Using Counterexamples for Improving the Precision of Reachability Computation with Polyhedra [⋆]

Chao Wang[1], Zijiang Yang[2], Aarti Gupta[1], and Franjo Ivančić[1]

[1] NEC Laboratories America, Princeton, NJ 08540, U.S.A.
[2] Western Michigan University, Kalamazoo, MI 49008, U.S.A.

Abstract. We present an *extrapolation with care set* operator to accelerate termination of reachability computation with polyhedra. At the same time, a counterexample guided refinement algorithm is used to iteratively expand the care set to improve the precision of the reachability computation. We also introduce two heuristic algorithms called *interpolate* and *restrict* to minimize the polyhedral representations without reducing the accuracy. We present some promising experimental results from a preliminary implementation of these techniques.

1 Introduction

Static analysis based on abstract interpretation [9] and model checking [7, 27] are popular techniques for program verification. They both rely on fixpoint computation, with the former heavily employing widening [11] to ensure termination. The precision of a widening operator is crucial for the effectiveness of abstract interpretation. Often a widening operator is carefully designed by the user *a priori* for an abstract domain, and if it does not provide enough precision, the user either accepts the result as inconclusive or has to redesign the operator. In this paper, we use counterexample guided refinement developed in model checking to automatically improve the precision of reachability computation using the polyhedral abstract domain.

Widening for convex polyhedra was introduced in [11] for numerical relational analysis and later extended to verification of integer-valued programs [15] and linear hybrid systems [16]. The operator was generalized in [5] and in [2] to powersets (or finite unions) of convex polyhedra. Approximation techniques were also studied in [17], where an *extrapolation* operator is introduced. The difference between widening and extrapolation is that the latter does not guarantee termination. The widening precision can be increased by partitioning methods [20]. In [1], a widening algorithm was introduced by combining several known heuristics and using convex widening as a last resort. In all these previous works, there is no automatic refinement involved.

In model checking, counterexample guided refinement [21, 6, 3] has been used together with predicate abstraction [13] to verify software programs. Predicate abstraction relies on finite sets of predicates to define abstract domains, and therefore can be

[⋆] This is a newer version containing a correction to Algorithm 1. The original version of this paper appeared in the Proceedings of CAV'07, in which Algorithm 1 was not presented correctly. The problem was found and fixed before the conference presentation. The correct version was presented in CAV'07.

viewed as an instance of domain refinement in abstract interpretation. However, finite abstractions in general are not as powerful as an infinite abstract domains with widening for Turing equivalent programming languages [10]. Although our new procedure uses a backward counterexample analysis similar to those in [3], our goal is to refine the care set in the same abstract domain instead of creating a new abstract model (or a new abstract domain).

In a recent work [14], Gulavani and Rajamani also proposed a counterexample driven refinement method for abstract interpretation, which identifies the fixpoint steps at which precision loss happens due to widening in forward fixpoint computation, and then use the least upper bound (convex hull for convex polyhedra) instead of widening at those steps. In effect, their refinement procedure simply skips the widening at particular steps (the least upper bound of two consecutive sets P and Q of a fixpoint computation is actually Q, since $P \sqsubseteq Q$). Our refinement procedure does not merely skip the over-approximation; instead, it produces a refined care set to guide the *direction-of-growth* in over-approximation at the next iteration of the refinement loop.

We define a new operator called *extrapolation with a care set*. Given two sets $P \sqsubseteq Q$ and a *care set* C such that $Q \cap C = \emptyset$, the extrapolation of P with respect to Q under C is a set S such that $Q \sqsubseteq S$ and $S \cap C = \emptyset$. In reachability computation, the care set C is initially empty—in this case the new operator can be substituted by normal widening whose result $S = P \nabla Q$ satisfies both $Q \sqsubseteq S$ and $S \cap C = \emptyset$. If a given invariant property ψ holds in the over-approximated reachable set, then the property is proved. Otherwise, we intersect this over-approximated set with $\neg\psi$, pick a subset, and start a precise backward analysis in order to build a counterexample. If a counterexample can be found, then we report it as a real error; otherwise, it remains to be decided. In the latter case, we analyze the spurious counterexample and produce a new care set C. The expanded care set C is used with extrapolation to compute a new reachability fixpoint. This iterative refinement process continues until either enough precision is achieved to derive a conclusive result, or the computing resources are exhausted. Note that the entire procedure is automatic, whereas for the existing widening techniques, typically the user has to redesign the widening operator manually when a false bug is reported.

We propose a set of algorithms for implementing the new operator in the domain of convex polyhedra. For two powersets P and Q of convex polyhedra, we apply the proposed operator to individual pairs $P_i \in P$ and $Q_i \in Q$ only when $P_i \sqsubseteq Q_i$. In practice, the use of a care set can significantly increase the precision of program analysis in the polyhedral powerset domain. We also introduce two new operators called *interpolate* and *restrict* to heuristically simplify the polyhedral representations. Applying these two operators during forward and backward reachability fixpoint computations does not cause a loss in precision.

There is an analogy between our widening criterion and the over-approximation in interpolant-based model checking [24]. That is, both are goal-directed and may over-approximate the reachable states by adding any state that cannot reach an error in a given number of steps, or along a given path. Our method can benefit from other recent improvements in widening-based approaches, such as lookahead widening [12]. Improved ways of selecting extrapolation points can benefit us also. Overall, we believe that our goal-directed approach for improving precision is complementary to these approaches based mostly on program structure.

2 Preliminaries

2.1 Abstract Interpretation

Within the general framework of abstract interpretation [9], the abstract postcondition and precondition operations, as well as the least upper bound, may all induce approximations. Widening is used to enforce and/or to accelerate the termination of fixpoint computations since in general the computation may not have a fixpoint or have one that cannot be reached in a finite number of iterations.

Widening (cf. [9]). A widening operator on a partial order set $(L, \sqsubseteq)$ is a partial function $\nabla : L \times L \to L$ such that

1. for each $x, y \in L$ such that $x \nabla y$ is defined, $x \sqsubseteq x \nabla y$ and $y \sqsubseteq x \nabla y$;
2. for all ascending chains $y_0 \sqsubseteq y_1 \sqsubseteq$ ..., the ascending chain defined by $x_0 := y_0$ and $x_{i+1} := x_i \nabla y_{i+1}$ for $i \geq 0$ is not strictly increasing.

An operator satisfying the first condition but not the *strictly increasing* requirement of the second condition is called an *extrapolation* [17]. In the sequel, we use ∇ to denote both widening and extrapolation when the context is clear. Since there is more freedom in choosing an actual implementation of an extrapolation operator than widening, it is possible for extrapolation to produce a tighter upper-bound set than widening.

For program verification, we consider a powerset domain of convex polyhedra over a linear target program, where only ∇ causes the precision loss (i.e., precondition, postcondition, and least upper bound are precise). We want to compute the reachability fixpoint $F = \mu Z \,.\, I \cup post(Z)$, where I is the initial predicate and $post(Z)$ is the postcondition of Z with respect to a set of transfer functions. In general, Z is a finite union of convex polyhedra. We define ψ as a predicate that is expected to hold in the program (i.e., the property of interest), then program verification amounts to checking whether $F \sqsubseteq \psi$. To apply widening/extrapolation in the reachability computation, let $y_{i+1} = x_i \cup post(x_i)$; that is,

$$\begin{array}{ll}
y_0 = I & x_0 = I \\
y_1 = I \;\; \cup post(I) & x_1 = I \;\; \nabla y_1 \\
y_2 = x_1 \cup post(x_1) & x_2 = x_1 \nabla y_2 \\
y_3 = \ldots &
\end{array}$$

Reachability computation in the concrete domain, as is often used in symbolic model checking [23], can be viewed as a special case (by making $x_i = y_i$ for all $i \geq 0$).

2.2 Polyhedral Abstract Domain

The polyhedral abstract domain was first introduced in [11] to capture numerical relations over integers. Let $\mathbb{Z}$ be the set of integer numbers and $\mathbb{Z}^n$ be the set of all n-tuples. A linear inequality constraint is denoted by $\mathbf{a}^T \cdot \mathbf{x} \leq b$, where $\mathbf{x}, \mathbf{a} \in \mathbb{Z}^n$ are n-tuples ($\mathbf{x}$

is the variable) and $b \in \mathbb{Z}$ is a scalar constant. A polyhedron P is a subset of $\mathbb{Z}^n$ defined by a finite conjunction of linear inequality constraints, $P = \{\mathbf{x} \in \mathbb{Z}^n \mid \forall i : \mathbf{a}_i^T \cdot \mathbf{x} \leq b_i\}$. We choose to use this *constraint system representation* in order to be consistent with our actual implementation, which is based on the Omega library [25]. An alternative would be to define P as a generator system comprising a finite set of vertices, rays, and lines. Some implementations (e.g., [22]) choose to maintain both to take advantages of their complementing strengths and to avoid the conversion overhead between the two.

The first widening operator for this abstract domain was introduced in [11], often being termed as *standard widening* (we follow this convention for ease of reference).

Standard Widening. Let P and Q be two polyhedra such that $P \sqsubseteq Q$; the widening of P with respect to Q, denoted by $P \nabla Q$, is computed as follows: when P is empty, return Q; otherwise, remove from P all inequalities not satisfied by Q and return.
The intuition behind standard widening is to predict the *directions of growth* from P to Q and then drop any constraint of P in these directions. The finiteness of the first polyhedron (where widening starts) ensures termination.

Widening Up-to. Let P and Q be two polyhedra such that $P \sqsubseteq Q$, and let M be a finite set of linear constraints. The widening up-to operator, denoted by $P \nabla_M Q$, is the conjunction of the standard widening $P \nabla Q$ with all the constraints in M that are satisfied by both P and Q.

The *widening up-to* operator was introduced in [15, 16] to improve standard widening whenever the result is known to lie in a known subset. This subset, or *up-to set* M, is defined as a set of constraints associated with each control state of a program. For instance, if a variable x is declared to be of subrange type 1..10, then $x \geq 1$ and $x \leq 10$ are added into M. If there exists a loop `for (x=0; x<5; x++)`, then the constraint $x < 5$ is also added into M. It is worth pointing out that the up-to set in [15, 16] is fixed. It does not consider automatic refinement adaptive to the property under verification.

3 Extrapolation with a Care Set

We define the *care set* to be an area within which no extrapolation result should reside in order to avoid false bugs. We use a precise counterexample guided analysis to gradually expand the care set and therefore improve the precision of the extrapolation with care set operator (defined below).

Definition 1. *An extrapolation with a care set C on a partial order set $(L, \sqsubseteq)$ is a partial function $\overset{\neg C}{\nabla} : L \times L \to L$ such that*

1. *for each $x, y \in L$ such that $x \overset{\neg C}{\nabla} y$ is defined, $x \sqsubseteq x \overset{\neg C}{\nabla} y$ and $y \sqsubseteq x \overset{\neg C}{\nabla} y$;*
2. *for all ascending chains $y_0 \sqsubseteq y_1 \sqsubseteq ...$ such that $y_i \cap C = \emptyset$, the ascending chain defined by $x_0 := y_0$ and $x_{i+1} := x_i \overset{\neg C}{\nabla} y_{i+1}$ for $i \geq 0$ satisfies $x_i \cap C = \emptyset$.*

Definition 1 is generic since it is not restricted to any particular abstract domain. In this paper, we consider an implementation for the domains of convex polyhedra and their powersets. Figure 1 provides a motivating example, in which P_1 and Q_1 are two

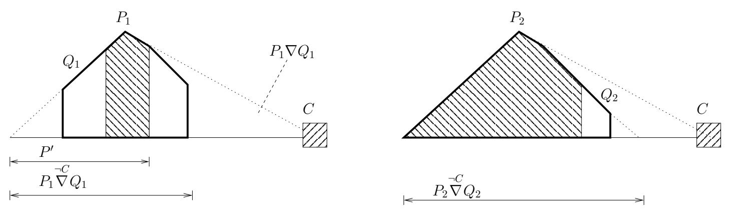

P_1
Q_1
$P_1\nabla Q_1$
C
P'
$P_1 \overset{\neg C}{\nabla} Q_1$
P_2
Q_2
C
$P_2 \overset{\neg C}{\nabla} Q_2$

Fig. 1. An example of using a care set with extrapolation

polyhedra and C is the care set. P_1 is represented by the shaded area in the middle, and Q_1 is represented by the solid thick lines. From P_1 to Q_1 there are two directions of growth. The standard widening $P_1 \nabla Q_1$ would generate the outer triangle that intersects C, thereby introducing false bugs.

We prohibit the growth to the right by using the care set C in extrapolation. By expanding only towards the left, the extrapolation result, denoted by $P_1 \overset{\neg C}{\nabla} Q_1$ on the left, and P_2 (shaded area) on the right, does not intersect C. In the next fixpoint step, we consider the extrapolation of P_2 with respect to Q_2 (solid thick lines) under the care set C. This time, the standard widening result would not intersect C. Therefore, we prefer that the result of extrapolation with a care set is the same as $P_2 \nabla Q_2$.

3.1 Using the Care Set

We now present an algorithm to compute the extrapolation with care set for convex polyhedra. In the sequel, a linear inequality constraint c is also referred to as a *half-space* since it represents a set $\{\mathbf{x} \in \mathbb{Z}^n | \mathbf{a}^T \cdot \mathbf{x} \leq b\}$. Let c be a constraint of a polyhedron P, and let c' be another constraint (may or may not be in P); we use $P^c_{c'}$ to denote the new polyhedron after replacing c with c' in P, and use P^c_{true} to denote the new polyhedron after dropping c from P.

Algorithm 1 *Let $P \sqsubset Q$ be two polyhedra, and C be a non-empty powerset such that $Q \cap C = \emptyset$. The extrapolation of P with respect to Q under C is computed as follows:*

1. *build a new polyhedron P': for each constraint c of P whose half-space does not contain Q, if $P^c_{\mathsf{true}} \cap C = \emptyset$, then drop c.*
2. *build a new polyhedron Q': drop any constraint c of Q whose half-space does not contain P', if $Q^c_{true} \cap C = \emptyset$.*

Return Q' as the result.

An example of applying this algorithm is the extrapolation of P_1 with respect to Q_1 under the care set C to generate P_2 in Figure 1. In this example, P' is the polyhedron formed by dropping the left-most constraint of P_1; then all but the left-most constraint of Q are satisfied by P', so the result Q' is the polyhedron obtained by dropping the left-most constraint of Q_1. It is clear that the result does not intersect with C. In general, the result S of Algorithm 1 satisfies $Q \sqsubseteq S \sqsubseteq P \nabla Q$.

Using this algorithm together with an iterative framework to improve the care set can guarantee that after refinement, the previous precision loss will not appear again.

However, if all the *directions of growth* (indicated by standard widening) are forbidden by the care set C, then Algorithm 1 will return Q. This may lead to postponing the widening operation forever (which may produce a non-terminating sequence). In theory, if desired, we could remedy the termination problem by switching from Algorithm 1 back to standard widening after a sufficiently large (but finite) number of fixpoint steps. Another alternative is to use the *widening up-to* operator instead of extrapolation, by accepting the fact that the refinement of care set may stop making progress. However, there is a trade-off between precision and termination, since program verification in general is undecidable in the polyhedral domain. In practice, it is often possible for the proposed technique to achieve both termination and increased precision.

This algorithm is defined for two convex polyhedra. In program verification, we intend to represent reachable state sets by finite unions of convex polyhedra. We extend the extrapolation operator to the powerset domain as follows: given two powersets P and Q, we apply Algorithm 1 only to individual convex polyhedra pairs $P_i \in P$ and $Q_i \in Q$ such that $P_i \sqsubseteq Q_i$. If $P_i \in P$ is not contained in any $Q_i \in Q$ (no matching pair), we simply use P_i. The extrapolation result is also a powerset. This is similar to the approach of Bultan *et al.* [5], except that their work does not use the care set.

3.2 Refinement for Improving the Care Set

Let $\hat{F}$ be the fixpoint of reachability computation achieved with extrapolation, and $F \sqsubseteq \hat{F}$ be the set of actual reachable states. If the invariant property ψ holds in $\hat{F}$, then ψ also holds in F. If there exists $s \in (\hat{F} \cap \neg\psi)$, it remains to be decided whether $s \in F$, or s is introduced because of extrapolation. If $s \in F$, then we can compute a concrete counterexample. Otherwise, there is a precision loss due to extrapolation.

We compute $\hat{F}$ by extrapolation with care set as follows, starting with $C = \neg\psi$.

- $\hat{F}_0 = I$;
- $\hat{F}_{i+1} = \hat{F}_i \overset{\neg C}{\nabla} (\hat{F}_i \cup post(\hat{F}_i))$, for $i \geq 0$ until fixpoint.

When the care set C is empty, the extrapolation with care set is equivalent to normal widening. If there is an index fi such that $\hat{F}_{fi} \cap C \neq \emptyset$, we stop reachability computation and start the backward counterexample analysis (using precondition computations),

- $B_{fi} = \hat{F}_{fi} \cap C$;
- $B_{i-1} = \hat{F}_{i-1} \cap pre(B_i)$ for all $i \leq fi$ and $i > 0$, if $B_i \neq \emptyset$.

If $B_0 \neq \emptyset$, we have found a concrete counterexample inside the sequence $B_0, \ldots, B_{fi}$. If $B_{i-1} = \emptyset$ for an index $i > 0$, then the set B_i is introduced by extrapolation.

Theorem 1. *If there exists an index $0 < i < fi$ such that $B_{i-1} = \emptyset$, then B_i must have been introduced into $\hat{F}_i$ by extrapolation during forward fixpoint computation.*

Proof. Since $\hat{F}_i = \hat{F}_{i-1} \overset{\neg C}{\nabla} (\hat{F}_{i-1} \cup post(\hat{F}_{i-1}))$, the set $B_i \subseteq \hat{F}_i$ is added either by postcondition or by extrapolation. Since $B_{i-1} = \emptyset$ and $B_{i-1} = \hat{F}_{i-1} \cap pre(B_i)$, B_i is not reached from $\hat{F}_{i-1}$ in one step. Therefore B_i is introduced by extrapolation. $\square$

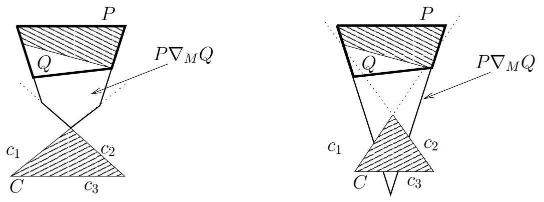


Fig. 2. Two examples of applying widening up-to operator: $M = \{\neg c_1, \neg c_2, \neg c_3\}$.

We expand the care set (initially empty) by making $C = C \cup B_i$. After that, extrapolation is allowed only if it does not introduce any erroneous state in B_i. Recall that extrapolation estimates directions of growth and then over-approximates growth in these directions. B_i provides hints about directions in which over-approximation should be prohibited. With the expanded care set C, we can re-start reachability fixpoint computation from $\hat{F}_{i-1}$ where $B_{i-1} = \emptyset$ and $B_i \neq \emptyset$. The bad states in set $\hat{F} \cap \neg\psi$ can no longer be reached through the same counterexample. This set may either become empty, or remain non-empty due to a different sequence of $\hat{F}_i$'s. In the latter case, we keep expanding the care set until one of the following happens: (1) a concrete counterexample is found and we report that the property ψ fails; (2) the set $\hat{F} \cap \neg\psi$ is empty and we report that the property ψ holds; (3) the limit of computing resources (CPU time or memory) is exceeded; in this case, the property remains undecided.

The correctness of the iterative refinement method is summarized as follows: (1) Since $\hat{F}$ remains an upper-bound of F, if a property fails in F, then it must fail in $\hat{F}$ as well. Therefore, a failing property will never be reported as true in our analysis. (2) Since we report bugs only when the precise backward analysis reaches an initial state, only failing properties can produce concrete counterexamples. Therefore, a passing property will never be reported as false in our analysis.

We have kept union exact in order to simplify the presentation of our refinement algorithm. However, only the requirement of exact postcondition/precondition is necessary. Union can be made less precise by, for instance, selectively merging convex polyhedra in a powerset, as long as the resulting powerset does not immediately intersect the care set. This precision loss can be recovered by using the same counterexample guided refinement (the argument is similar to Theorem 1).

3.3 Improving the Up-To Set

Once the care set C is computed, it can be used to derive the up-to set M for the *widening up-to* operator. In our iterative refinement framework, the extrapolation operator can be replaced by ∇_M—this is an alternative way of implementing our iterative widening refinement procedure. In [15, 16], the original ∇_M operator relies on a fixed set M of linear constraints, which are often derived statically from control conditions of the target program. Given a care set C, we can negate individual constraints of its polyhedra and add them into the *up-to set* M; that is, $M = \{\neg c_i \mid c_i \text{ is a constraint of a polyhedron in } C\}$.

In widening up-to computation $P\nabla_M Q$, the half-space of a constraint in M, or $\neg c_i$, does not have to contain P and Q. If $\neg c_i$ contains P and Q, the definition of ∇_M demands that $\neg c_i$ also contains $P\nabla_M Q$. However, if $\neg c_i$ does not contain P and Q, then $\neg c_i$ does not need to contain $P\nabla_M Q$ either. Figure 2 shows two examples for this computation, where C is the care set and M is the derived up-to set. In the left example, since both $\neg c_1$ and $\neg c_2$ (representing areas above the two lines) contain Q (hence P), the result of $P \nabla_M Q$ is the conjunction of $\neg c_1$, $\neg c_2$, and standard widening $P \nabla Q$; the constraint $\neg c_3$ (representing the area below the line) does not contain Q. In the right example, since none of the three constraints $\neg c_1, \neg c_2, \neg c_3$ contains Q, the widening result is simply $P \nabla Q$ itself, and therefore $(P \nabla_M Q) \cap C \neq \emptyset$.

In general, the *up-to set* M is weaker than the care set C in restricting the ways to perform overapproximation, so there is no guarantee that the widing up-to result does not intersect C. Therefore, it is possible that the reachability fixpoint computation (with widening up-to) after the refinement of care set generates a previously inspected spurious counterexample. As a result, although each individual forward reachability fixpoint computation always terminates, the overall iterative refinement loop may stop making progress (a tradeoff).

4 Optimizations

Counterexample analysis using precise precondition computations may become computationally expensive. In this section, we present several optimizations to make it faster.

4.1 Under-Approximating Backward Analysis

The overhead of counterexample analysis can be reduced by under-approximating the set B_i for $i \leq fi$ and $i > 0$; that is, at each backward step, we use a non-empty subset $B'_i \sqsubseteq B_i$. For instance, B'_i could be a single convex polyhedron when B_i is a finite union of polyhedra. The simplified counterexample analysis is given as follows:

1. $B_{fi} = (\hat{F}_{fi} \cap \neg\psi)$ and $B'_{fi} = \textsc{subSet}(B_{fi})$;
2. $B_{i-1} = \hat{F}_{i-1} \cap pre(B'_i)$ and $B'_{i-1} = \textsc{subSet}(B_{i-1})$ for all $i \leq fi$, if $B'_i \neq \emptyset$.

The correctness of this simplification, that the proof of Theorem 1 still holds after we replace B_i with B'_i, is due to the following two reasons. First, the precondition is precise in our powerset domain (it would not hold, for instance, in the convex polyhedral lattice where LUB may lose precision). Second, the overall iterative procedure remains correct by adding each time B'_i instead of B_i into the care set C—the difference between this simplified version and the original counterexample analysis is that the simplified one uses a more lazy approach for refinement, and it may need more than one refinement pass to achieve the same effect as using B_i. When using B'_i instead of B_i to compute the care set, we may not be able to remove the spurious set $B_i \setminus B'_i$ right away. However, if spurious counterexamples leading to $B_i \setminus B'_i$ appear again after refinement, $\textsc{subSet}(B_i \setminus B'_i)$ will be picked up to start computing the new care set.

Finally, if $B'_0 \neq \emptyset$, it guarantees that there is a real counterexample since all precondition computation results, although underapproximated, are accurate. What we miss is the guarantee to find a concrete counterexample during the earliest possible pass, because there may be cases where $B_i \neq \emptyset$ but $B'_i = \emptyset$.

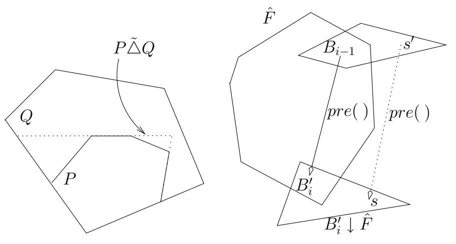
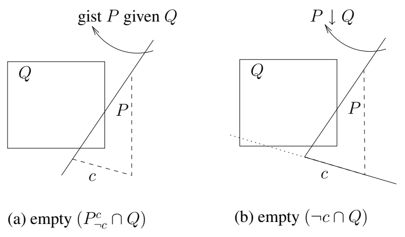


Fig. 3. (a) *interpolate* and (b) *restrict*



gist P given Q
Q
P
c
(a) empty $(P^c_{\neg c} \cap Q)$
$P \downarrow Q$
Q
P
c
(b) empty $(\neg c \cap Q)$

Fig. 4. comparing *gist* with *restrict*

4.2 Simplifications of Polyhedral Representations

We now present two heuristic algorithms for simplifying the representations of polyhedra without reducing the accuracy of fixpoint computation. These are orthogonal to the use of extrapolation with a care set.

Definition 2 (Interpolate). *Let P and Q be two sets such that $P \sqsubset Q$. The interpolate $P\tilde{\triangle}Q$ is a new set such that $P \sqsubseteq (P\tilde{\triangle}Q) \sqsubseteq Q$.*

The interpolate can be used to simplify the *from set*, i.e., the set for which we compute the postcondition during the reachability computation. Let $\hat{F}_{i-1} \sqsubset \hat{F}_i$ be two consecutive sets in this computation. We use $(\hat{F}_i \setminus \hat{F}_{i-1})\tilde{\triangle}\hat{F}_i$ instead of $\hat{F}_i$ (or the frontier $\hat{F}_i \setminus \hat{F}_{i+1}$) to compute the postcondition. In principle, any set S such that $P \sqsubseteq S \sqsubseteq Q$ can be used as the *from set* without reducing the accuracy of the reachability result. We prefer one with a simplier polyhedral representation.

Algorithm 2 *Let $P \sqsubset Q$ be two convex polyhedra. We compute a new polyhedron S by starting with $S = P$ and keep dropping its constraints c as long as $S^c_{\mathsf{true}} \sqsubset Q$. We return, between Q and S, the one with the least number of constraints.*

This heuristic algorithm tries to minimize the representation of P by inspecting every constraint greedily. Figure 3 (a) gives an example for applying this algorithm, where dropping any constraints in $P\tilde{\triangle}Q$ makes the result grow out of Q.

We note that similar ideas and algorithms exist for BDDs [4] and are routinely used in symbolic model checking [23]. Our definition of $\tilde{\triangle}$ is a generalization of these ideas for abstract domains. Also note that since the purpose here is heuristic simplification, any cheap convex over-approximation technique (instead of a tight convex-hull) can be used to first compute a convex set from $\hat{F}_i \setminus \hat{F}_{i+1}$.

Definition 3 (Restrict). *Let P and Q be two sets. The restrict $P \downarrow Q$ is defined as the new set $\{\mathbf{x} \in \mathbb{Z}^n \mid \mathbf{x} \in P \cap Q$, or $\mathbf{x} \notin Q\}$.*

Restrict computes a simplified set S for P such that (1) its intersection with Q, or $S \cap Q$, equals $P \cap Q$ and (2) S may contain an arbitrary subset of $\neg Q$. Restrict can be used to simplify the *from set* in precondition computation, with respect to a known $\hat{F}$. In counterexample analysis, when computing $B_{i-1} = \hat{F}_{i-1} \cap pre(B'_i)$, we use $pre(B'_i \downarrow \hat{F})$ instead of $pre(B'_i)$. As is shown in Figure 3 (b), adding $s \in \neg\hat{F}$ does not add any erroneous states into $\hat{F}_k$ for $0 \leq k < (i-1)$.

Algorithm 3 *Let P and Q be convex polyhedra. We define the computation of $(P \downarrow Q)$ as follows: If $P = \mathbb{Z}^n$ or if $Q = \emptyset$, return $\mathbb{Z}^n$. Otherwise, in the recursive step, choose a constraint c from P: if $\neg c \cap Q$ is empty, return $(P^c_{\mathsf{true}} \downarrow Q)$, else return $c \cap (P^c_{\mathsf{true}} \downarrow (Q \cap c))$.*

This algorithm is inspired by the *gist* operator [26], which itself is a restrict operator on polyhedra. In particular, *gist P given Q* returns a conjunction containing a minimal subset of constraints in P such that $(gist\ P\ given\ Q) \cap Q = P \cap Q$. However, *gist* in its original form is expensive and not suitable for fast heuristic simplification. In Algorithm 3, we have safely dropped the *minimal subset* requirement by checking the emptiness of $\neg c \cap Q$ (instead of $P^c_{\neg c} \cap Q$ as in [26]). This may sometimes produce a less compact representation: in Figure 4, for example, $P \downarrow Q$ (right) has two constraints while the $gist$ result has only one. However, the computation of $P \downarrow Q$ is more efficient and the result remains a generalized cofactor [29].

5 Application in Program Verification

We have implemented the proposed techniques in the F-SOFT [19, 18] platform. F-SOFT is a tool for analyzing safety properties in C programs by using both static analysis and model checking. Static analysis is used to quickly filter out properties that can be proved in a numerical abstract domain [28]. Unresolved properties are then given to the model checker. Despite the combination of different analysis engines, in practice there are still many properties that (1) cannot be proved by static analysis techniques with standard widening, and (2) symbolic model checking takes a long time to terminate because of the large sequential depth and state explosion. Our work aims at resolving these properties using extrapolation with an iteratively improved care set.

Implementation We incorporated the proposed technique into a symbolic analysis procedure built on top of CUDD [30] and the Omega library [25], as described in [32]. It begins with a C program and applies a series of source-level transformations [18], until the program state is represented as a collection of simple scalar variables and each program step is represented as a set of parallel assignments to these variables (each program step corresponds to a *basic block*). The transformations produce a control flow structure that serves as the starting point for both static analysis and model checking. We use BDDs to track the control flow logic (represented as Boolean functions) and polyhedral powersets to represent numerical constraints of the target program.

The reachable sets (e.g. $\hat{F}_i$) are decomposed and maintained as powersets at the individual program locations (basic blocks), so that each location l is associated with a subset F^l_i. Each location l is also associated with a care set C^l. Extrapolation with the care set C^l is applied only locally to F^l_i and $F^l_i \cup post^l(F_i)$, where $post^l(F_i)$ denotes the subset of postcondition of F_i that resides at the program location l.

During forward reachability computation, we apply extrapolation selectively at certain program locations: We identify back-edges in the control flow graph whose removal will break all the cycles in the control flow. Tails of back-edges serve as the *synchronization points* in the fixpoint computation. A *lock-step* style [32] fixpoint computation is conducted as follows: we use transfer functions on the forward-edges only in the fixpoint computation until it terminates; we propagate the reached state set simultaneously

Table 1. Comparing methods for computing reachability fixpoint (? means unknown)

Test Program				Analysis Result				Total CPU Time (s)			
name	loc	vars	blks	widen only	extra refine	MIX m.c.	BDD m.c.	widen only	extra refine	MIX m.c.	BDD m.c.
bakery	94	10	26	?	true	true	true	18	5	13	2
tcas-1	1652	59	133	?	true	true	true	18	34	128	433
tcas-2	1652	59	133	?	true	true	true	18	37	132	644
tcas-3	1652	59	133	?	true	true	true	18	49	135	433
tcas-4	1652	59	133	?	true	true	true	18	19	137	212
tcas-5	1652	59	133	?	false	false	false	18	80	150	174
appl-a	1836	78	307	true	true	?	?	17	22	>1800	>1800
appl-b	1836	78	307	?	false	false	?	11	94	277	>1800
appl-c	1836	78	307	?	false	false	?	13	111	80	>1800
appl-d	1836	78	307	?	false	false	?	13	68	78	>1800

through the back-edges; we then perform forward computation again. Inside this reachability computation framework, we apply extrapolation only at the tails of back-edges and only when we propagate the state sets through back-edges.

Experiments Our experiments were conducted on a set of control intensive C programs. Among the test cases, *bakery* is a C model of Leslie Lamport's bakery protocol, with a mutual exclusion property. The *tcas* examples are various versions of the Traffic alert and Collision Avoidance System [8] with properties originally specified in linear temporal logic; we model these properties by adding assertions to the source code to trap the corresponding bugs, i.e., an error exists only if an unsafe statement becomes reachable. The *appl* examples are from an embedded software application for a portable device. Most of the properties cannot be resolved directly by conventional static analysis techniques (due to the low widening precision).

Our experiments were conducted on a Linux machine with 3 GHz Pentium 4 CPU and 2GB of RAM. The results are given in Table 1, wherein we list in Columns 1-4 the name, the lines of C code, the number of variables, and the number of basic blocks. These numbers are collected after the test programs have been aggressively simplified using program slicing and constant value propagation. Columns 5-8 indicate the analysis result of four different methods: *widen-only* denotes an implementation of the standard widening algorithm, *extra-refine* denotes our new iterative refinement method with the use of care set, *MIX-mc* denotes a model checking procedure using a combination of BDDs and finte unions of polyhedra [32], and *BDD-mc* denotes a symbolic model checker using only BDDs that has been tuned specifically for sequential programs [31]. We have tried SAT-based BMC also, but our BDD-based algorithm outperforms BMC on these examples (experimental comparison in [31, 32]). Columns 9-12 compare the runtime of different methods.

Among the four methods, *widen-only* is the fastest but also the least precise in terms of the analysis result—it cannot prove any of true properties except *appl-a*. *BDD* and *MIX* are symbolic model checking algorithms, which often take a longer time to complete and are in general less scalable. In contrast, the new method *extra-refine* achieves a runtime comparable to *widen-only* and at the same time is able to prove all these true properties. For false properties, *widen-only* always reports them as "potential errors." The other three methods, *extra-refine*, *MIX*, and *BDD*, are able to produce concrete counterexamples if they complete. Due to state explosion and the large sequential depth of the software models, *BDD* does not perform well on these properties. Both *MIX* and *extra-refine* find all the counterexamples, due to their use of Integer-level representations internally. The method *extra-refine* has a run time performance comparable to (often slightly better than) that of *MIX* on these properties, although in *appl-c* it takes more time to produce a counterexample than *MIX* due to the overhead of performing multiple forward-backward refinement passes.

Unlike common programming errors such as array bound violations, most properties in the above examples are at the functional level and are harder to prove by using a general-purpose static analyzer only. Although our new method is also based on abstract interpretation, the precision of its extrapolation operator is adaptive and *problem-specific*, i.e., it adapts to the property at hand through use of counterexamples.

6 Conclusions

We have presented a new refinement method to automatically improve the precision of extrapolation in abstract interpretation by iteratively expanding a care set. We propose, for the polyhedral domain, a set of algorithms for implementing *extrapolation with a care set* and for refining the care set using counterexample guided analysis. Our preliminary experimental evaluation shows that the new extrapolation based method can retain the scalability of static analysis techniques and at the same time achieve an accuracy comparable to model checking. For future work, we plan to investigate the use of care sets in other numerical abstract domains including the octagon and interval domains.

References

[1] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. In *Symposium on Static Analysis*, pages 337–354. Springer, 2004. LNCS 2694.

[2] R. Bagnara, P. M. Hill, and E. Zaffanella. Widening operators for powerset domains. In *Verification, Model Checking, and Abstract Interpretation*, pages 135–148. Springer, 2004. LNCS 2937.

[3] T. Ball, R. Majumdar, T. Millstein, and S. K. Rajamani. Automatic predicate abstraction of C programs. In *Programming Language Design and Implementation*, 2001.

[4] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Computer*, C-35(8):677–691, 1986.

[5] T. Bultan, R. Gerber, and C. League. Verifying systems with integer constraints and boolean predicates: A composite approach. In *Symposium on Software Testing and Analysis*, pages 113–123, 1998.

[6] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, pages 154–169. Springer, 2000. LNCS 1855.

[7] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of Workshop on Logics of Programs*, pages 52–71. Springer, 1981. LNCS 131.

[8] A. Coen-Porisini, G. Denaro, C. Ghezzi, and M. Pezze. Using symbolic execution for verifying safety-critical systems. In *European Software Engineering Conference/Foundations of Software Engineering*, pages 142–151, 2001.

[9] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *International Symposium on Programming*, pages 106–126, 1976.

[10] P. Cousot and R. Cousot. Comparing the galois connection and widening/narrowing approaches to abstract interpretation. In *Programming Language Implementation and Logic Programming*, pages 269–295. Springer, 1992. LNCS 631.

[11] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Principles of Programming Languages*, pages 84–96, 1978.

[12] D. Gopan and T. W. Reps. Lookahead widening. In *Computer Aided Verification*, volume 4144 of *LNCS*, pages 452–466. Springer, 2006.

[13] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. In *Computer Aided Verification*, pages 72–83. Springer, 1997. LNCS 1254.

[14] B. S. Gulavani and S. K. Rajamani. Counterexample driven refinement for abstract interpretation. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 474–488. Springer, 2006. LNCS 3920.

[15] N. Halbwachs. Delay analysis in synchronous programs. In *Computer-Aided Verification*, pages 333–346. Springer, 1993. LNCS 697.

[16] N. Halbwachs, Y. E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in Systems Design*, 11(2):157–185, 1997.

[17] T. A. Henzinger and P.-H. Ho. A note on abstract interpretation strategies for hybrid automata. In *Hybrid Systems II*, pages 252–264. Springer, 1995. LNCS 999.

[18] F. Ivančić, I. Shlyakhter, A. Gupta, M.K. Ganai, V. Kahlon, C. Wang, and Z. Yang. Model checking C program using F-Soft. In *IEEE International Conference on Computer Design*, pages 297–308, 2005.

[19] F. Ivančić, Z. Yang, I. Shlyakhter, M.K. Ganai, A. Gupta, and P. Ashar. F-SOFT: Software verification platform. In *Computer-Aided Verification*, pages 301–306. Springer, 2005. LNCS 3576.

[20] B. Jeannet, N. Halbwachs, and P. Raymond. Dynamic partitioning in analyses of numerical properties. In *Symposium on Static Analysis*, pages 39–50. Springer, 1999. LNCS 1694.

[21] R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes*. Princeton University Press, Princeton, NJ, 1994.

[22] The Parma Polyhedra Library. *http://www.cs.unipr.it/ppl/*. University of Parma, Italy.

[23] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, MA, 1994.

[24] K. L. McMillan. Interpolation and SAT-based model checking. In *Computer Aided Verification (CAV'03)*, pages 1–13. Springer, July 2003. LNCS 2725.

[25] The Omega Project. *http://www.cs.umd.edu/projects/omega/*. University of Maryland.

[26] W. Pugh and D. Wonnacott. Going beyond integer programming with the Omega test to eliminate false data dependences. *IEEE Trans. on Parallel and Distributed Systems*, 6(2):204–211, 1994.

[27] J. P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Symposium on Programming*, 1981.

[28] S. Sankaranarayanan, F. Ivančić, I. Shlyakhter, and A. Gupta. Static analysis in disjunctive numerical domains. In *Symposium on Static Analysis*, 2006.

[29] T. R. Shiple, R. Hojati, A. L. Sangiovanni-Vincentelli, and R. K. Brayton. Heuristic minimization of BDDs using don't cares. In *Design Automation Conference*, pages 225–231, 1994.

[30] F. Somenzi. *CUDD: CU Decision Diagram Package*. University of Colorado at Boulder, ftp://vlsi.colorado.edu/pub/.

[31] C. Wang, Z. Yang, F. Ivančić, and A. Gupta. Disjunctive image computation for embedded software verification. In *Design, Automation and Test in Europe*, pages 1205–1210, 2006.

[32] Z. Yang, C. Wang, F. Ivančić, and A. Gupta. Mixed symbolic representations for model checking software programs. In *Formal Methods and Models for Codesign*, pages 17–24, 2006.