre}}^*(X) \subseteq \widetilde{\mathrm{pre}}^*_M(X)$ and $\widetilde{\mathrm{post}}^*(X) \subseteq \widetilde{\mathrm{post}}^*_M(X)$.

Example 4.1. Consider the safety verification problem depicted in Fig. 2 where the gray states determine the restricted space $M = 0134568$. It turns out that $\mathrm{post}^*_M(\mathit{Init}) = 013468$, $\mathrm{pre}^*_M(\mathit{Bad}) = 01368$, $\widetilde{\mathrm{post}}^*_M(\mathit{NInit}) = 2579$ and $\widetilde{\mathrm{pre}}^*_M(\mathit{Safe}) = 24579$.

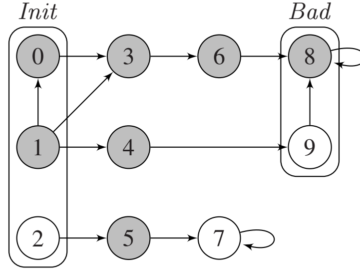


Fig. 2. Restricted predicated transformers

Note, for example, that $9 \in \text{post}^*(\mathit{Init}) = 0134689$ but $9 \not\in \text{post}^*_M(\mathit{Init})$. Also, $4 \in \widetilde{\text{pre}}^*_M(\mathit{Safe})$ but $4 \not\in \widetilde{\text{pre}}^*(\mathit{Safe}) = 257$ because there is no path that begins with 4 and remains inside M. □

Thus, when using a M-restricted predicate transformer instead of a standard (i.e. unrestricted) predicate transformer it is enough to consider only the states belonging to M. It should be clear that when M is much smaller than the whole state space Σ such a restriction to states in M may induce space and time savings.

5 Minimal Counterexamples

One main idea of our abstraction refinement algorithm consists in overapproximating the set of states that belong to some safety counterexample, i.e. a finite path from an initial state to a bad state. However, a counterexample π may be redundant, namely π might contain a shorter sub-path that still is a safety counterexample. For example, in the transition system in Fig. 2, the path $\pi = 103688$ is a safety counterexample because it begins with an initial state and ends with a bad state although π is redundant because it contains a sub-path $\pi' = 0368$ that is a counterexample. Our algorithm will compute and maintain an overapproximation of the states that belong to counterexamples that are not reducible. Such counterexamples are called minimal counterexamples.

Let us formalize the above notions. Let $\mathcal{T} = \langle \Sigma, R, \mathit{Init}, \mathit{Safe} \rangle$ specify a safety problem. A (safety) counterexample is a finite path $\pi \in \text{Path}(\mathcal{T})$ such that $\text{first}(\pi) \in \mathit{Init}$ and $\text{last}(\pi) \in \mathit{Bad}$. A *minimal counterexample* is a counterexample $\pi \in \text{Path}(\mathcal{T})$ such that $\text{states}(\pi) \smallsetminus \{\text{first}(\pi), \text{last}(\pi)\} \subseteq \mathit{Safe} \cap \mathit{NInit}$. We define $\text{MinCex} \stackrel{\text{def}}{=} \{\pi \in \text{Path}(\mathcal{T}) \mid \pi \text{ is a minimal counterexample}\}$.

Assume that M is an overapproximation of the states that occur in some minimal counterexample, i.e. $\text{states}(\text{MinCex}) \subseteq M$. Then, we provide a characterization of safe systems that only considers states in M: it turns out that a system is safe iff any state that is reachable from an initial state through a path inside M is safe.

Lemma 5.1. *If* $\text{states}(\text{MinCex}) \subseteq M$ *and* $\mathit{Init} \subseteq \mathit{Safe}$ *then the system* $\mathcal{T}$ *is safe iff* $\text{post}^*_M(\mathit{Init}) \subseteq M \cap \mathit{Safe}$ *iff* $\text{pre}^*_M(\mathit{Bad}) \subseteq M \cap \mathit{NInit}$.

6 A Forward-Backward Abstraction Refinement Algorithm

The Forward-Backward Abstraction Refinement algorithm FBAR is defined in Fig. 3. FBAR takes as input a safety verification problem $\mathcal{T} = \langle \Sigma, R, Init, Safe \rangle$ and a disjunctive abstraction $\mu_0 \in \mathrm{dAbs}(\wp(\Sigma))$. The main ideas and features of FBAR are summarized in the following list.

```
    Data: Init initial states, Safe safe states such that Init ⊆ Safe
    Data: Bad = Σ ∖ Safe bad states, NInit = Σ ∖ Init noninitial states
    Data: μ0 ∈ dAbs(℘(Σ)) initial disjunctive abstract domain such that
              Safe, Bad, Init, NInit ∈ μ0
 1  begin
 2      M := Σ; U := Safe; V := NInit; X := Safe; Y := Bad;
 3      for i := 0, 1, 2, ... do
 4          while true do
 5              U' := lfp(λZ.μi(M ∩ U ∩ (Init ∪ post(Z))));
 6              if μi(M ∩ (Init ∪ post(U'))) ⊆ U then return OK;
 7              M := U' ∪ μi(V ∩ Bad ∩ post(U'));
 8              V' := lfp(λZ.μi(M ∩ V ∩ (Bad ∪ pre(Z))));
 9              if μi(M ∩ (Bad ∪ pre(V'))) ⊆ V then return OK;
10              M := V' ∪ μi(U' ∩ Init ∩ pre(V'));
11              if (U' = U and V' = V) then break;
12              U, V := U', V';
13          X := M ∩ X; Y := M ∩ Y;
14          while true do
15              X := X ∩ μi(M ∖ Y);
16              X' := gfp(λZ.μi(X ∩ p̃re_M(Z)));
17              if Init ∩ M ⊈ X' then return KO;
18              Y := Y ∪ μ̃i(M ∖ X');
19              Y' := lfp(λZ.μ̃i(Y ∪ pre_M(Z)));
20              if Y' ⊈ NInit then return KO;
21              if (X' = X and Y' = Y) then break;
22              X, Y := X', Y';
23          X := X ∩ p̃re_M(X);
24          if X = X' then return OK;
25          μ(i+1) := Cl∩,∪(μi ∪ {X});
26  end
```

Fig. 3. FBAR Algorithm

(A) The loop at lines 4-12 computes and maintains an overapproximation M of the states that occur in some minimal counterexample by relying on a combined forward-backward abstract exploration of the state space. Such a combined forward-backward abstract computation was first described by Cousot [6] and further

investigated and applied in [8,9,19]. The following procedure is iterated: we start from $M \cap \mathit{Init}$ and abstractly go forward within M through post_M (line 5); then, we come back by starting from $M \cap \mathit{Bad}$ and abstractly going backward within M through pre_M (line 8). The abstract sets U and V are the results of these, resp., forward and backward abstract fixpoint computations. The following invariant properties hold (cf. Lemma 6.2 (1)): U is an overapproximation of the safe states that can be reached through a path within M, while V is an overapproximation of the noninitial states that can reach a bad state through a path within M. The combined forward-backward computation of U and V allows us to iteratively refine the overapproximation M of $\mathrm{states}(\mathrm{MinCex})$ (lines 7 and 10).

(B) The OK condition at line 6 implies that $\mathrm{post}^*_M(\mathit{Init}) \subseteq U$, so that $\mathrm{post}^*_M(\mathit{Init}) \subseteq M \cap \mathit{Safe}$, and therefore, by Lemma 5.1, the system is safe. Analogously, for the "backward" OK condition at line 9.

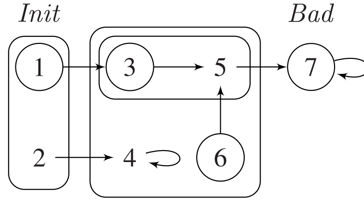
(C) The loop at lines 14-22 computes iteratively the abstract sets X and Y as follows. X is an overapproximation of the states in M that are globally safe and is computed at line 16 as a greatest fixpoint of the best correct approximation in μ_i of $\widetilde{\mathrm{pre}}_M$. On the other hand, Y is an underapproximation of the states in M that can reach a bad state and is computed at line 19 as a least fixpoint of the best correct approximation of pre_M w.r.t. the underapproximating abstraction $\widetilde{\mu}_i$. This is formally stated by Lemma 6.2 (3). While the sequence of computed X's forms a descending chain of abstract sets, the Y's give rise to an ascending chain of abstract sets. These abstract computations are iterated because the abstract set Y may help in refining X and, vice versa, X may help in refining Y. In fact, observe that the states in $M \smallsetminus Y$ form a superset of the states in M that are globally safe, so that the overapproximation X can be refined by intersection with the abstract set $\mu_i(M \smallsetminus Y)$. A dual reasoning holds for Y, where we exploit the fact that μ_i is a disjunctive abstraction and therefore $\widetilde{\mu}_i$ is an underapproximating abstraction.

(D) Since $X' \supseteq M \cap \widetilde{\mathrm{pre}}^*(\mathit{Safe})$, the KO condition at line 17 implies that $\mathit{Init} \cap M \not\subseteq M \cap \widetilde{\mathrm{pre}}^*(\mathit{Safe})$, namely that $\mathit{Init} \not\subseteq \widetilde{\mathrm{pre}}^*(\mathit{Safe})$, so that the system is unsafe. Analogously, for the "underapproximating" KO condition at line 20.

(E) At the exit of the loop at lines 14-22, we perform a concrete step of computation at line 23 by calculating a refiner set $\mathit{Ref} = X \cap \widetilde{\mathrm{pre}}_M(X)$. In contrast to the algorithm CGR where the refiner Z_{i+1} cannot already be in the abstraction μ_i, here it may happen that $X = \mathit{Ref}$. In this case (line 24), we obtain that $\mathrm{post}^*_M(\mathit{Init}) \subseteq X$ and this allows us to conclude that the system is safe. Otherwise, $\mathit{Ref} \subsetneq X$ is used to refine μ_i to μ_{i+1} that is obtained by closing $\mu_i \cup \{\mathit{Ref}\}$ both under intersections — in order to have an abstraction — and unions — in order to have a disjunctive abstraction.

Let us now illustrate how FBAR works on a simple finite example.

Example 6.1. Let us consider the safety verification problem represented by the following diagram, where $\mu_0 = \mathrm{Cl}_\cup(\{1, 12, 3, 35, 3456, 6, 7\})$.



Then, FBAR allows to derive that the system is unsafe with no refinement. In fact, FBAR gives rise to the following “execution trace”:

[line 2]: $M = 1234567;\ U = 123456;\ V = 34567;\ X = 123456;\ Y = 7;$
[line 5]: $U' = 123456;$
[line 6]: $1234567 = \mu_0(M \cap (\mathit{Init} \cup \mathrm{post}(U'))) \not\subseteq U = 123456;$
[line 7]: $M = 1234567;$
[line 8]: $V' = 3567;$
[line 9]: $13567 = \mu_0(M \cap (\mathit{Bad} \cup \mathrm{pre}(V'))) \not\subseteq V = 34567;$
[line 10]: $M = 13567;$
[line 11]: $U' = U \ \&\ V' \neq V;$
[line 12]: $U = 123456;\ V = 3567;$
[line 5]: $U' = 135;$
[line 6]: $1357 = \mu_0(M \cap (\mathit{Init} \cup \mathrm{post}(U'))) \not\subseteq U = 123456;$
[line 7]: $M = 1357;$
[line 8]: $V' = 357;$
[line 9]: $1357 = \mu_0(M \cap (\mathit{Bad} \cup \mathrm{pre}(V'))) \not\subseteq V = 3567;$
[line 10]: $M = 1357;$
[line 11]: $U' \neq U \ \&\ V' \neq V;$
[line 12]: $U = 135;\ V = 357;$
[lines 5-10]: a further iteration that does not change U', V' and M
[line 11]: $U' = U \ \&\ V' = V;$
[line 13]: $X = 135;\ Y = 7;$
[line 15]: $X = 135;$
[line 16]: $X' = \varnothing;$
[line 16]: $\mathit{Init} \cap M \not\subseteq X' \quad \Rightarrow$ KO

Thus, FBAR needs no abstraction refinement and seven abstract fixpoint computations. On the other hand, CGR needs three abstraction refinements and eight abstract fixpoint computations in order to conclude that the system is unsafe. In fact, it computes the following sequence: $Z_0 = 123456$, $R_0 = 123456$, $S_0 = 123456$, $Z_1 = 12346$, $R_1 = 12346$, $S_1 = 12346$, $Z_2 = 124$, $R_2 = 124$, $S_2 = 124$, $Z_3 = 24$, $R_3 = 24$, $S_3 = 24$ and then concludes KO.

It can be also checked that the dual algorithm $\mathrm{CGR}^{\leftarrow}$ needs one abstraction refinement in order to conclude that the system is unsafe while in this case $\mathrm{CGR}^{\leftarrow}$ performs just four abstract fixpoint computations. □

The above described properties of the FBAR procedure are stated precisely as follows.

Lemma 6.2. *The following properties are invariant in the algorithm* FBAR *in Fig. 3:*

(1) $\text{post}^*_{M \cap Safe}(Init) \subseteq U' \subseteq U \subseteq M$ & $\text{pre}^*_{M \cap NInit}(Bad) \subseteq V' \subseteq V \subseteq M$.
(2) states(MinCex) $\subseteq M$.
(3) $M \cap \widetilde{\text{pre}}^*(Safe) \subseteq X' \subseteq X \subseteq M$ & $Y \subseteq Y' \subseteq M \cap \text{pre}^*(Bad)$.

These invariant properties allows us to show that FBAR is a correct algorithm for safety checking.

Theorem 6.3 (Correctness). *If* FBAR *outputs* OK/KO *then $\mathcal{T}$ is safe/unsafe.*

6.1 Termination

Termination of FBAR is similar to that of CGR.

Theorem 6.4 (Termination)

(1) *If μ_0 is finite and there exists $\mathcal{X} \subseteq \wp(\Sigma)$ such that for all $i \geq 0$, $X_i \in \mathcal{X}$ and $\langle \mathcal{X}, \subseteq \rangle$ satisfies the descending chain condition then* FBAR *terminates.*
(2) *If $\mathcal{T}$ is unsafe then* FBAR *terminates.*

Hence, if the refiner sets X_i's at line 23 all belong to a subset of the state space that satisfies the descending chain condition then FBAR terminates. This obviously implies termination when the state space Σ is finite. Ganty et al. [13,14] show that this descending chain condition allows to show that the instantiation of the CGR algorithm to the class of well-structured transition systems (WSTSs) always terminates. This is an important result because WSTSs are a broad and relevant class of infinite-state transition systems that include, among others, Petri Nets, broadcast protocols and lossy-channel systems [11]. Since this termination condition works for FBAR exactly in the same way as for CGR, we conjecture that the descending chain condition should allow to show that FBAR terminates on WSTSs.

6.2 Relationship with CGR

We made a formal comparison between the FBAR and CGR algorithms and showed that when no abstraction refinement is needed, FBAR is better than CGR, i.e., if CGR terminates with a given abstraction μ with no abstraction refinement then this also happens for FBAR. As shown by the examples in Section 3, the converse is not true.

Theorem 6.5. *If* CGR *for some disjunctive abstract domain μ outputs* OK/KO *with no abstraction refinement then* FBAR *for μ outputs* OK/KO *with no abstraction refinement.*

We did not succeed in comparing formally FBAR and CGR when abstraction refinements indeed happen. This does not appear to be an easy task mainly because FBAR and CGR use both different refiners — FBAR refines using $\widetilde{\text{pre}}_M$ while CGR uses $\widetilde{\text{pre}}$ — and different ways of refining the abstraction — FBAR needs a refined disjunctive abstraction while CGR needs a mere abstraction. We can only report that we were not able to find an example where CGR terminates while FBAR does not.

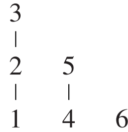
Let us also remark that Cousot-Ganty-Raskin [10] described how some acceleration techniques that compute underapproximations of the reflexive-transitive closure R^* of

the transition relation R can be integrated into CGR. The correctness of this technique basically depends on the fact that replacing the transition relation R with a relation T such that $R \subseteq T \subseteq R^*$ is still correct in CGR. This also holds for FBAR so that these same techniques can be applied.

6.3 Implementation of Disjunctive Abstractions

An implementation of FBAR is subject for future work. However, let us mention that disjunctive abstractions can indeed be efficiently implemented through a state partition and a relation defined over it.

Let us recall from [20] the details of such a representation. Let $\mu \in \mathrm{dAbs}(\wp(\Sigma))$. The partition $\mathrm{par}(\mu) \in \mathrm{Part}(\Sigma)$ induced by μ is defined by the following equivalence relation $\sim_\mu \subseteq \Sigma \times \Sigma$: $x \sim_\mu y$ iff $\mu(\{x\}) = \mu(\{y\})$. Moreover, let us define the following relation $\trianglelefteq_\mu$ on $\mathrm{par}(\mu)$: $\forall B_1, B_2 \in \mathrm{par}(\mu)$, $B_1 \trianglelefteq_\mu B_2$ iff $\mu(B_1) \subseteq \mu(B_2)$. It turns out that $\langle \mathrm{par}(\mu), \trianglelefteq_\mu \rangle$ is a poset. For example, consider the disjunctive abstraction μ depicted in Section 2, where $\mu = \mathrm{Cl}_\cup(\{1, 12, 123, 4, 45, 6\})$. The poset $\langle \mathrm{par}(\mu), \trianglelefteq_\mu \rangle$ is then as follows:



This allows us to represent the abstraction μ as follows: for any $S \subseteq \Sigma$, $\mu(S) = \cup\{B \in \mathrm{par}(\mu) \mid \exists C \in \mathrm{par}(\mu).\, C \cap S \neq \varnothing \;\&\; B \trianglelefteq_\mu C\}$.

As shown in [20], it turns out that this partition/relation-based representation provides an efficient way for representing and maintaining disjunctive abstractions. Moreover, [20] also shows that the abstraction refinement step $\mu_{i+1} = \mathrm{Cl}_{\cap,\cup}(\mu_i \cup \{X\})$ at line 25 can be efficiently implemented by a procedure that is based on partition splitting and runs in $O(|\mathrm{par}(\mu_i)|^2 + |X|)$-time.

7 Future Work

A number of tasks are left for future research. Firstly, it would be interesting to complete the formal comparison between FBAR and CGR by investigating whether and how their refinements and final outputs can be related in the general case when the abstraction is refined. We also left open our conjecture that, analogously to CGR, the FBAR algorithm terminates when applied to well-structured transition systems. After the completion of such a comparison with CGR, it would be worth to develop a prototype of FBAR that can reuse the implementation of disjunctive abstraction refinements already available from [20].

Acknowledgments. This manuscript has been completed after the death of Olivia Rossi Doria on June 13th, 2006. Francesco Ranzato and Francesco Tapparo take this opportunity to acknowledge Olivia's contribution to this paper as well as her personal friendship and professional skills.

References

1. Ball, T., Rajamani, S.K.: The SLAM Project: Debugging system software via static analysis. In: Proc. 29th ACM POPL, pp. 1–3 (2002)
2. Chaki, S., et al.: Modular verification of software components in C. IEEE Transactions on Software Engineering 30(6), 388–402 (2004)
3. Clarke, E.M., Grumberg, O., Peled, D.A.: Model checking. The MIT Press, Cambridge (1999)
4. Clarke, E., et al.: Progress on the State Explosion Problem in Model Checking. In: Wilhelm, R. (ed.) Dagstuhl Seminar 2000. LNCS, vol. 2000, pp. 176–194. Springer, Heidelberg (2001)
5. Clarke, E.M., et al.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003)
6. Cousot, P.: Mèthodes itèratives de construction et d'approximation de points fixes d'opèrateurs monotones sur un treillis, analyse sèmantique de programmes. PhD Thesis, Univ. de Grenoble, France (1978)
7. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Proc. 6th ACM POPL, pp. 269–282 (1979)
8. Cousot, P., Cousot, R.: Abstract interpretation and application to logic programs. J. Logic Programming 13(2-3), 103–179 (1992)
9. Cousot, P., Cousot, R.: Refining model checking by abstract interpretation. Automated Software Engineering 6(1), 69–95 (1999)
10. Cousot, P., Ganty, P., Raskin, J.-F.: Fixpoint-Guided Abstraction Refinements. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 333–348. Springer, Heidelberg (2007)
11. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theoretical Computer Science 256(1-2), 63–92 (2001)
12. Ganty, P.: The Fixpoint Checking Problem: An Abstraction Refinement Perspective. PhD Thesis, Université Libre de Bruxelles (2007)
13. Ganty, P., Raskin, J.F., Van Begin, L.: A Complete Abstract Interpretation Framework for Coverability Properties of WSTS. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 49–64. Springer, Heidelberg (2005)
14. Ganty, P., Raskin, J.F., Van Begin, L.: From Many Places to Few: Automatic Abstraction Refinement for Petri Nets. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 124–143. Springer, Heidelberg (2007)
15. Gulavani, B.S., Rajamani, S.K.: Counterexample Driven Refinement for Abstract Interpretation. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006 and ETAPS 2006. LNCS, vol. 3920, pp. 474–488. Springer, Heidelberg (2006)
16. Henzinger, T.A., et al.: Lazy abstraction. In: Proc. 29th ACM POPL, pp. 58–70 (2002)
17. Henzinger, T.A., et al.: Software Verification with BLAST. In: Ball, T., Rajamani, S.K. (eds.) SPIN 2003. LNCS, vol. 2648, pp. 235–239. Springer, Heidelberg (2003)
18. Manevich, R., et al.: Abstract Counterexample-Based Refinement for Powerset Domains. In: Reps, T., Sagiv, M., Bauer, J. (eds.) Wilhelm Festschrift. LNCS, vol. 4444, pp. 273–292. Springer, Heidelberg (2007)
19. Massé, D.: Combining Forward and Backward Analyses of Temporal Properties. In: Danvy, O., Filinski, A. (eds.) PADO 2001. LNCS, vol. 2053, Springer, Heidelberg (2001)
20. Ranzato, F., Tapparo, F.: An Abstract Interpretation-Based Refinement Algorithm for Strong Preservation. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 140–156. Springer, Heidelberg (2005)