Static Analysis in Disjunctive Numerical Domains.

Sriram Sankaranarayanan, Franjo Ivančić, Ilya Shlyakhter, Aarti Gupta

NEC Laboratories America,
4 Independence Way, Princeton, NJ

Abstract. The convexity of numerical domains such as polyhedra, octagons, intervals and linear equalities enables tractable analysis of software for buffer overflows, null pointer dereferences and floating point errors. However, convexity also causes the analysis to fail in many common cases. Powerset extensions can remedy this shortcoming by considering disjunctions of predicates. Unfortunately, analysis using powerset domains can be exponentially more expensive as compared to analysis on the base domain.

In this paper, we prove structural properties of fixed points computed in commonly used powerset extensions. We show that a fixed point computed on a powerset extension is also a fixed point in the base domain computed on an "elaboration" of the program's CFG structure. Using this insight, we build analysis algorithms that approach path sensitive static analysis algorithms by performing the fixed point computation on the base domain while discovering an "elaboration" on the fly. Using restrictions on the nature of the elaborations, we design algorithms that scale polynomially in terms of the number of disjuncts. We have implemented a light-weight static analyzer as a part of the F-Soft project with encouraging initial results.

1 Introduction

Static analysis over numerical domains has been used to check programs for buffer overflows, null pointer references and other violations such as division by zero and floating point errors [25, 4, 12]. Numerical domains such as intervals, octagons and polyhedra maintain information about the set of possible values of integer and real-valued program variables along with their inter-relationships. The *convexity* of these domains makes the analysis tractable. On the other hand, fundamental limitations arising out of convexity leads to imprecision in the analysis, ultimately yielding many false alarms. Elimination of these false alarms is achieved through *path-sensitive analysis* by means of disjunctive domains obtained through powerset extensions. Such extensions can be constructed systematically from the base domain using standard techniques [14, 8].

Powerset extensions of numerical domains consider a disjunction of predicates at each program location. While the presence of these disjuncts helps surmount convexity limitations, the complexity of the analysis can be exponentially higher due to more complex domain operations and also due to the large number of disjuncts that can be produced during the course of the analysis. Furthermore, the presence of disjuncts require special techniques to lift the widening from the base domain up to the disjunctive domain [2].

Controlling the production of disjuncts during the course of the analysis is one of the key aspects of managing the complexity of the analysis. The design of such strategies can be performed by techniques that annotate data flow objects by partial trace information such as *trace partitioning* [20, 16], and other path-sensitive data-flow analysis techniques that implicitly manage complexity by joining predicates only when the property to be proved remains unchanged as a result [11], or "semantically" by careful domain construction [19, 2].

In this paper, we first show that fixed points computed over powerset extensions correspond to fixed points over the base domain computed on an "elaboration" of the CFG. As a result, the complexity of flow-sensitive analysis can also be controlled by means of a strategy for producing elaborations of the CFG being analyzed. We consider analysis techniques that perform the fixed point iteration hand in hand with the construction of the elaboration that characterizes the fixed point. As an application, we consider *bounded elaborations*, that correspond to power-set extensions wherein the number of disjuncts in each abstract object is bounded by a fixed number K. We discuss the implementation our ideas in a light weight static analyzer for the C language as a part of the F-Soft project and demonstrate promising results.

This paper is organized as follows: Section 2 presents preliminary concepts of abstract interpretation and presents numerical domains along with their limitations. Powerset extensions are presented in Section 3. Section 4 presents the notion of an elaboration and techniques for constructing an elaboration while performing the analysis. Section 5 describes our implementation and results over some benchmark programs.

2 Preliminaries

We present basic notions of abstract interpretation and numerical domains.

Programs and Invariants

Since the paper focuses on static analysis over numerical domains, we may regard programs as purely ranging over integer or real-valued variables. Let $V = \{x_1, \ldots, x_n\}$ denote integer-valued program variables, collectively referred to as $\boldsymbol{x}$. The program operations over these variables include numerical operations such as addition and multiplication. We shall assume first-order predicates over the program state belonging to an appropriate language. Given such a predicate ψ, the set of valuations to $\boldsymbol{x}$ satisfying ψ is denoted $[\![\psi]\!]$. A program is represented by its *Control-flow graph*(CFG).

Definition 1 (Control-flow Graphs (CFGs)). *Formally, a CFG is a tuple* $\Pi :\ \langle V, L, \mathcal{T}, \ell_0, \Theta\rangle$*:*

– L: *a set of locations (cutpoints);*
– $\mathcal{T}$: *a set of transitions (edges), where each transition* $\tau : \ \ell_i \rightarrow \ell_j$ *is an edge between the* pre-location ℓ_i *and a* post-location ℓ_j*. Each transition models the changes in the values of program variables using a transition relation.*
– $\ell_0 \in L$: *the* initial location*;* Θ *is an assertion over* $\boldsymbol{x}$ *representing the* initial condition*.*

A state s of the program maps each variable x_i to an integer value $s(x_i)$. Let Σ denote the set of program states. The relational semantics of a transition can be modeled using the notion of a (concrete) post condition:

Definition 2 (Post Condition). *Let* $S \subseteq \Sigma$ *be a set of states. The (concrete) post condition* $S' : \ post_{\Sigma}(S, \tau)$ *across a transition* τ *is a set of states* $S' \subseteq \Sigma$*. The post condition models the effect(s) of executing* τ *on each state satisfying* S*.*

An assertion ψ over $\boldsymbol{x}$ is an *invariant* of a CFG at a location ℓ iff it is satisfied by every state reachable at ℓ. An *assertion map* associates each location of a CFG with a predicate. An assertion map η is *invariant* if $\eta(\ell)$ is an invariant, for each $\ell \in L$. Invariants are established using the *inductive assertions method* due to Floyd and Hoare [13, 17].

Definition 3 (Inductive Assertion Maps). *An assertion map* η *is inductive iff it satisfies the following conditions:*

Initiation: $[\![\Theta]\!] \subseteq [\![\eta(\ell_0)]\!]$,
Consecution: *For each transition* $\tau : \ell_i \rightarrow \ell_j$,

$$post_{\Sigma}([\![\eta(\ell_i)]\!], \tau) \subseteq [\![\eta(\ell_j)]\!] \ .$$

It is well known that any inductive assertion map is invariant. However, the converse need not be true. The standard technique for proving an assertion invariant is to find an inductive assertion that strengthens it.

Abstract Interpretation

Abstract interpretation [7] is a generic technique for computing inductive assertions of CFGs using an iterative process. In order to compute an inductive map, we start from an initial map and repeatedly weaken the predicates mapped at each location to converge to a *fixed point*. The assertions labeling each location can be shown to be inductive when the fixed point is reached.

Abstract Domain. In order to carry out an abstract interpretation, we define an *abstract domain* along with some operations on the elements of the abstract domain known as the *domain operations*. Informally, an abstract domain is a lattice of predicates Γ over the program state including the assertions $\top$ and $\bot$ representing *true* and *false* respectively. The domain is defined by the abstract lattice $\langle \Gamma, \models \rangle$ and the concrete lattice of sets of program states ordered by inclusion $\left\langle 2^{\Sigma}, \subseteq \right\rangle$ along with the abstraction function $\alpha : \ 2^{\Sigma} \mapsto \Gamma$ and the concretization (or the meaning) function $\gamma : \ \Gamma \mapsto 2^{\Sigma}$. A key requirement is that α, γ form a *Galois connection* (see [7, 9] for comprehensive surveys). The abstract domain operations include:

Join Given $d_1, \ldots, d_m \in \Gamma$, their join $d : \ d_1 \sqcup \ldots \sqcup d_m \in \Gamma$ satisfies $d_i \models d$.
Meet (Intersection) Given $d_1, \ldots, d_m \in \Gamma$, their meet $d : \ d_1 \sqcap \ldots \sqcap d_m$ satisfies $d \models d_i$.
Post-Condition Given $d \in \Gamma$ and a transition τ, its abstract post condition $d' : \ post_{\Gamma}(d, \tau)$ satisfies

$$post_{\Sigma}(\gamma(d), \tau) \subseteq \gamma(post_{\Gamma}(d, \tau))\,.$$

Note that if the abstract domain is clear from context, we may drop the subscript from the abstract post condition.
Inclusion Test Given objects d_1 and d_2, decide if $d_1 \models d_2$.
Widening Given $d_1, d_2 \in \Gamma$ such that $d_1 \models d_2$, their widening $d : \ d_1 \nabla d_2$ over approximates the join, i.e., $d_1 \sqcup d_2 \models d$. Repeated applications of widening on an increasing sequence of abstract objects, guarantees convergence to a fixed point in a finite number of iterations.

Other operations of interest include *projection*, which is commonly used to eliminate variables that are out of scope in interprocedural analysis and the *weakest precondition*, which may be used to refine the abstraction in case of failure to prove a property.

Forward Propagation. An abstract assertion map $\eta : \ L \mapsto \Gamma$ labels each CFG location ℓ with an abstract object $\eta(\ell) \in \Gamma$. An abstract assertion map η is inductive iff the map $\gamma \circ \eta$ is an inductive assertion map. Given a CFG Π along with an abstract domain Γ, *forward propagation* seeks to construct an inductive abstract assertion map, iteratively as follows:

Initial Step The initial map $\eta^{(0)}$ is defined as follows:

$$\eta^{(0)}(\ell_0) = \begin{cases} \Theta, \ \ell = \ell_0, \\ \bot, \ \text{otherwise.} \end{cases}$$

Iterative Step The iterative step computes the join of the current assertion at a location ℓ with the post-condition of all its incoming transitions

$$\eta^{(i+1)}(\ell) = \eta^{(i)}(\ell) \sqcup \bigsqcup_{\tau_j : \ \ell_j \to \ell} post_{\Gamma}(\eta^{(i)}(\ell_j), \tau_j)\,.$$

For convenience, we denote this as $\eta^{(i+1)} = \mathfrak{F}(\eta^{(i)})$. Note that $\mathfrak{F}$ is *monotonic* w.r.t $\models$, i.e., $\eta^{(i)}(\ell) \models \eta^{(i+1)}(\ell)$ for all $\ell \in L$.
Convergence Convergence occurs if $\eta^{(i+1)}(\ell) \models \eta^{(i)}(\ell)$ for each $\ell \in L$.

For the sake of simplicity, we do not consider the use of narrowing to improve the fixed point in this discussion. Given an initial map $\eta^{(0)}$, forward propagation computes $\eta^{(i+1)}$ iteratively as $\mathfrak{F}(\eta^{(i)})$ until convergence $\eta^{(i+1)}(\ell) \models \eta^{(i)}(\ell)$. Such a map is a *fixed point* w.r.t $\mathfrak{F}$. It can be shown that a fixed point map is also inductive. Hence, if the forward propagation converges, it results in an inductive assertion at each cutpoint. Convergence is guaranteed in finitely many iterative steps if the domain satisfies the *ascending chain condition*. Examples of such domains include finite domains and notably the domain of linear equalities [18]. On the other hand, domains such as intervals and polyhedra do not satisfy this condition. Hence, the widening operation ∇ is used repeatedly to force convergence in finitely many steps.

Numerical Domains. Numerical domains such as intervals, octagons and polyhedra reason about the values of integer or real-valued program variables. These domains are widely used to check programs for buffer-overflows, null pointer dereferences, division-by-zero, floating point instabilities [4].

The *interval domain* consists of interval predicates of the form $\bigwedge_i x_i \in [l_i, u_i]$ with the possibility of open intervals. The complexity of the domain operations is linear in the number of variables. Analysis techniques for this domain have been widely studied [6, 21]. The *octagon domain* due to Miné consists of assertions of the form $\bigwedge \pm x_i \pm x_j \leq c$ along with interval constraints over the variables. The nature of the constraints in this domain permits a graphical representation and the computation of many domain operations using the shortest path algorithm as a primitive. The operations in this domain are at most cubic in the number of variables. The polyhedral domain consists of convex polyhedra over the program variables represented by constraints of the form $\bigwedge a_0 + a_1 x_1 + \cdots + a_n x_n \geq 0$ [10, 15]. Domain operations over this domain are expensive (exponential space in the size of the polyhedra). However, relaxations of the operations and the structure of the constraints in the domain can yield polynomial time approximations to these operations [24, 23, 22, 5].

One of the key properties of these domains is that of *convexity*. Convexity makes the domain operations tractable. However, it also limits the ability of these domains to represent sets of states. For instance, consider a convex predicate including states A and B represented as points $\boldsymbol{x}_1, \boldsymbol{x}_2$ in $\mathcal{R}^n$. Such a predicate necessarily includes states that lie on the line joining these two points. In many cases, the reachable states of a program form a non convex set in $\mathcal{R}^n$. Therefore, convex abstract domains cannot represent such sets without the addition of spurious states. Such a drawback leads to cases wherein the domain is fundamentally unable to compute an invariant that proves the property of interest.

Example 1. Figure 1 shows a program that stores the result of a condition $0 \leq i \leq 10$ in a variable x. The table to the right shows the invariants computed after each labeled location. Note that the invariant $i \leq 10$, required at L4 to prove the absence of overflows, cannot be established. Although the program is free from overflows, convex numerical domains will not be able to establish correctness.

Powerset extensions are used to remedy the problem of convexity.

```
int a[10]
if (i ≥ 0 ∧ i < 10) then
  L1: x := 1
else
  L2: x := 0
end if
L3: · · ·
if x = 1 then
  L4: a[i] := · · ·
end if
```

Location	Invariant
L1	$i \geq 0 \wedge i \leq 10 \wedge x = 1$
L2	$x = 0$
L3	$0 \leq x \wedge x \leq 1$
L4	$x = 1$

Fig. 1. Example program (left) and the octagon domain invariants (right)

3 Powerset Extensions

Given a base abstract domain of predicates, a *powerset extension* of the domain consists of disjunctions of the base domain predicates.

Definition 4 (Powerset extension). *A powerset extension of an abstract domain $\langle \Gamma, \models \rangle$ is given by the domain $\left\langle \hat{\Gamma}, \widehat{\models} \right\rangle$ such that*

$$\hat{\Gamma} = \{ S = \langle d_1, \ldots, d_m \rangle \mid d_i \in \Gamma,\ m \geq 0 \} .$$

The concretization function $\hat{\gamma}$ for a powerset extension is defined as $\hat{\gamma}(S) = \bigcup_{d \in S} \gamma(d)$. The abstraction function $\hat{\alpha}(X)$ can be defined in many ways, for instance $\hat{\alpha}(X) = \{\alpha(X)\}$. The ordering relation $\widehat{\models}$ may be defined in many ways to derive different extensions. However, any such definition needs to be faithful to the semantics induced by $\hat{\gamma}$, i.e. if $S_1 \widehat{\models} S_2$ then $\hat{\gamma}(S_1) \subseteq \hat{\gamma}(S_2)$.

Extending Partial Orders. The *natural powerset extension* is obtained by considering $\left\langle \hat{\Gamma}, \models_{\mathrm{N}} \right\rangle$ such that $S_1 \models_{\mathrm{N}} S_2$ *iff* $\hat{\gamma}(S_1) \subseteq \hat{\gamma}(S_2)$. This is the partial order induced by the concrete domain on the abstract domain through $\hat{\gamma}$. The *Hoare powerset extension* $\models_{\mathrm{P}}$ is a partial order defined as follows:

$$S_1 \models_{\mathrm{P}} S_2 \iff (\forall d_1 \in S_1)\ (\exists\ d_2 \in S_2)\ d_1 \models d_2 .$$

Informally, we require that every object in S_1 be "covered" by some object in S_2. This can be refined to yield a *Egli-Milner type* partial order $\models_{\mathrm{EM}}$ [1, 2]

$$S_1 \models_{\mathrm{EM}} S_2 \iff S_1 = \emptyset \text{ or } (S_1 \models_{\mathrm{P}} S_2 \text{ and } (\forall\ d_2 \in S_2)\ (\exists\ d_1 \in S_1)\ d_1 \models d_2) .$$

In addition to $S_1 \models_{\mathrm{P}} S_2$, each element in S_2 must cover some element in S_1.

Example 2. Consider the interval domain $\langle I, \sqsubseteq \rangle$ over variables x_1, x_2. Let $S_1 = \{\varphi_1 :\ x_1 \in [0, 1]\}$ and $S_2 = \{\psi_1 :\ x_1 \in [\frac{1}{2}, 2], \psi_2 :\ x_1 \in [-1, \frac{1}{2}]\}$. It is easily seen that $S_1 \sqsubseteq_{\mathrm{N}} S_2$, however $S_1 \not\sqsubseteq_{\mathrm{P}} S_2$ since each element of S_2 is incomparable with the element in S_1.

On the other hand let $S_3 = \{\xi_1 : \ x_1 \in [0,2], \ \xi_2 : \ x_1 \in [-1,0]\}$. Note that $S_1 \sqsubseteq_{\mathrm{P}} S_3$ since $\varphi_1 \sqsubseteq \xi_1$. On the other hand ξ_2 does not cover any object in S_1, hence $S_1 \not\sqsubseteq_{\mathrm{EM}} S_3$.

Consider the interval domain $\langle I, \sqsubseteq\rangle$ of conjunctions of closed, open and half-open intervals over the program variables and its natural powerset extension $\left\langle \hat{I}, \sqsubseteq_{\mathrm{N}} \right\rangle$. It is computationally hard to decide the $\sqsubseteq_{\mathrm{N}}$ relation.

Theorem 1. *Given $S_1, S_2 \in \hat{I}$, deciding if $S_1 \sqsubseteq_{\mathrm{N}} S_2$ is co-NP-hard.*

Proof. We perform a reduction from the problem of proving universality of DNF formulas. We introduce a variable x_i corresponding to each proposition p_i. The literal p_i is represented by the predicate $x_i \in [0, \infty)$ and $\neg p_i$ by $x_i \in (-\infty, 0)$. Each DNF clause translates into a interval domain predicate $\bigwedge x_i \in (l_i, u_i)$.Therefore, the validity of the propositional formula can be reduced to checking the inclusion $\{\top\} \sqsubseteq_{\mathrm{N}} \{T(D_1), \ldots, T(D_m)\}$, wherein $T(D_i)$ represents the interval predicate modeling the DNF clause D_i. $\square$

The hardness of $\models_{\mathrm{N}}$ extends to natural powerset extensions of most numerical domains and many non-numerical domains that are sufficiently powerful to enable the translation above. Other partial orders $\models_{\mathrm{P}}$ and $\models_{\mathrm{EM}}$ are easier to compute using $O(|S_1| + |S_2|)^2$ many base domain ($\models$) comparisons.

The domain operations in a powerset domain can be defined by suitably lifting the base domain operations. Notably, the join operation in a powerset domain reduces to a set union. The meet operation $S_1 \hat{\sqcap} S_2$ is given by the pairwise meet of elements from S_1, S_2. Post condition is computed element-wise; i.e., if $S = \{d_1, \ldots, d_k\} \in \hat{\Gamma}, \ \widehat{post}(S, \tau) = \{post(d_1, \tau), \ldots, post(d_k, \tau)\}$.

Widening operations can be obtained as extensions of the widening on the base domain using carefully crafted strategies [2]. The use of such widening operators frequently results in fixed points which satisfy inclusion using the $\models_{\mathrm{P}}$ or even the $\models_{\mathrm{EM}}$ ordering. Thus, even if a domain were designed to use joins over a stronger partial order, the final fixed point obtained may be over $\models_{\mathrm{P}}$ or the $\models_{\mathrm{EM}}$ ordering.

Example 3. Consider the program below:

$s := -1$
while $\cdots$ **do**
  $s := -s$ { Invariant: $(s = 1 \ \vee \ s = -1)$ }
end while

The invariant $s = 1 \ \vee \ s = -1$ is a fixed point in the powerset extension of the interval domain using the $\sqsubseteq_{\mathrm{P}}$ ordering.

CFG Elaboration

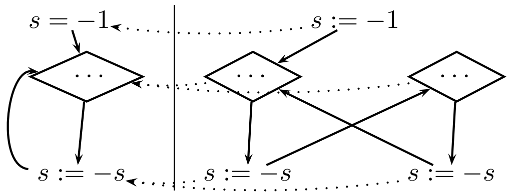
We now prove a simple connection between the fixed point obtained on a domain $\left\langle \hat{\Gamma}, \models_{\mathrm{P}} \right\rangle$ using forward propagation on a CFG Π and the fixed point in the base domain using the notion of an "*elaboration*". Intuitively, an elaboration of a CFG replicates each location of the CFG multiple times. Each such replication preserves all the outgoing transitions from the original location.

Definition 5. *Consider CFGs $\Pi_e : \langle L_e, \mathcal{T}_e, \ell_0', \Theta \rangle$ and $\Pi : \langle L, \mathcal{T}, \ell_0, \Theta \rangle$ over the same set of variables V. The CFG Π_e is an* elaboration *of Π iff there exists a map $\rho : L_e \mapsto L$ such that*

- *The initial location in Π_e maps to the initial location of Π: $\rho(\ell_0') = \ell_0$.*
- *Consider locations $\ell \in \Pi$ and $\ell_e \in \Pi_e$ such that $\rho(\ell_e) = \ell$. For each outgoing transition $\tau : \ell \rightarrow m \in \mathcal{T}$, there is an outgoing transition $\tau_e : \ell_e \rightarrow m_e \in \mathcal{T}_e$ such that $\rho(m_e) = m$. Furthermore every outgoing transition $\tau_e : \ell_e \rightarrow m_e \in \mathcal{T}_e$ is a replication of some transition $\tau : \rho(\ell_e) \rightarrow \rho(m_e) \in \mathcal{T}$.*

Each $\ell_e \in L_e$ is said to be a replication *of $\rho(\ell_e) \in L$. Note that every outgoing transition of $\rho(\ell_e)$ is replicated in ℓ_e. We denote the replication of the transition $\tau : \ell \rightarrow m$ starting from ℓ_e as $\tau(\ell_e) : \ell_e \rightarrow m_e$. An elaboration resembles a (structural) simulation relation between Π_e and Π.*

Example 4. The figure below shows a CFG Π from Example 3 along with an elaboration. The dashed line shows the relation ρ.



We shall now prove that every fixed point assertion map on a powerset domain $\left\langle \hat{\Gamma}, \models_{\mathrm{P}} \right\rangle$ on a CFG Π corresponds to a fixed point in the base domain $\langle \Gamma, \models \rangle$ on some elaboration Π_e and vice-versa.

Definition 6 (Collapsing). *Let $\eta_e : L_e \mapsto \Gamma$ be an assertion map on the elaboration Π_e in the base domain. Its collapse $C(\eta_e)$ is a map on the original CFG Π, $L \mapsto \hat{\Gamma}$ such that for each $\ell \in L$,*

$$C(\eta_e)(\ell) = \{ \eta(\ell_e) \mid \rho(\ell_e) = \ell \} \,.$$

The collapsing operator computes the disjunction of the domain objects at each replicated location.

Lemma 1. *If η_e is a fixed point map for Π_e in the domain $\langle \Gamma, \models \rangle$ then $C(\eta_e)$ is a fixed point map for Π in the domain $\left\langle \hat{\Gamma}, \models_{\mathrm{P}} \right\rangle$.*

Proof. (Sketch) For convenience we denote $\eta_c = C(\eta_e)$. It suffices to show initiation $\Theta \models_{\mathrm{P}} \eta_c(\ell_0)$ and consecution for each transition $\tau: \ \ell_i \to \ell_j$, we require $\widehat{post}(\eta_c(\ell_i), \tau) \models_{\mathrm{P}} \eta_c(\ell_j)$. Initiation is obtained by noting that initial states must be replicated in an elaboration. Expanding the definition for LHS,

$$\begin{aligned}\widehat{post}(\eta_c(\ell_i), \tau) &= \widehat{post}(\{\eta_e(\ell_e) | \rho(\ell_e) = \ell_i\}, \tau)\\ &= \{post(\eta_e(\ell_e), \tau) | \rho(\ell_e) = \ell_i\}\end{aligned}$$

Similarly the RHS is expanded $\eta_c(\ell_j) = \{\eta_e(\ell'_e) \mid \rho(\ell'_e) = \ell_e\}$. In order to show the containment, note that an elaboration requires that $\tau(\ell_{ie}): \ \ell_{ie} \to \ell_{je}$ should be an outgoing transition for each replication ℓ_{ie} with $\rho(\ell_{ie}) = \ell_i$ and $\rho(\ell_{je}) = \ell_j$.

Using the fact that η_e is a fixed point map, we note that each element $post(\eta_e(\ell_{ie}), \tau)$ on the LHS is contained in the element $\eta_e(\ell_{je})$ from the RHS. □

Conversely, the fixed point in $\left\langle \hat{\Gamma}, \models_{\mathrm{P}} \right\rangle$ induces an elaboration of the CFG.

Definition 7 (Induced Elaboration). *Let $\hat{\eta}$ be a fixed point map for Π in the domain $\left\langle \hat{\Gamma}, \models_{\mathrm{P}} \right\rangle$. Such a fixed point induces an elaboration Π_e and a induced map η_e defined as follows:*

- *Locations: Let $\hat{\eta}(\ell) = \{d_1, \ldots, d_m\}$. The elaboration contains replicated locations $\langle \ell, 1 \rangle, \ldots, \langle \ell, m \rangle \in L_e$, one per disjunct such that $\rho(\langle \ell, j \rangle) = \ell$. Also, $\eta_e(\langle \ell, j \rangle) = d_j$.*
- *Transitions: For each transition $\tau: \ell_i \to \ell_j$ we require an outgoing transition $\tau(\ell_i, k): \langle \ell_i, k \rangle \to \langle \ell_j, l \rangle$ for some l. The target index l is defined using the proof of consecution of $\hat{\eta}$ under τ: $\widehat{post}(\hat{\eta}(\ell_i), \tau) \models_{\mathrm{P}} \hat{\eta}(\ell_j)$.*
  Let $\hat{\eta}(\ell_i) = \{d_1, \ldots, d_m\}$ and $\eta(\ell_j) = \{e_1, \ldots, e_n\}$ (Note that we may represent the empty set equivalently by the singleton $\{\perp\}$). We require

  $$\widehat{post}(\{d_1, \ldots, d_m\}, \tau) \models_{\mathrm{P}} \{e_1, \ldots, e_n\}\,.$$

  However, $\widehat{post}(\{d_1, \ldots, d_m\}, \tau) = \{post(d_1, \tau), \ldots, post(d_m, \tau)\}$. By definition of $\models_{\mathrm{P}}$ order, we require for each k,

  $$(\forall\ k \in [1, m])(\exists\ l \in [1, n])\ post(d_k, \tau) \models e_l\,.$$

  Therefore, we set $\tau(\ell_i, k): \ \langle \ell_i, k \rangle \to \langle \ell_j, l \rangle$. It immediately follows that η_e satisfies consecution for this transition in the base domain $\langle \Gamma, \models \rangle$. Note that since the choice of a target index l is not unique, there may be many induced elaborations for a given assertion map.

Example 5. The elaboration shown in Example 4 is induced by the fixed point shown in Example 3.

Lemma 2. *Given a fixed point map η_c for Π in the domain $\left\langle \hat{\Gamma}, \models_{\mathrm{P}} \right\rangle$, its induced map η_e is a fixed point for the induced elaboration Π_e in the base domain $\langle \Gamma, \models \rangle$.*

Proof. The proof follows from the definition above.

Thus, elaborations are structural connections among the disjuncts of the final fixed point made explicit using a syntactic representation. In fact, interesting structural connections can be defined for powerset domains with other partial orders such as $\models_{\mathrm{EM}}$ or $\models_{\mathrm{N}}$. Making these connections explicit enables us to get around the NP-hardness of checking $\models_{\mathrm{N}}$. We defer the discussion of such extensions to an extended version of this document.

4 On-the-fly Elaborations

In the previous section, we have demonstrated a close connection between fixed points in a broad class of powerset domains and the fixed point in the base domain computed on a structural elaboration of the original CFG. As a result, analysis in powerset domains can be reduced to the process of an analysis on the base domain carried out on some CFG elaboration. As a caveat, we observe that even though it is possible to find some elaboration that produces the same fixed point as in the powerset extension with some widening operator, an *apriori* fixed elaboration scheme may not be able to produce the same fixed point.

In order to realize the full potential of a powerset extension, the process of producing an elaboration of the CFG needs to be dynamic, by considering *partial elaborations* of the CFG as the analysis progresses. Such a scheme can also be seen as a powerset extension wherein the containment relations between the individual disjuncts in a predicate are explicitly depicted.

Partial Elaboration A partial elaboration $\langle \Pi_e, U\rangle$ of a CFG $\Pi : \langle L, \mathcal{T}, \ell_0\rangle$ is a tuple consisting of a CFG $\Pi_e : \langle L_e, \mathcal{T}_e, \ell_{0e}\rangle$ and an *unresolved* set $U \subseteq L_e \times \mathcal{T}$ of pairs, each consisting of a location from Π_e and a transition from Π.

As with a CFG elaboration, each location $\ell_e \in \Pi_e$ is a replication of some location $\rho(\ell_e) \in \Pi$. Furthermore, for each transition $\tau : \ell_i \to \ell_j \in \Pi$ and each $\ell_{ie} \in L_e$ replicating ℓ_i, exactly one of the following holds:

- There exists a replicated transition $\tau(\ell_{ie}) :\ \ell_{ie} \to \ell_{je} \in \mathcal{T}_e$, or else,
- $\langle \ell_{ie}, \tau\rangle \in U$.

In other words, U contains all the outgoing transitions of Π which have not been replicated in a given location of Π_e. A partial elaboration is a (complete) elaboration iff $U = \emptyset$. Given a CFG Π, an *initial* partial elaboration Π_e^0 is given by $L_e^0 = \{\ell_0\}$, $\mathcal{T}_e = \emptyset$ and $U = \{\langle \ell_0, \tau\rangle \mid \tau :\ \ell_0 \to \ell_i\}$; in other words, the initial location of Π is replicated exactly once and all its outgoing transitions are unresolved. Two basic transformations are permitted on a partial elaboration:

Location Addition: We add a new location ℓ_{ie} to L_e replicating some node $\rho(\ell_{ie}) \in L$, i.e., $L_e' = L_e \cup \{\ell_{ie}\}$. Furthermore, all transitions in $\mathcal{T}$ outgoing from ℓ_i are treated as unresolved, i.e., $U' = U \cup \{\langle \ell_{ie}, \tau\rangle \mid \tau :\ \rho(\ell_{ie}) \to \ell_j\}$.

Transition Resolution: Given a pair $\langle \ell_{ie}, \tau : \ \ell_i \to \ell_j \rangle \in U$, we replicate τ in Π_e as $\tau(\ell_{ie}) : \ell_{ie} \to \ell_{je}$ for some replication ℓ_{je} of the target location ℓ_j.

Our analysis at each stage consists of a partial elaboration $\left\langle \Pi_e^{(i)}, U^{(i)} \right\rangle$ along with an abstract assertion map $\eta^{(i)} : \ L_e \mapsto \Gamma$. Each iteration involves an update to the map $\eta^{(i)}$ followed by an update to the partial elaboration.

Consider an unresolved entry $\langle \ell_e, \tau : \ \ell_i \to \ell_j \rangle \in U^{(i)}$. Its resolution involves the choice of a target node ℓ_{je} replicating ℓ_j. Let $d : \ \mathit{post}(\eta^{(i)}(\ell_{ie}), \tau)$ denote the result of the post condition of the unresolved transition. Furthermore, let $\ell_{(j,1)}, \ldots, \ell_{(j,m)} \in L_e$ denote the existing replications of the target location ℓ_j and $d_k = \eta^{(i)}(\ell_{(j,k)})$ denote the k^{th} disjunct. The choice of a target location for the transition $\tau(\ell_{ie})$ depends on the post condition d and the assertions $d_1, \ldots, d_m$. The target can either be chosen from the existing target replications $\ell_{(j,1)}, \ldots, \ell_{(j,m)}$, or a new node $\ell_{(j,m+1)}$ can be added as a new replication of the target. We shall assume a *merging heuristic* $\mathsf{MergeHeuristic}\,(d, \langle d_1, \ldots, d_m \rangle)$ to compute the index i s.t. $1 \leq i \leq m+1$ for the target location of the transition.

Formally, at each step we first update the map $\eta^{(i)} = \mathfrak{F}(\eta^{(i-1)})$ as described in Section 2. The partial elaboration $\left\langle \Pi_e^{(i)}, U^{(i)} \right\rangle$ is then refined by first choosing an unresolved pair $\langle \ell_{ie}, \tau : \ell_i \to \ell_j \rangle \in U$, and then applying a merging heuristic

$$\ell_{j,*} = \mathsf{MergeHeuristic}\left(\mathit{post}(\eta^{(i)}(\ell_{ie}), \tau), \left\langle \eta^{(i)}(\ell_{je}) \mid \ell_{je} \text{ replicates } \ell_j \right\rangle \right) .$$

The transition $\tau(\ell_{ie})$ is resolved as a result, and the entry $\langle \ell_{ie}, \tau \rangle$ is removed from $U^{(i)}$. If the merging heuristic results in a new location $\ell_{j,*}$, then new entries are added to $U^{(i)}$ to reflect unresolved outgoing transitions from the newly added location. If there are no more unresolved pairs in $U^{(i+1)}$, the partial elaboration is also a full elaboration. Thenceforth, the map η is simply propagated on this elaboration until fixed point is reached.

Upon termination, we guarantee that $U^{(i)} = \emptyset$, i.e., the partial elaboration is a full elaboration and the map $\eta^{(i)}$ is a fixed point map on this elaboration. Termination of the scheme depends mainly on the nature of the merging heuristic chosen. Since a transition from U is resolved at each step, termination is guaranteed as long as the creation of new locations ceases at some point in the analysis. A simple way to ensure this requirement is to bound the number of replications of each location to a prespecified limit $K > 0$.

Merging Heurstics. Formally a merging heuristic $\mathsf{MergeHeuristic}\,(d, \langle d_1, \ldots, d_m \rangle)$ chooses an index $1 \leq i \leq m+1 \leq K$ in order to compute the join $d_i \sqcup d$ if $i \leq m$ or create a new location in the partial elaboration as described above. The key goal of a merging heuristic is that the resulting join add as few extraneous concrete states as possible. Such extraneous states arise since the join is but an approximation of the disjunction of concrete states: $\gamma(d_1) \cup \gamma(d_2) \subseteq \gamma(d_1 \sqcup d_2)$.

In numerical domains, the states of the program can be viewed as points in $\mathcal{R}^n$. It is possible to correlate the extraneous concrete states with a distance metric on the abstract objects. Let $k(d, d')$ be a distance metric defined on Γ

and $\alpha \in \mathcal{R}$ be a *distance cutoff.* Let $d_{\min} = \text{argmin}\{k(d, d_i) | 1 \leq i \leq m\}$ be the "closest" abstract object to d w.r.t k. The merging heuristic induced by k, α is defined as

$$\mathsf{MergeHeuristic}\,(d, \langle d_1, \ldots, d_m \rangle) = \begin{cases} d_{m+1}, \;\; m < K \;\text{ and } k(d, d_{\min}) \geq \alpha \\ d_{\min}, \;\; m = K \text{ or } k(d, d_{\min}) < \alpha \end{cases}$$

In other words, a new location is spawned whenever it is possible to do so (i.e., $m < K$) and the closest object is farther than α apart in terms of distance. Failing these, the closest object is chosen as the target of the unresolved transition. The cutoff α ensures that newly formed disjuncts are initially well separated from the others in terms of the metric k.

The *Hausdorff distance*, is a commonly used measure of distance between two sets. Given $P, Q \subseteq \mathcal{R}^n$, their Hausdorff distance is defined as

$$\mathsf{Hausdorff}(P, Q) = max_{\boldsymbol{x} \in P}\{min_{\boldsymbol{y} \in Q} \; \{ \; ||\boldsymbol{x} - \boldsymbol{y}||\}\} \, .$$

While such metrics provide a good measure of the accuracy of the join, they are hard to compute. We shall use a range-based Hausdorff distance metric.

Range Distance Metric. Let $x_1, \ldots, x_n$ be the program variables and d_1, d_2 be abstract objects. For each variable x_i, we shall compute ranges $I_1 : \; [p_1, q_1]$ and $I_2 : \; [p_2, q_2]$ of the values of x_i. Such ranges may be efficiently computed for most numerical domains including the polyhedral domain by resorting to linear programming. The ranges are said to be *incompatible* if one of the two intervals is open in a direction where the other interval is closed, i.e., their Hausdorff distance is unbounded (∞). If the ranges are compatible, the Hausdorff distance is computed based on their end points. The overall distance is a lexicographic tuple $\langle m, s \rangle$ where m is the number of dimensions along which d_1, d_2 have incompatible ranges while s is the sum of the distances along the compatible dimensions.

Example 6. Consider the polyhedra $p_1 : \; 1 \leq x \leq 5 \; \wedge \; y \geq 0$ and $p_2 : \; -1 \leq y \leq 1 \; \wedge \; 10 \leq x \leq 20$. The ranges along x, $[1, 5]$ and $[10, 20]$ have a Hausdorff distance of 9. On the other hand the ranges along y are $[0, \infty)$ and $[-1, 1]$ are incompatible. The overall distance between p_1, p_2 is therefore $(1, 9)$.

Widening. Widening is applied to loops formed on the partial elaboration of the CFG by identifying cutpoints, i.e., a set of CFG locations that cut every loop in the CFG. Note that any loop in the partial elaboration results from a loop in the original CFG:

Lemma 3. *If C_e be a loop in a partial elaboration Π_e, then $\rho(C_e)$ is a loop in the original CFG.*

The converse is not true. Therefore, not all loops in a CFG be replicated as a loop in the partial elaboration. However, once a loop is formed in a partial elaboration, it remains a cycle regardless of the other edges or locations that may be added to it. Furthermore, the post condition computed along such new edges can only accelerate the termination once the widening phase has begun. These observations can be used to simplify the use of widening to that on the base domain, to reuse widening strategies available on the base domain to partial elaborations and finally, to limit the number of applications of widening. This is one of the key advantages of maintaining structural connections among the disjuncts in terms of a partial elaboration.

5 Applications

We consider an application of our ideas to a intra-procedural static analyzer for checking run time errors of systems programs such as buffer overflows and null pointer dereferences. Our prototype analyzer constructs a CFG representation by parsing while performing memory modeling for arrays and data structures using a flow insensitive pointer analysis. This is followed by model simplification using constant folding and range analysis. A linearization abstraction converts operations such as multiplication, integer division, modulo and bitwise logical operations into non-deterministic choices. Similarly, arrays and pointers are modeled by their allocated sizes while their contents are abstracted away.

Our analyzer is targeted towards proving buffer overflows and string access patterns of systems code. The analyzer is *context insensitive*; all function calls are inlined using caller ID variables to differentiate between calling contexts. All variables are assumed to have global scope. Reduction in the number of variables in the model is achieved by tracking live variables during the analysis and by creating small clusters of related variables. Clusters are detected by backward traversal of the CFG, collecting the variables that occur in the same expressions or conditions. The maximum number of variables in each cluster is artificially limited by a user specified parameter. For each cluster, statements involving variables that do not belong to the current cluster are abstracted away. The analysis is performed on each of these clusters. A property is considered proved only if it can be proved on at least one of the abstractions.

The analysis is performed using the polyhedral domain using base domain operations implemented in the PPL library [3] and relaxations described in our previous work [22]. The maximum number of disjuncts K and the maximum cluster sizes are parameters to this analysis. The merging heuristic used is induced by a slight modification of the range Hausdorff distance described previously. Back edges are tracked dynamically in the partial elaboration thereby avoiding unnecessary widening operations.

We analyzed a variety of benchmark programs using our analysis for different values of K. Table 1 shows the performance comparisons for a selection of benchmark programs. For each program "#Prop" indicates the total number of properties to be checked, "#C" indicates the number of clusters. We employ a clustering strategy wherein the number of variables per clusters is kept uniformly close to 15. For each value of K, we report the time taken (T) and the number of proofs("#P"). Timings were measured on an Intel Pentium 3GHz processor with

Table 1. Performance comparison using benchmark programs.

Name	KLOC	#Prop	#C	$K=1$		$K=2$		$K=5$		$K=10$	
				T	#P	T	#P	T	#P	T	#P
code1	1.5	136	56	143	44	134	76	233	77	370	77
code2	1.5	126	46	51	65	115	67	193	68	343	68
code3	2	189	92	207	84	232	81	367	84	600	83
code4	2	187	91	205	83	221	86	360	86	587	86
code5	1.9	142	10	31	44	42	44	101	50	191	52
code6	15	634	22	215	176	270	176	375	182	652	184

4GB RAM. The gains produced by the use of disjunctive invariants are tangible and pronounced in some cases. The lack of monotonicity of our scheme, evident in "code3", can be remedied by performing the analysis for smaller values of K before attempting a large value. For small number of disjuncts, the overhead of merging disjuncts seems to be linear in K.

The false positive rate in our analysis is high, primarily due to the coarseness of the abstractions currently employed and the lack of a clustering strategy that performs uniformly well on all the benchmarks. We hope to improve our abstraction by providing better approximations for integer division, modulo and some bitwise operators, tracking the null terminators of strings and modeling contents of arrays and strings. These refinements can substantially reduce the number of false positives for our analyzer. We also hope to study disjunctive interprocedural analysis on more tractable domains such as octagons and intervals.

References

1. Abramsky, S., and Jung, A. Domain theory. In S.Abramsky and D.M. Gabbay and T.S.E. Maibum*, editors, Handbook of Logic in Computer Science*, vol. 3. Clarendon Press, Oxford, UK, 1994, ch. 1, pp. 1–168.
2. Bagnara, R., Hill, P. M., and Zaffanella, E. Widening operators for powerset domains. In *Proceedings of the fifth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI 2004)* (2004), vol. 2947 of *LNCS*, pp. 135—148.
3. Bagnara, R., Ricci, E., Zaffanella, E., and Hill, P. M. Possibly not closed convex polyhedra and the Parma Polyhedra Library. In *SAS* (2002), vol. 2477 of *LNCS*, Springer–Verlag, pp. 213–229.
4. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., and Rival, X. A static analyzer for large safety-critical software. In *ACM SIGPLAN PLDI'03* (June 2003), vol. 548030, ACM Press, pp. 196–207.
5. Clarisó, R., and Cortadella, J. The octahedron abstract domain. In *Static Analysis Symposium* (2004), vol. 3148 of *LNCS*, Springer–Verlag, pp. 312–327.
6. Cousot, P., and Cousot, R. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming* (1976), Dunod, Paris, France, pp. 106–130.

7. Cousot, P., and Cousot, R. Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Principles of Programming Languages* (1977), pp. 238–252.
8. Cousot, P., and Cousot, R. Systematic design of program analysis frameworks. In *Symposium on Principles of Programming Languages (POPL 1979)* (1979), ACM Press, New York, NY, pp. 269–282.
9. Cousot, P., and Cousot, R. Comparing the Galois connection and widening/narrowing approaches to Abstract interpretation, invited paper. In *PLILP '92* (1992), vol. 631 of *LNCS*, Springer–Verlag, pp. 269–295.
10. Cousot, P., and Halbwachs, N. Automatic discovery of linear restraints among the variables of a program. In *ACM POPL* (Jan. 1978), pp. 84–97.
11. Das, M., Lerner, S., and Seigle, M. ESP: Path-sensitive program verification in polynomial time. In *Proceedings of Programming Language Design and Implementation (PLDI 2002)* (2002), ACM Press, pp. 57–68.
12. Dor, N., Rodeh, M., and Sagiv, M. CSSV: Towards a realistic tool for statically detecting all buffer overflows in C. In *Proc. PLDI'03* (2003), ACM Press.
13. Floyd, R. W. Assigning meanings to programs. *Proc. Symposia in Applied Mathematics 19* (1967), 19–32.
14. Giacobazzi, R., and Ranzato, F. Optimal domains for disjunctive abstract intepretation. *Sci. Comput. Program. 32*, 1-3 (1998), 177–210.
15. Halbwachs, N., Proy, Y., and Roumanoff, P. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design 11* (1997), 157–185.
16. Handjieva, M., and Tzolovski, S. Refining static analyses by trace-based partitioning using control flow. In *SAS* (1998), vol. 1503 of *LNCS*, Springer–Verlag, pp. 200–214.
17. Hoare, C. A. R. An axiomatic basis for computer programming. *Commun. ACM 12*, 10 (1969), 576–580.
18. Karr, M. Affine relationships among variables of a program. *Acta Inf. 6* (1976), 133–151.
19. Manevich, R., Sagiv, S., Ramalingam, G., and Field, J. Partially disjunctive heap abstraction. In *Static Analysis Symposium (SAS)* (2004), vol. 3148 of *LNCS*, Springer–Verlag, pp. 265–279.
20. Mauborgne, L., and Rival, X. Trace partitioning in abstract interpretation based static analyzers. In *ESOP* (2005), vol. 3444 of *LNCS*, Springer–Verlag, pp. 5–20.
21. Rugina, R., and Rinard, M. Symbolic bounds analysis of pointers, array indices, and accessed memory regions. In *Proc. Programming Language Design and Implementation (PLDI'03)* (2000), ACM Press.
22. Sankaranarayanan, S., Colón, M., Sipma, H. B., and Manna, Z. Efficient strongly relational polyhedral analysis. In *VMCAI* (2006), LNCS, Springer–Verlag, pp. 111–125.
23. Sankaranarayanan, S., Sipma, H. B., and Manna, Z. Scalable analysis of linear systems using mathematical programming. In *Verification, Model-Checking and Abstract-Interpretation (VMCAI 2005)* (January 2005), vol. 3385 of *LNCS*.
24. Simon, A., King, A., and Howe, J. M. Two variables per linear inequality as an abstract domain. In *LOPSTR* (2003), vol. 2664 of *Lecture Notes in Computer Science*, Springer, pp. 71–89.
25. Wagner, D., Foster, J., Brewer, E., , and Aiken, A. A first step towards automated detection of buffer overrun vulnerabilities. In *Proc. Network and Distributed Systems Security Conference* (2000), ACM Press, pp. 3–17.