Abstract Counterexample-based Refinement for Powerset Domains

R. Manevich[1,†], J. Field[2] , T. A. Henzinger[3,§], G. Ramalingam[4,¶], and M. Sagiv[1]

[1] Tel Aviv University, {rumster,msagiv}@tau.ac.il
[2] IBM T.J. Watson Research Center, jfield@watson.ibm.com
[3] EPFL, tah@epfl.ch
[4] Microsoft Research India, grama@microsoft.com

Abstract. Counterexample-guided abstraction refinement (CEGAR) is a powerful technique to scale automatic program analysis techniques to large programs. However, so far it has been used primarily for model checking in the context of predicate abstraction. We formalize CEGAR for general powerset domains. If a spurious abstract counterexample needs to be removed through abstraction refinement, there are often several choices, such as which program location(s) to refine, which abstract domain(s) to use at different locations, and which abstract values to compute. We define several plausible preference orderings on abstraction refinements, such as refining as "late" as possible and as "coarse" as possible. We present generic algorithms for finding refinements that are optimal with respect to the different preference orderings. We also compare the different orderings with respect to desirable properties, including the property if locally optimal refinements compose to a global optimum. Finally, we point out some difficulties with CEGAR for non-powerset domains.

1 Introduction

The CEGAR (Counterexample-guided Abstraction Refinement) paradigm [1, 3] has been the subject of a significant body of work in the automatic verification community. The basic idea is as follows. First, we statically analyze a program using a given abstraction. When an error is discovered, the analyzer generates an abstract counterexample, and checks whether the error occurs in the corresponding concrete execution path. If so, the execution path is presented to the user. Otherwise, the analyzer examines the spurious abstract counterexample and refines the abstraction to remove it. The analyzer continues refining iteratively, driven by abstract counterexamples, until it either reaches a fixpoint or runs out of resources.

Our motivation for creating a general model for the abstract counterexample-based refinement problem is twofold. First, given a spurious abstract counterexample, there may be more than one abstraction refinement that eliminates it.

[†] This research is partially supported by the Clore Fellowship Programme.
[§] Supported in part by the Swiss National Science Foundation.
[¶] Work done while author was at IBM T.J. Watson Research Center.

Indeed, in some situations the set of suitable refinements is infinite. However, two refinements that remove the same spurious abstract counterexample may differ significantly in their cost to compute, or in their effect on the number of subsequent refinement steps required to reach convergence. Until now, we had no way to cleanly separate refinement strategies from the abstractions that use them; this in turn made it difficult to compare and contrast the cost and effectiveness of different CEGAR techniques, or to mix and match abstraction constructions and refinement strategies. Second, most abstraction refinement techniques have heretofore been based on predicate abstractions. We would like to extend the applicability of automatic refinement to other domains, in particular to expensive domains such as the ones used for shape analysis [12], in order to achieve scalability. Although in this paper we do not demonstrate how to instantiate our framework for abstract domains not based on predicate abstraction, we hope that the framework can be used as a first step in this direction.

In the remainder of this paper, we lay out a framework for counterexample-guided abstraction refinement for arbitrary abstract domains, place existing work in this framework, define various "preference orderings" that may be used to select among candidate abstraction refinements, and compare and contrast the consequences of various refinement strategies that use these orderings. For example, we show that for certain preference orderings, computing optimal refinements for each counterexample path iteratively does not necessarily compose to a globally optimal abstraction.

Our focus is on a single refinement step within the CEGAR method: how to refine the abstraction in order to remove one abstract counterexample. We present a theoretical framework for refining abstractions in order to remove spurious abstract counterexamples. Our framework is based on the theory of abstract interpretation [4], making it possible to extend the problem beyond predicate abstraction. Inspired by the concept of "parsimonious abstractions" [7], we allow different abstractions to be associated with different control flow locations. Furthermore, we consider a parametric variant of refinement. In this setting, rather than computing refinements over the space of all abstractions of a concrete domain, we assume we are given as a parameter a lattice of predefined abstractions, e.g., because its values are computationally cheap to manipulate.

During the investigation of the problem, we discovered that the refinement of powerset domains[5] is simpler than refinement of non-powerset domains. In the latter case, refinement may need to be done for a set of control flow paths simultaneously, and special underapproximation techniques may have to be devised (see Appendix A for further details). We therefore focus on refinement of powerset domains.

The main results of this paper are as follows:

Refinement Orderings. For a given spurious counterexample, there may be many possible refinements that eliminate it. We identify and focus on three di-

[5] An abstract domain is a powerset domain if it is closed under unions. For example, predicate abstractions yield relational domains.

mensions[6] in the space of refinements: (i) the abstract domain at the control flow locations along the trace, (ii) the set of control flow locations where refinement occurs, and (iii) the abstract elements that appear along the refined trace at those locations. We use these dimensions to define preference orders on the space of refinements.

Refinement Procedures. We present new refinement procedures that provide optimal refinements with respect to a given preference order and we identify, along the way, a set of abstract domain operations that can be used for refining abstract counterexamples.

Constrained Problem Settings. We modify the initial idealized setting of the problem to consider a more realistic situation where refinement of an abstract domain is done by adding abstract values from a given lattice of abstract values.

Local Optima vs. Global Optima. We consider the question of whether proving a property by refining abstract counterexamples in a locally optimal way provides globally optimal solutions (w.r.t. the orderings we define). We show that the answer depends on the given ordering, and that for two of the orderings the answer is negative, and for one it is positive.

Running Example. Consider the simple program shown in Table 1. The goal is to prove the assertion at label 5, which is true, since the variable `x` is assigned the values $0, 1, 2, 3$ at labels $2, 3, 4, 5$, respectively. The initial abstract domain A has the following overapproximations available for these values at the corresponding program labels: `2:`$\{x < 7\}$, `3:`$\{x = 1\}$, `4:`$\{x = 0\}$, and `5:`$\{x = 2\}$. Using a sound (overapproximate) abstract semantics, the analysis determines that `x` can have any value at labels 4 and 5 (since the abstract values $\{x = 0\}$ and $\{x = 2\}$ cannot be used to approximate the values of `x` at labels 4 and 5), which is insufficient to prove the assertion.

We can eliminate the abstract counterexample by adding new (more precise) abstract values to the abstraction at the corresponding program labels, resulting in the abstract domain U. One such refinement adds the following abstract values at the corresponding labels: `2:`$\{x = 0\}$, `4:`$\{x = 2\}$, and `5:`$\{x = 3\}$. The refined abstract semantics produces the sequence of values `2:`$\{x = 0\}$, `3:`$\{x = 1\}$, `4:`$\{x = 2\}$, and `5:`$\{x = 3\}$, which proves the assertion, since $\{x = 3\} \subseteq \{x < 10\}$. Another refinement that eliminates the abstract counterexample adds the following values to the abstractions at the corresponding labels: `3:`$\{x < 8\}$, `4:`$\{x < 9\}$, and `5:`$\{x < 10\}$. We denote the refined abstract domain by W. The refined abstract semantics produces the sequence of values `2:`$\{x < 7\}$, `3:`$\{x < 8\}$, `4:`$\{x < 9\}$, and `5:`$\{x < 10\}$, which proves the assertion, since $\{x < 10\} \subseteq \{x < 10\}$.

[6] In this paper, we use the term dimension loosely to talk about properties of refinements. The properties are in fact inter-related.

Notice that the two refinements discussed here refine at different sets of labels (the first refines at $\{2,4,5\}$, and the second refines at $\{3,4,5\}$). They use different abstract values to perform the refinement, and the "shape" of the abstract values is different — the first refinement uses rather simple abstract values, of the form $\{x = a\}$, where a is a constant, whereas the second refinement uses values of the form $\{x < a\}$. This shows that the same problem can have solutions with different characteristics. We will look at several characteristics of the set of possible refinements and explain how to favor some refinements over others, according to these characteristics.

Table 1. Program label and statement; Val_A:initial sequence of abstract values; Val_U:sequence of abstract values with abstract domain U; and Val_W:sequence of abstract values using abstract domain W. In every row, the values correspond to the label before the statement

label: statement	Val_A	Val_U	Val_W
1: `x=0`	$\{x \in \mathbb{N}\}$	$\{x \in \mathbb{N}\}$	$\{x \in \mathbb{N}\}$
2: `x=x+1`	$\{x < 7\}$	$\{x = 0\}$	$\{x < 7\}$
3: `x=x+1`	$\{x \in \mathbb{N}\}$	$\{x = 1\}$	$\{x < 8\}$
4: `x=x+1`	$\{x \in \mathbb{N}\}$	$\{x = 2\}$	$\{x < 9\}$
5: `assert(x<10)`	$\{x \in \mathbb{N}\}$	$\{x = 3\}$	$\{x < 10\}$

Outline. In Sec. 2, we present a theoretical framework for the abstract counterexample refinement problem, based on abstract interpretation. In Sec. 3, we characterize the space of solutions for the problem, and define axes and preference orders on the set of solutions. In Sec. 4, we present refinement procedures that produce optimal solutions for the preference orders we define. In Sec. 5, we consider a constrained variant of the problem. In Sec. 6, we investigate the relation between locally optimal solutions and globally optimal solutions for different optimality criteria. Sec. 7 discusses related work and concludes the paper.

2 Abstract Counterexample-based Refinement

Throughout the paper, we fix the abstract counterexample refinement problem to be $\langle P, C, \phi, \overline{A}, \pi \rangle$ where: P is a program, C is a concrete domain (with a fixed concrete semantics), ϕ is a property we wish to prove, $\overline{A}$ is the initial *compound* abstract domain, and $\pi \doteq l_1, \ldots, l_k$ is a sequence of program location representing the abstract counterexample with the associated abstract values $a_1, \ldots, a_k$. We now explain each element of these elements in detail.

Program. The program P is represented by a control flow graph (CFG). The vertices of the CFG are program locations $\{entry = l_1, \ldots, l_m = exit\}$. The edges of the CFG are labeled by program statements using the notation $st_{i,j}$ to denote the statement on the edge (l_i, l_j).

Concrete Domain and Concrete Semantics. Let $STATES$ be the set of all possible concrete states that may occur at any program location (the states do not include the program counter). The complete lattice C is the powerset of $STATES$, ordered by the subset relation. A concrete operational semantics assigns a forward meaning to each program statement st, $\mathsf{post}(st) : C \rightarrow C$.[7] The *entry* location is associated with an initial value *init*, which is the set of concrete states that program execution may begin with.

Property. The *exit* location is associated with a safety property ϕ—a set of concrete states—which defines the set of legal program executions. A concrete execution that ends at the *exit* location with a concrete state $\sigma \in STATES$ is considered legal when $\sigma \in \phi$.

Localized Abstractions and Abstract Semantics. Every program location l_i is associated with a powerset abstract domain in the form of a complete sublattice $A_i \subseteq C$ such that A_i is given by an upper closure operator[8], $\rho_i : C \rightarrow C$, i.e., $A_i \doteq \{\rho_i(c) \mid c \in 2^{STATES}\}$. Thus, the elements of our abstract domains are sets of concrete states. This is not a limiting assumption, since every Galois Connection is isomorphic to one such abstract domain with a corresponding upper closure operator. We use this assumption purely to simplify the presentation.

The initial abstraction for P is a *compound abstract domain* $\overline{A} \doteq \langle A_1, \ldots , A_m\rangle$. We say that an abstract domain A is more precise than B if $B \subseteq A$. We use the point-wise extension of this order to compare the precision of two compound abstract domains.

The abstract semantics of a statement $st_{i,j}$ is given by $\mathsf{post}^\sharp(st_{i,j}) \doteq \rho_j \circ \mathsf{post}(st_{i,j})$ where A_j is obtained via the upper closure operator ρ_j.

Abstract Counterexample. We define a *trace* to be a sequence of program locations $l_1, \ldots , l_k$ (with possible repetitions of locations) that form a path in the CFG. For a given abstraction $\overline{A}$ such that A_i is the abstract domain at location l_i, given by the upper-closure operator ρ_i, we define the sequence of abstract values associated with the trace by: $a_1 \doteq \rho_1(init)$, and $a_{i+1} \doteq \mathsf{post}^\sharp(st_i)(a_i)$ for every $i = 1 \ldots k - 1$, where st_i is the statement between l_i and l_{i+1}. We will use this notation as a convention for other abstract domains and corresponding sequences of abstract values along a trace. The trace and associated abstract values together make an abstract counterexample if $a_k \nsubseteq \phi$.

There are two different cases. The abstract counterexample is a *real counterexample* if $(\mathsf{post}(st_k) \circ \ldots \circ \mathsf{post}(st_1))(init) \nsubseteq \phi$. Otherwise, we say that the abstract counterexample is a *spurious counterexample*.

Goal. The goal of the problem is to first differentiate between real counterexamples and spurious counterexamples. Second, in the case of a spurious counterexample, a solution is a compound abstract domain $\overline{A'} \doteq \langle A'_1, \ldots , A'_m\rangle$,

[7] Notice that the statements are interpreted directly over sets of concrete states, since we are using the collecting semantics [11].

[8] $\rho : C \rightarrow C$ is an upper closure operator if it is monotone ($x \subseteq y$ implies $\rho(x) \subseteq \rho(y)$), extensive ($x \subseteq \rho(x)$), and idempotent ($\rho(x) = \rho(\rho(x))$). In particular, this means that for every $x \in C$, $\rho(x)$ is the best overapproximation of x in $\{\rho(y) \mid y \in C\}$.

which we call a *a refinement*, such that: (i) $A'_i \supseteq A_i$ for every $i = 1, \ldots, k$ (i.e., A'_i is as least as precise as A_i), and (ii) $(\mathsf{post}^{\sharp'}(st_{k-1}) \circ \ldots \circ \mathsf{post}^{\sharp'}(st_1) \circ \rho'_1)(init) \subseteq \phi$, where $\mathsf{post}^{\sharp'}(st_i) \doteq \rho'_j \circ \mathsf{post}(st_i)$.

Additional Definitions and Notations. We shall refer to the operations $\mathsf{pre}(st, c)$ and $\mathsf{post}(st, c)$, which supply the semantic weakest-precondition and strongest-postcondition of the statement st and set of concrete states c, respectively. We shall also refer to the curried versions of the pre and post operators, i.e., $\mathsf{pre}(st) = \lambda\, c \in C\, .\, \mathsf{pre}(st, c)$ and $\mathsf{post}(st) = \lambda\, c \in C\, .\, \mathsf{post}(st, c)$.

For a powerset domain A, the operation $\underline{\rho}(c)$, which is not standard in abstract interpretation, supplies the best, i.e., the tightest, under-approximation for a set of concrete states c. This operation can be defined by $\underline{\rho}_A(c) \doteq \bigcup_{a \in A, a \subseteq c} a$. The resulting abstract element is in the abstract domain since the domain is closed under union.

Given a powerset domain A, the best transformer for a statement st is given by $\overline{\mathsf{post}}_A(st) \doteq \rho_A \circ \mathsf{post}(st)$. The best underapproximation of the backward meaning of a statement is given by $\underline{\mathsf{pre}}_A(st) \doteq \underline{\rho}_A \circ \mathsf{pre}(st)$.

The operation $U \sqcap_{Rel} W$ accepts two powerset domains and gives the coarsest, i.e., most abstract, powerset domain that is more precise than U and W (App. B supplies further details on this operation). For a set of concrete states $S \in STATES$, $\mathcal{D}(S) \doteq \{S, \top_C\}$, where $\top_C = STATES$, is the coarsest abstract domain containing the element S. For sets of concrete states $S_1, ..., S_b$, the notation $\mathcal{D}(S_1, ..., S_b) \doteq \sqcap_{Rel}(\mathcal{D}(S_1), \ldots, \mathcal{D}(S_k))$ denotes the coarsest abstract domain containing the sets. The notation $\mathcal{D}_\top$ stands for the abstract domain $\{\top_C\}$.

Limitations of the Model. There are certain features in static analyses that are not handled in this paper:

- We consider only the problem of refining abstract counterexamples along a fixed number of iterations over loops. This simplification does not affect the correctness of a solution, only its "quality". That is, the resulting solution eliminates a spurious counterexample but may be sub-optimal with respect to the preference orderings we define.
- We assume that all of the abstract domains are powerset domains. This assumption allows us to refine different control-flow paths independently and also to define unique optimal refinements for the orderings we define.
- The model considered above assumes that the analysis does not use widening operators, which are sometimes used by static analyses to accelerate least-fixpoint computations.
- We ignore scoping mechanisms, e.g., procedures and objects.

These limitations do not affect the applicability of the model but may affect its effectiveness in real applications.

3 Refinement Orderings

The problem defined in Sec. 2 does not lead, in general, to a unique refinement, as shown by the running example.

We denote by $Ref(\overline{A})$ the set of solutions to the given abstract counterexample refinement problem $\langle P, C, \phi, \overline{A}, \pi \rangle$, i.e., the set of compound powerset domains that refine $\overline{A}$ and remove the abstract counterexample.

Notice that a trivial refinement $\overline{A'} \doteq \langle A'_1 = C, \dots, A'_k = C \rangle$ (i.e., the concrete domain at every position) eliminates any spurious counterexample along a given path. However, this defeats the purpose of counterexample-based refinement, which is an attempt to refine the given abstraction only as much as needed to achieve the verification goal. Since there are potentially many refinements, we would like to be able to evaluate them according to some quality ordering, and favor refinements of high quality. In this section, we define interesting properties of refinements, allowing us define the orderings.

Definition 1 (Refinement Dimensions). *Let $\langle P, C, \phi, \overline{A}, \pi \rangle$ be an instance of the abstract counterexample refinement problem, and let $\overline{B} \doteq \langle B_1, \dots, B_k \rangle$ be in $Ref(\overline{A})$.*

Domains Dimension. *The coordinate of the refinement along the domain dimension is the vector $\langle B_1, \dots, B_k \rangle$.*

Values Dimension. *The forward abstract interpretation of the trace $l_1, \dots, l_k$ with $\overline{B}$ yields the sequence of abstract values $b_1 \doteq \rho_{B_1}(init)$, and $b_{i+1} \doteq \overline{\mathsf{post}}_{B_{i+1}}(st_i, b_i)$. We define $Val_B \doteq \langle b_1, \dots, b_k \rangle$.*

Indices Dimension. *We define $Ind_B \doteq \{i \mid A_i \subset B_i\}$ to be the set of indices where $\overline{B}$ refines $\overline{A}$.*

In the running example, $Ind_U = \{2, 4, 5\}$ and $Ind_W = \{3, 4, 5\}$; and the refined domains and corresponding abstract values are shown in Table 2.

In the rest of this section, we define a preference ordering for each dimension and establish lower and upper bounds for each ordering.

Table 2. Label; initial abstraction $\overline{A}$ and abstract values; a refinement $\overline{U}$; abstract values Val_U; a refinement $\overline{W}$; abstract values Val_W. In every row, the abstract domains and values correspond to the label before the statement

lab.	A	Val_A	U	Val_U	W	Val_W
1	$\mathcal{D}_\top$	$\top_C$	$\mathcal{D}_\top$	$\top_C$	$\mathcal{D}_\top$	$\top_C$
2	$\mathcal{D}(\{x<7\})$	$\{x<7\}$	$\mathcal{D}(\{x<7\},\{x=0\})$	$\{x=0\}$	$\mathcal{D}(\{x<7\})$	$\{x<7\}$
3	$\mathcal{D}(\{x=1\})$	$\top_C$	$\mathcal{D}(\{x=1\})$	$\{x=1\}$	$\mathcal{D}(\{x=1\},\{x<8\})$	$\{x<8\}$
4	$\mathcal{D}(\{x=0\})$	$\top_C$	$\mathcal{D}(\{x=0\},\{x=2\})$	$\{x=2\}$	$\mathcal{D}(\{x=0\},\{x<9\})$	$\{x<9\}$
5	$\mathcal{D}(\{x=2\})$	$\top_C$	$\mathcal{D}(\{x=2\},\{x=3\})$	$\{x=3\}$	$\mathcal{D}(\{x=2\},\{x<10\})$	$\{x<10\}$

3.1 A Preference Ordering on the Domains Dimension

We consider the following preference ordering on the domains dimension.

Definition 2 (Domain Coarseness Ordering). *For two compound abstract domains $\overline{U}$ and $\overline{W}$, we write $\overline{U} \preccurlyeq_{dom} \overline{W}$ if $W_i \subseteq U_i$ for every $i = 1 \dots k$. That is, $\overline{W}$ is less precise than $\overline{U}$.*

In the running example, $\overline{U}$ and $\overline{W}$ are incomparable by the domain coarseness ordering.

A refinement can extend the abstract domains along the trace by adding "useless" abstract values, i.e., values that are not needed in order to eliminate the given abstract counterexample. It is possible to remove useless abstract values to obtain an *equivalent* refinement that is maximal w.r.t. the ordering $\preccurlyeq_{dom}$. We now formalize this.

Definition 3. *For two compound abstract domains $\overline{U}$ and $\overline{W}$, we write $\overline{U} \sim_{val} \overline{W}$ when $Val_U = Val_W$.*

It is straightforward to verify that $\sim_{val}$ is an equivalence relation. We now show that every equivalence class of $\sim_{val}$ contains a maximal element (w.r.t. $\preccurlyeq_{dom}$).

Definition 4. *Let $\pi \doteq l_1, \dots, l_k$ be a sequence of program locations and let $\overline{A} \doteq \langle A_1, \dots, A_k \rangle$ be the initial abstraction. Let $\overline{U} \doteq \langle U_1, \dots, U_k \rangle$ be a compound abstract domain with the associated sequence of abstract values $u_1, \dots, u_k$. We define the compound abstract domain $\widehat{\overline{U}}$ by $\widehat{U}_i \doteq A_i \sqcap_{Rel} \mathcal{D}(u_i)$, i.e., $\widehat{\overline{U}}$ minimally refines A_i with the abstract value computed at location l_i by the abstract semantics with the compound abstraction $\overline{U}$.*

The following proposition shows that for every refinement there is a unique maximal refinement with respect to $\preccurlyeq_{dom}$.

Proposition 1. *For every compound domain $\overline{U} \in Ref(\overline{A})$: (i) $\overline{U} \sim_{val} \widehat{\overline{U}}$, (ii) $\widehat{\overline{U}} \in Ref(\overline{A})$, and (iii) for every $\overline{W} \in Ref(\overline{A})$, if $\overline{W} \sim_{val} \overline{U}$ then $\overline{W} \preccurlyeq_{dom} \widehat{\overline{U}}$.*

The refinement algorithm of Ball et el. [1] works in two phases. The first phase computes a set of predicates that can be added to the abstract domain to eliminate a spurious counterexample. The second phase tries to remove redundant predicates, i.e., predicates that are not needed to eliminate the counterexample. In our framework, the second phase can be seen as an attempt to maximize with respect to the ordering $\preccurlyeq_{dom}$.

In the sequel, we consider only members of $\sim_{val}$-equivalence classes that are maximal w.r.t. $\preccurlyeq_{dom}$.

3.2 A Preference Ordering on the Values Dimension

We now define an ordering on the abstract values dimension.

Definition 5 (Value Coarseness Ordering). *For two compound abstract domains $\overline{U}, \overline{W} \in Ref(\overline{A})$, we write $\overline{U} \preccurlyeq_{val} \overline{W}$ when the corresponding sequences of abstract values $u_1, \ldots, u_k$ and $w_1, \ldots, w_k$ are such that $u_i \subseteq w_i$ for $i = 1, \ldots, l$.*

In the running example, $\overline{U} \preccurlyeq_{val} \overline{W}$.

In order to define optimal refinements for the value coarseness ordering, we use the following definition, which establishes lower and upper bounds for this ordering.

Definition 6. *[Extremal Values] Let $\pi \doteq l_1, \ldots, l_k$ be a trace. We define and fix three sequence of abstract values.*

The lower envelope *is the sequence of abstract values: $f_1 \doteq init$, and $f_{i+1} \doteq \mathsf{post}(st_i, f_i)$, for $i = 1, \ldots, k-1$.*

The sequence of abstract values computed by a backward analysis, using weakest-precondition is $b_k \doteq \phi$ and $b_j \doteq \mathsf{pre}(st_j, b_{j+1})$, for $j = 1 \ldots k-1$.

The upper envelope *is the sequence of abstract values: $h_i \doteq a_i \cap b_i$ (recall that a_i is the abstract value at location l_i computed with $\overline{A}$).*

The following lemma uses the extremal values to supply a constructive way to differentiate between real and spurious abstract counterexamples.

Lemma 1. *Let $\langle P, C, \phi, \overline{A}, \pi \rangle$ be an instance of the abstract counterexample refinement problem. Then: (i) π is spurious if and only if $f_1 = init \subseteq h_1$, and (ii) if $f_1 \subseteq h_1$ then $f_i \subseteq h_i$, for every $i = 1, \ldots, k$.*

The next lemma establishes lower and upper bounds for the value coarseness ordering.

Lemma 2. *Let $\overline{U}$ be a compound abstract domain in $Ref(\overline{A})$ with the sequence of abstract values $\{u_i\}_{i=1}^k$. Let $\{a_i\}_{i=1}^k$ be the sequence of abstract values for the initial abstraction $\overline{A}$ and let $\{f_i\}_{i=1}^k$, $\{b_i\}_{i=1}^k$, and $\{h_i\}_{i=1}^k$ be the sequences defined Def. 6. Then, the following holds for every $i = 1, \ldots, k$: $f_i \subseteq u_i \subseteq h_i$.*

The first phase of the refinement algorithm of Ball et el. [1] finds refinement predicates using strongest postconditions, i.e., it computes the abstract values of the lower envelope. In our framework, this can be seen as an attempt to minimize with respect to the ordering $\preccurlyeq_{val}$. The refinement algorithm of Henzinger et al. [8] computes the weakest-precondition to find the abstract values used for refinement. In our framework, this can be seen as an attempt to maximize with respect to the ordering $\preccurlyeq_{val}$.

3.3 A Preference Ordering on the Indices Dimension

Notice that in the running example, $\overline{U}$ and $\overline{W}$ refine at different sets of control flow locations, except for locations 4 and 5. We now ask ourselves whether there exists a minimal set of indices where refinement is necessary for every refinement. This gives a lower bound along the indices dimension. Formally, we are interested in the set

$$Ind_{min} \doteq \bigcap \{ Ind_U \mid \overline{U} \in Ref(\overline{A}) \} \ .$$

The following proposition gives such a lower bound.

Proposition 2. *Let $\langle P, C, \phi, \overline{A}, \pi\rangle$ be an instance of the abstract counterexample refinement problem such that π is spurious, and let $\{f_i\}_{i=1}^k$ and $\{h_i\}_{i=1}^k$, be the lower and upper envelope values, respectively. The set Ind_{min}, defined above, can be constructively defined as follows: $Ind_{min} = \{i \mid \rho_i(f_i) \nsubseteq h_i\}$.*

In the running example, $Ind_{min} = \{4, 5\}$.

We say that a set $I \subseteq \{1, \ldots, k\}$ is *sufficient* (to eliminate a spurious abstract counterexample) if there exists a refinement that extends the abstract domains only at the locations in I. Although the set Ind_{min} is included in the set of indices of every refinement, it is not always sufficient. In the running example, it is not enough to refine at just the control flow locations in $Ind_{min} = \{4, 5\}$.

Intuitively, we would like to favor refinements that extend the abstract domains at as few locations as possible.

We would like to find a minimal set of locations, with respect to the subset relation, where refinement is sufficient to remove a spurious counterexample. The intuition is that refining fewer abstract domains could lead to a cheaper analysis.

We define an ordering on the indices dimension. The ordering aims to minimize the set of locations where refinement occurs by choosing the locations that have the highest indices.

Definition 7 (Lazy Indices Ordering). *For two compound abstract domains $\overline{U}, \overline{W} \in Ref(\overline{A})$, we write $\overline{U} \preccurlyeq_{lazy} \overline{W}$ if $Ind_U \subseteq Ind_W$, or when Ind_U is lexicographically greater than or equal to Ind_W.*

In the running example, $\overline{W} \preccurlyeq_{lazy} \overline{U}$.

The lazy indices ordering has certain interesting properties. First, the set of indices is minimal in the sense that it is not possible to remove any location from the set and remain with a set of locations that are sufficient to remove a spurious counterexample. Second, the first index is as far as possible from the beginning of the trace. The refinement method in [8] starts refining as late as possible in order to reduce the amount of re-computation in subsequent refinement iteration steps. Some refinement techniques (e.g., [1] and [7]) refine at every location along the trace, aiming to eliminate as many spurious counterexamples as possible that share a common prefix with the given counterexample; this can be seen as a method that attempts to maximize with respect to the ordering $\preccurlyeq_{lazy}$.

From Def. 7 it immediately follows that the set of refinements that are minimal with respect to the lazy refinement ordering determine a unique set of indices. We denote this set by Ind_{lazy}. In the next section we present a procedure for finding Ind_{lazy}. The upper bound on the indices dimension, for the lazy indices ordering, is given by the set of indices $1, \ldots, k$.

3.4 Combining Preference Orderings

We combine preference orderings as follows. For an ordering $\preccurlyeq$, let $\succcurlyeq$ denote its reversed version. A combined ordering $\langle a, b\rangle$, where $a \in \{\preccurlyeq_{lazy}, \succcurlyeq_{lazy}\}$ and

$b \in \{\preccurlyeq_{val}, \succcurlyeq_{val}\}$, compares two compound abstract domains lexicographically, first by the ordering a and then by the ordering b. We also consider the combined ordering $\langle b, a\rangle$.

4 Refinement Procedures

In this section, we provide refinement procedures that yield optimal solutions for the preference orderings defined in the previous section. We describe the procedures and state the corresponding correctness and optimality claims.

We first describe two helper procedures: Fig. 1(a) shows a procedure for finding the abstract values found by backward propagation of the property, by using weakest preconditions, and for finding the abstract values of the upper envelope, according to Lem. 2; and Fig. 1(b) is used to detect real counterexamples before attempting to apply any refinement procedure. If $init \not\subseteq b_1$, then executing the concrete semantics with any value in $init \setminus b_1$ yields a concrete counterexample.

Proposition 3. *Given an instance of the abstract counterexample refinement problem $\langle P, C, \phi, \overline{A}, \pi\rangle$, the procedure in Fig. 1(b) detects whether π is a real counterexample or a spurious counterexample.*

```
UpperEnvelope()
$b_k := \phi$
for $i = k-1$ to 1
  $b_i := \mathsf{pre}(st_i, b_{i+1})$
for $i = k-1$ to 1
  $h_i := b_i \cap a_i$
```

(a)

```
DetectRealError()
call UpperEnvelope()
if $init \not\subseteq h_1$ then
  ReportError('Found a real error!')
```

(b)

Fig. 1. (a) A procedure that computes the upper envelope values, according to Lem. 2; (b) A procedure for detecting real counterexamples

4.1 Refining with the Most Concrete Values/Most Abstract Values

The refinement procedures for the preference orderings $\{\preccurlyeq_{val}, \succcurlyeq_{val}\}$ are shown in Fig. 2. These procedures use Lem. 2 in order to choose the values with which to refine. Applying **RefineMostConcrete** to the running example yields the refinement $\overline{U}$, and applying **RefineMostAbstract** yields the refinement $\overline{W}$.

Theorem 1. *Given an instance of the abstract counterexample refinement problem $\langle P, C, \phi, \overline{A}, \pi\rangle$ where π is a spurious counterexample, the procedures shown in Fig. 2 output the optimal refinements for the orderings $\langle \succcurlyeq_{val}, b\rangle$ and $\langle \preccurlyeq_{val}, b\rangle$, respectively. That is, they result with refinements that have the most concrete* *values (Fig. 2(a)) or the most abstract values (Fig. 2(b)), regardless of the ordering on the indices, b. (This is because the optimums are unique and thus exactly determine the locations where refinement occurs.)*

```
RefineMostConcrete()
1: $st_0$ := skip
2: $R_0 := init$
3: for $i := 0$ to $k$
4:    $R_i := \mathsf{post}(st_{i-1}, R_{i-1})$
5:    if $R_i \not\in A_i$ then
6:       $A'_i := A_i \sqcap_{Rel} R_i$
      else
7:       $A'_i := A_i$
```

(a)

```
RefineMostAbstract()
1: $st_0$ := skip
2: $R_0 := \rho_1(init)$
3: for $i := 0$ to $k$
4:    $R_i := \overline{\mathsf{post}}_{A_i}(st_{i-1}, R_{i-1})$
5:    if $R_i \not\subseteq h_i$ then
6:       $R_i := h_i$
7:       $A'_i := A_i \sqcap_{Rel} R_i$
      else
8:       $A'_i := A_i$
```

(b)

Fig. 2. (a) A procedure that refines with the most concrete values (the lower envelope values $\{f_i\}_{i=1}^{k}$); and (b) A procedure that refines with the most abstract values (the upper envelope values $\{h_i \doteq a_i \cap b_i\}_{i=1}^{k}$)

4.2 Computing *Ind*$_{lazy}$

Theorem 2. *Given an instance of the abstract counterexample refinement problem $\langle P, C, \phi, \overline{A}, \pi\rangle$ where π is a spurious counterexample, the procedure in Fig. 3(a) outputs the sequence of indices Ind_{lazy}.*

The procedure in Fig. 3(a) uses the approximations that are available in the abstract domains when they are contained in the upper envelope. Otherwise, it does not overapproximate the values, in order to increase the opportunity of finding appropriate approximations in the domains at the subsequent locations. Applying the procedure to the running example result in locations $\{3, 4, 5\}$.

4.3 Adapting the Refinement for the Most Concrete Values/Most Abstract Values

Theorem 3. *Given an instance of the abstract counterexample refinement problem $\langle P, C, \phi, \overline{A}, \pi\rangle$ where π is a spurious counterexample, and a set of indices $I \subseteq \{1, \ldots, k\}$ that are sufficient for refinement, the procedures in Fig. 3(b) and Fig. 3(c) output the refinements with the most concrete abstract values and the most coarse abstract values, respectively.*

The advantage of having the indices as an additional parameter is that we can use the same procedures with different orderings on the indices, not just the

```
IndLazy()
$Ind_{lazy} := \{\}$
$st_0 :=$ skip
$R_0 := init$
for $i := 1$ to $k$
  $L_i := \mathsf{post}(st_{i-1}, R_{i-1})$
  $H_i := \rho_i(L_i)$
  if $H_i \subseteq h_i$ then
    $R_i := H_i$
  else
    $R_i := L_i$
    $Ind_{lazy} := Ind_{lazy} \cup \{i\}$
```

(a)

```
RefineLow(I : indices)
$st_0 :=$ skip
$R_0 := init$
for $i := 0$ to $k$
  $L_i := \mathsf{post}(st_{i-1}, R_{i-1})$
  $H_i := \rho_i(L_i)$
  if $i \in I$ then
    $R_i := H_i$
    $A'_i := A_i \sqcap_{Rel} R_i$
  else
    $R_i := L_i$
    $A'_i := A_i$
```

(b)

```
RefineHigh(I : indices)
$st_k :=$ skip
$R_{k+1} := \phi$
for $i := k$ to 1
  $H_i := \mathsf{pre}(st_i, R_{i+1})$
  $L_i := \underline{\rho}_i(H_i)$
  if $i \in I$ then
    $R_i := H_i$
    $A'_i := A_i \sqcap_{Rel} R_i$
  else
    $R_i := L_i$
    $A'_i := A_i$
```

(c)

Fig. 3. (a) A procedure for finding *Ind*$_{lazy}$; (b) A procedure for refining at a specified set of control flow locations with the most precise abstract values; and (c) A procedure for refining at a specified set of control flow locations with the coarsest abstract values

lazy indices ordering. Applying **RefineHigh** to the running example with the indices $\{3, 4, 5\}$ yields the refinement $\overline{W}$.

We combine the procedures to produce optimal refinements for a combined ordering $\langle a, b\rangle$. The procedure is shown in Fig. 4.

```
RefineIndVal($\langle a, b\rangle$ : ordering)
if $a$ is $\preccurlyeq_{lazy}$
  $I :=$ IndLazy()
else // $a$ is $\succcurlyeq_{lazy}$
  $I := [1, \ldots, k]$
if $b$ is $\preccurlyeq_{val}$
  RefineHigh($I$)
else // $b$ is $\succcurlyeq_{val}$
  RefineLow($I$)
```

Fig. 4. A procedure for the optimal refinement for the preference ordering $\langle a, b\rangle$

Theorem 4. *Given an instance of the abstract counterexample refinement problem $\langle P, C, \phi, \overline{A}, \pi\rangle$ where π is a spurious counterexample, and a combined ordering $\langle a, b\rangle$, the procedure in Fig. 4 outputs the optimal refinement w.r.t. $\langle a, b\rangle$.*

5 Constrained Problem Settings

In previous sections, we considered the abstraction refinement problem over the entire space of abstractions of the concrete domain. In this section, we constrain the set of possible abstractions in the following way. For a concrete domain C, we assume that an abstract domain $D \subseteq C$ is given as a parameter. Intuitively, D is rather precise but possibly very expensive for static analysis. We consider the lattice of abstractions that weaken D:

$$\mathit{Weak}(D) \doteq \{D' \mid \rho \text{ is an upper-closure operator and } D' = \rho(D)\} \ .$$

(As an example for a lattice of abstractions consider the set of abstractions obtained by choosing different subsets from a fixed set of predicates.) Essentially, the domain D establishes a lower bound on the precision of the refinement. This can be used both to guide the refinement process in the abstract values it chooses, and to limit the number of refinement iterations (by the height of the lattice D). Although it is possible to use different lattices of abstractions at different locations, we use the same lattice in every location to simplify things. Generalizing to different lattices of abstractions at different locations is straightforward.

We now rephrase the goal of the abstraction refinement problem in the modified settings. The goal of the refinement procedure is to check whether it is possible to refine the compound abstract domain $\overline{A} \doteq \langle A_1, \ldots, A_m \rangle$ into $\overline{A'} \doteq \langle A'_1, \ldots, A'_m \rangle$, where A_i and A'_i are in $\mathit{Weak}(D)$ for every $i = 1, \ldots, m$, such that $(\mathsf{post}^{\sharp'}(st_{k-1}) \circ \ldots \circ \mathsf{post}^{\sharp'}(st_1) \circ \rho'_1)(\mathit{init}) \subseteq \phi$, where $\mathsf{post}^{\sharp'}(st_i) = \rho'_j \circ \mathsf{post}(st_i)$ for every $i = 1 \ldots k-1$. If this is impossible (with $A'_i = D_i$ for $i = 1, \ldots, n$) then report that D is insufficiently precise to prove that ϕ holds on the path.

The refinement procedures from the previous section can be adapted to the new setting by replacing $\mathsf{post}(st)$ with $\mathsf{post}_D(st) \doteq \rho_D \circ \mathsf{post}(st)$, and replacing $\mathsf{pre}(st)$ with $\underline{\mathsf{pre}}_D(st) \doteq \underline{\rho}_D \circ \mathsf{pre}(st)$.

We note that in this setting, the optimality of the refinement procedures is used in a narrower sense than in the previous sections. Here, optimality is given with respect to the lattice $\mathit{Weak}(D)$ (not $\mathit{Weak}(C)$).

Cost-based Preference Orderings. It is possible to define more sophisticated preference orderings by assigning costs to abstract values. For example, the abstract values used in the running example by the refinement U are simpler than the values used by the refinement W, and thus may be assigned lower costs. A very simple technique to find cheap refinements is to first try refining with a low cost elements, and if this fails try refining with more costly elements. We plan to investigate the issue of cost-based ordering in future work.

6 Local Optima vs. Global Optima

Until now, we have focused on a single step within the CEGAR framework — refining along a given path. The analyzer starts with an initial abstraction and then iteratively applies the refinement step to individual paths in the CFG in order to prove the property. When this process converges, it yields a final abstraction, i.e., a compound abstract domain for the entire CFG, and set of abstract values (stored in each control flow location). For a given preference ordering, let us call the final abstraction (and abstract values) that result by applying an optimal refinement in each refinement step a *locally optimal solution*. We continue by defining preference orderings on entire CFGs, which allow us to define *globally optimal solutions*. We then compare the two types of solutions.

The dimensions defined in Sec. 3 do not depend on having the locations set on a path. Therefore, they extend immediately to an entire CFG. Thus, we can extend the domain coarseness ordering by point-wise comparison of the abstract domains in all CFG locations. Similarly, we can extend the values ordering by point-wise comparison of the fixpoint values computed with the two abstractions. The ordering on indices depends on a "natural" order on the locations along a path. We therefore consider an extension only for acyclic CFGs, by fixing a topological order on the locations.

We say that a refinement is *globally optimal* if it is precise enough to prove the property and optimal with respect to a given preference ordering.

```
1:  x = 3
2:  y = 3
    if (...)
3t:    x = x + 1
    else
3f:    y = y + 1
4:  assert(x == 4 || y == 4)
```

(a)

label: statement	A	Val_A
`1:  x = 0`	$\mathcal{D}_\top$	$\top_C$
`    if (...)`		
`2t:    x = x + 1`	$\mathcal{D}(\{x = 0\})$	$\{x = 0\}$
`    else`		
`2f:    x = x + 5`	$\mathcal{D}(\{x = 0\})$	$\{x = 0\}$
`3:  x = x + 1`	$\mathcal{D}(\{x < 5\})$	$\{x < 8\}$
`4:  x = x + 1`	$\mathcal{D}(\{x = 2\}, \{x = 6\})$	$\top_C$
`5:  assert(x < 10)`	$\mathcal{D}(\{x < 10\})$	$\top_C$

(b)

Fig. 5. (a) An example program for the domain coarseness order; (b) An example program for the lazy indices order; initial abstraction; corresponding abstract values

Domain Coarseness Order. We now show that for the domain coarseness ordering the globally optimal solution can be better than a locally optimal solution. This is not surprising. Iterative refinement might refine at a given location numerous times (for different spurious counterexamples) until reaching the final abstraction $\overline{U}$. On the other hand, it is possible to use Def. 4 to produce a coarser abstraction that suffices to prove the property by refining with at most one abstract value at every location. The next example shows this gap.

Example 1. Assume we are given the program fragment shown in Fig. 5(a) with an empty initial abstraction and that we wish to verify that the assertion in line 4 holds. There are two control flow paths $(1, 2, 3\text{t}, 4)$ and $(1, 2, 3\text{f}, 4)$, which we first consider separately:

– For the path $(1, 2, 3\text{t}, 4)$, we compute the weakest-precondition $p_{3\text{t}} : x = 3 \vee y = 4$ at location 3t, $p_2 : x = 3$ at location 2, and $p_1 : \mathit{true}$ at location 1.

– For the path $(1, 2, 3\text{f}, 4)$, we compute the weakest-precondition $p_{3\text{f}} : x = 4 \vee y = 3$ at location 3f, $p_2 : \mathit{true}$ at location 2, and $p_1 : \mathit{true}$ at location 1.

By refining the abstraction for each path separately, we get the abstract domain $\mathcal{D}(\{(x = a, y = b) \mid a = 3 \vee b = 4\}) \wedge \mathcal{D}(\{(x = a, y = b) \mid a = 4 \vee b = 3\})$, which includes all of the points from both of the individual abstract domains. However, if we compute the weakest-precondition over the two control flow paths together, we get the set of states $\{(x = a, y = b) \mid (a = 3 \vee b = 4) \wedge (a = 4 \vee b = 3)\} = \{(x = 3, y = 3), (x = 4, y = 4)\}$. This allows us to obtain a coarser abstraction than before, representing only two concrete states. $\square$

Lazy Indices Order. The following example shows that the globally optimal solution, with respect to the lazy indices order, can be better than a locally optimal solution. Consider the program shown in Fig. 5(b). The globally optimal solution contains the single location $\{3\}$, since it is possible to refine the domain at location 3 to $\mathcal{D}(\{x < 5\}, \{x = 1\}, \{x = 2\})$ and prove the property. This solution is obtained by modifying the procedure **IndLazy** to operate simultaneously on sets of acyclic paths.

If the analysis first refines the trace $(1, 2\text{t}, 3, 4, 5)$, the lazy indices order gives us the set of locations $\{4\}$ (e.g., it is possible to refine the trace with the abstract domain $\mathcal{D}(\{x = 2\}, \{x = 6\}, \{x < 6\})$). However, no matter how we refine at location 4 (either with the most precise value $\{x = 1\}$, or with the most abstract value $\{x < 6\}$), the trace $(1, 2\text{f}, 3, 4, 5)$ still needs to be refined at location 3 (say, with the value $\{x = 5\}$). Thus, the locations where refinement occurs are $\{3, 4\} \preccurlyeq_{lazy} \{3\}$, which shows the optimality gap.

Abstract Values Order. We now state a positive optimality result. Intuitively, this optimality result holds because the global optimum for the values order, at every location, is given by the intersection of the upper envelope values for the set of all program traces at that location.

Theorem 5. *An analysis that iteratively refines spurious counterexamples using the procedure* ***RefineMostAbstract*** *produces the globally optimal solution.*

7 Related Work and Conclusions

Gulavani and Rajamani [6] describe and implement an algorithm for refining widening operators to joins and join operators to disjunctions for non-powerset domains, which our model does not handle. However, their method does not refine powerset domains, which we consider here. Therefore, their work is orthogonal to ours.

Giacobazzi and Quintarelli [5] use a procedure that iteratively refines the abstract domain in order to achieve completeness of the abstract interpretation. They show that in the limit, a complete abstract interpretation removes all spurious counterexamples. This gives a semi-algorithm that can be applied to a given path to remove the abstract counterexample, albeit indirectly. The algorithm is guaranteed to converge for finite concrete domains. Our framework is geared towards eliminating a given counterexample directly using a different set of operations, and since our refinement procedure does not require completeness of the abstract interpretation it always terminates.

Loginov et al. [10] use inductive learning to refine abstractions for 3-valued shape analysis. Their refinement is not guided by abstract counterexamples but rather by imprecisions detected during the analysis itself.

Beyer et al. [2] integrate the TVLA system [9] into the BLAST software model checker [8] and use information from counterexamples to refine 3-valued structures in order to make shape analysis more scalable. Their technique is one application of abstract-counterexample refinement that goes beyond predicate abstraction.

In this paper, we present a theoretical framework for refinement of powerset domains in order to eliminate spurious counterexamples. We describe operations on abstract domains and generic algorithms that can be used as a starting point for applying counterexample-guided refinement in new settings, e.g., shape analysis. Using the framework, we are able to define different optimality criteria and compare locally optimal refinements with globally optimal refinements, which was not done until now.

References

1. T. Ball and S. K. Rajamani. The slam project: Debugging system software via static analysis. In *POPL '02: Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 1–3, New York, NY, USA, 2002. ACM Press.
2. D. Beyer, T. A. Henzinger, and G. Théoduloz. Lazy shape analysis. In *Proceedings of the 18th International Conference on Computer-Aided Verification (CAV)*, volume 4144 of *LNCS*, pages 532–546. Springer Verlag, 2006.
3. E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proc. Computer Aided Verification*, pages 154–169, 2000.
4. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximation of fixed points. In *Proc. Symp. on Principles of Prog. Languages*, pages 238–252, New York, NY, 1977. ACM Press.
5. R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples, and refinements in abstract model-checking. In P. Cousot, editor, *Static Analysis: 8th International Symposium, SAS 2001*, pages 356–373. Springer-Verlag GmbH, July 2001.
6. B. S. Gulavani and S. K. Rajamani. Counterexample driven refinement for abstract interpretation. In *Appeared in the 12th. International Conference on Tools and Algortihms for the Construction and Analysis of Systems, TACAS'06*, pages 474–488. Springer-Verlag, Mar 2006.
7. T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *POPL '04: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 232–244, 2004.

8. T.A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy abstraction. In *Symposium on Principles of Programming Languages*, pages 58–70, 2002.
9. T. Lev-Ami and M. Sagiv. TVLA: A framework for Kleene based static analysis. In *Proc. Static Analysis Symp.*, volume 1824 of *LNCS*, pages 280–301. Springer-Verlag, 2000.
10. A. Loginov, T. Reps, and M. Sagiv. Abstraction refinement via inductive learning. In *Proceedings of the 17th International Conference on Computer-Aided Verification (CAV)*, LNCS, pages 519–533. Springer Verlag, 2005.
11. F. Nielson, H.R. Nielson, and C. Hankin. *Principles of Program Analysis.* Springer-Verlag, 2001.
12. M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 24(3):217–298, 2002.

A The Challenges of Refining Non-Powerset Domains

Inter-Path Dependencies. We now show that in the face of non-powerset abstract domains, it is not always possible to remove spurious counterexample by refining single control flow paths independently. Consider the C program fragment shown in Fig. 6(a) and assume that the abstract domain used to verify the property in the assertion statement is the constant propagation domain, or *ICP*, shown in Fig. 6(b). The verification fails, since the abstract value propagated after the statement `x=2` is $x = 2$, the abstract value propagated after the statement `x=4` is $x = 4$, and the join of the values is $x = \top$, which is not precise enough to prove the property, even though the property holds on every concrete execution.

Now, if we consider each control flow path through the `if` statement separately, we discover that the abstract domain is precise enough to verify the property: we get $x = 2$ for the path the follows the positive branch and $x = 4$ for the path that follows the negative branch, both of which satisfy the property. The loss of imprecision is due to the join operator, which approximates the set of values $\{2, 4\}$ from the two paths with $\top$. This shows that in order to remove spurious counterexamples, the analysis needs to consider both control flow paths.

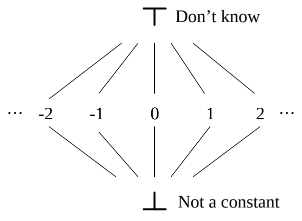
Non-Existence of Best Underapproximations. It is known that for an abstract domain, every concrete state $c \in C$ has the best overapproximation in A, given by $\rho(c)$. However, this is not generally true for underapproximation, as stated by the next lemma.

Lemma 3. *For a concrete domain C and an abstract domain A, the best underapproximation in A of every concrete element in C is ensured to exist only when A is a powerset domain.*

Proof. We first give an example showing that when A is a non-powerset domain the best underapproximation does not exist. Consider the concrete domain given by the powerset of integers and its abstraction by the ICP lattice. The set $\{2, 3\}$

```
if (...)
    x = 2
else
    x = 4
assert(x==2 || x==4)
```

(a)



(b)

Fig. 6. An example for inter-path dependency: (a) A program fragment; (b) The Constant Propagation Lattice

can be underapproximated in the ICP domain by either 2 or 3. Both of these underapproximations are tight, yet they are incomparable.

Let c be an element in C. We claim that $\underline{\rho}_A(c) \equiv \bigcup_{a\in A, a\subseteq c} a$ is the best underapproximation of c in A[9].

First, since every $x \in A$ such that $x \subseteq c$ contains only states in c, $\underline{\rho}_A(c) \subseteq c$ is an underapproximation. Furthermore, since A is a powerset domain, $\underline{\rho}_A(c) \in A$ is an underapproximation in A. Now, if $x' \in A$ is an underapproximation of c then $x \subseteq \bigcup_{x\in A, x\subseteq c} x$, and therefore $\underline{\rho}_A(c)$ is the tightest underapproximation. □

As seen in this paper, best underapproximations are very useful for abstract counterexample refinement. They are used both to determine in which control flow location to refine and how to refine (which abstract values to use).

B Abstract Domain Refinement

In this section, we discuss the operations used for refinement in the general abstract domains setting and in the setting of powerset abstract domains in particular. We use Fig. 7 as an illustrative example.

Let $C \doteq 2^{STATES}$ be a concrete domain containing all subsets of concrete states as its elements with set inclusion as an ordering relation. Given two abstract domains A and B (given by upper closure operators), the reduced product operation of A and B, denoted by $A \sqcap B$, is the simplest (i.e., most abstract) abstract domain containing $A \cup B$. In the context of this paper, A is an initial abstraction of C, and B is a simple domain containing the elements we wish to be included in the refined abstract domain. The reduced product operation is natural here, since it does the minimal amount of refinement needed to add the new elements. As an example, Fig. 7(a) and Fig. 7(b) show to abstract domain, and Fig. 7(c) shows their reduced product.

Notice that the domains in Fig. 7(a) and Fig. 7(b) are powerset domains. However, their reduced product is not, since, for example, the element $\{1, 2, 3\}$

[9] In this case, $\underline{\rho}_A$ is a lower closure operator, and A is a complete sublattice of C.

is missing. In this paper, we focus only on powerset abstract domains. We would like an operation that takes as input two powerset abstract domains and gives the simplest powerset abstract domain that includes its operands. Therefore, the reduced product operation in itself is inappropriate for this kind of abstract domain refinement. To fix this, we need to extend the result of the reduced product in order to achieve closure under set union. The abstract domain operation that achieves this is called *disjunctive completion*. We denote by $A \sqcap_{Rel} B$ the operation that accepts two powerset abstract domains and returns the simplest powerset abstract domain that is more precise than A and B. Fig. 7(d) shows the result of applying this operation to the abstract domains in Fig. 7(a) and Fig. 7(b).

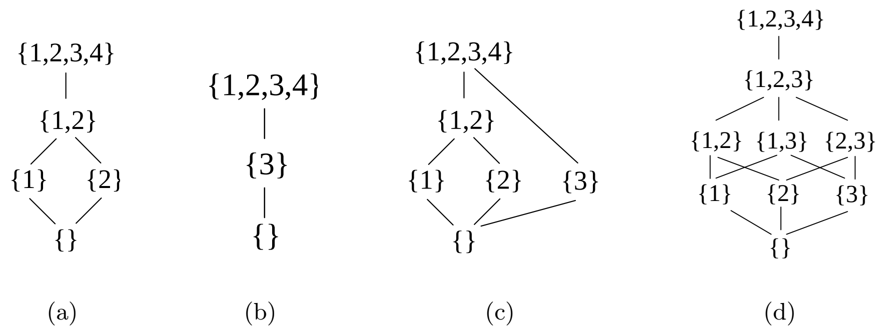


Fig. 7. Abstractions of $C = 2^{\{1,2,3\}}$: (a) A powerset abstract domain A^1; (b) A powerset abstract domain A^2; (c) The reduced product $A = A^1 \sqcap A^2$; and (d) The disjunctive completion of A