Program Analysis using Symbolic Ranges

Sriram Sankaranarayanan, Franjo Ivančić, Aarti Gupta

NEC Laboratories America,
{srirams,ivancic,agupta}@nec-labs.com

Abstract. Interval analysis seeks static lower and upper bounds on the values of program variables. These bounds are useful, especially for inferring invariants to prove buffer overflow checks. In practice, however, intervals by themselves are often inadequate as invariants due to the lack of relational information among program variables.
In this paper, we present a technique for deriving symbolic bounds on variable values. We study a restricted class of polyhedra whose constraints are stratified with respect to some variable ordering provided by the user, or chosen heuristically. We define a notion of normalization for such constraints and demonstrate polynomial time domain operations on the resulting domain of symbolic range constraints. The abstract domain is intended to complement widely used domains such as intervals and octagons for use in buffer overflow analysis. Finally, we study the impact of our analysis on commercial software using an overflow analyzer for the C language.

1 Introduction

Numerical domain static analysis has been used to prove safety of programs for properties such as the absence of buffer overflows, null pointer dereferences, division by zero, string usage and floating point errors [30, 3, 13]. Domains such as *intervals, octagons, and polyhedra* are used to symbolically over-approximate the set of possible values of integer and real-valued program variables along with their relationships under the *abstract interpretation framework* [19, 8, 11, 21, 6, 25, 17, 27]. These domains are classified by their *precision*, i.e, their ability to represent sets of states, and *tractability*, the complexity of common operations such as union (join), post condition, widening and so on. In general, enhanced precision leads to more proofs and less false positives, while resulting in a costlier analysis.

Fortunately, applications require a domain that is "*precise enough*" rather than "*most precise*". As a result, research in static analysis has resulted in numerous trade-offs between precision and tractability. The octagon abstract domain, for instance, uses polyhedra with two variables per constraint and unit coefficients [21]. The restriction yields fast, polynomial time domain operations. Simultaneously, the pairwise comparisons captured by octagons also express and prove many common run time safety issues in practical software [3]. Nevertheless, a drawback of the octagon domain is its inability to reason with properties that may need constraints of a more complex form. Such instances arise frequently.

In this paper, we study *symbolic range constraints* to discover symbolic expressions as bounds on the values of program variables. Assuming a linear ordering among the program variables, we restrict the bound for a variable x to involve variables of order strictly higher than x. Thus, symbolic ranges can also be seen as polyhedra with triangular constraint matrices. We present important syntactic and semantic properties of these constraints including a sound but incomplete proof system derived through syntactic rewriting, and a notion of normalization under which the proof system is complete. Using some basic insights into the geometry of symbolic range constraints, we study algorithms for the various domain operations necessary to carry out program verification using symbolic range constraints. We also study the practical impact of our domain on large programs including performance comparisons with other domains.

Related work. Range analysis has many applications in program verification and optimization. Much work has focused on the interval domain and its applications. Cousot & Cousot present an abstract interpretation scheme for interval analysis using widening and narrowing [8]. Recent work has focused on the elimination of widenings/narrowings in the analysis using linear programming [23], rigorous analysis of the data flow equations [28], and *policy iteration* [7, 14].

Blume & Eigenmann study symbolic ranges for applications in compiler optimizations [4]. Their approach allows ranges that are *non-linear* with multiplication and max/min operators. However, the presence of non-linearity leads to domain operations of exponential complexity. Whereas polynomial time operations are derived heuristically, the impact of these heuristics on precision is unclear. Even though some aspects of our approach parallel that of Blume *et al.*, there are numerous fundamental differences: we focus on range constraints that are always linear, convex and triangulated based on a single, explicit variable ordering. These restrictions vastly simplify the design and implementation of domain operations while providing some insights into the properties of symbolic range constraints. Finally, we provide an experimental evaluation of the efficacy of our domain for eliminating array bounds checks in practical examples.

Symbolic ranges may also be obtained using the LP-based approach of Rugina & Rinard [23, 32]. The bounds obtained relate the current value of a variable and the values of parameters at the function entry. The advantage of this approach is its freedom from heuristics such as widening/narrowing. However, the LP-formulation is based on a weaker proof system for generating consequences of linear inequalities by directly comparing coefficients, and potentially generates weaker invariants.

Symbolic range constraints are also used in the path-based analysis tool ARCHER due to Xie et al. [31]. However, the bounding expressions used in their approach can involve at most one other variable.

We illustrate symbolic ranges for invariant computation using a motivating example presented in Fig. 1(a). Assuming that the analysis starts at the function `foo`, we analyze whether the assertion at the end of the function holds. Fig. 1(b) shows the control flow graph for this example after program slicing. Fig. 1(c) shows an interval analysis computation for this example. In this example, the

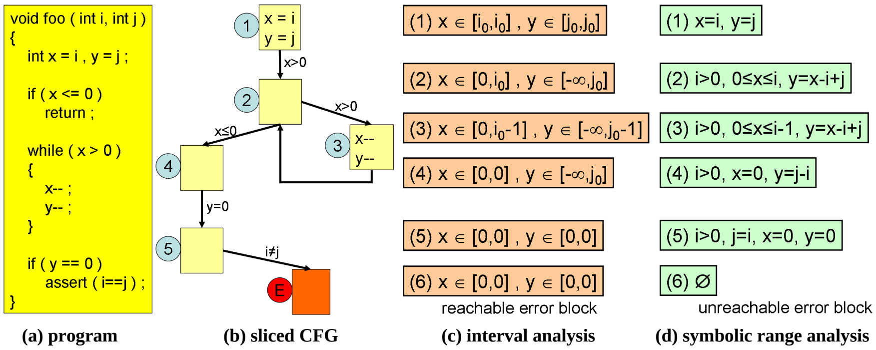

Fig. 1. A motivating example

interval analysis is not powerful enough to conclude that the assertion can never be violated.

Consider the analysis using symbolic ranges, for the variable ordering `i`,`j`, `x`, `y` (see Fig. 1(d)). Since symbolic ranges can represent the loop invariant `y`=`x-i+j`, the analysis discovers that for `x`=`y`=0 this implies `i`=`j` at the point of the assertion. Note also that this assertion cannot be proved using octagons, since the loop invariant is not expressible in terms of octagonal relationships.

2 Preliminaries

We assume that all program variables are conservatively modeled as reals. Our analysis model does not consider features such as complex data structures, procedures and modules. These may be handled using well-known extensions [22]. Let C be the first order language of assertions over free variables $\boldsymbol{x}$, and $\models \subseteq C \times C$ denote entailment. An assertion φ represents a set of models $[[\varphi]]$.

Definition 1 (Control Flow Graph). *A Control Flow Graph (CFG)* $\Pi : \langle \boldsymbol{x}, L, E, \mathsf{c}, \mathsf{u}, \ell_0 \rangle$ *consists of variables* $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$*, locations* L *and edges* E *between locations. Each edge* E *is labeled by a condition* $\mathsf{c}(e) \in C$*, and an update* $\mathsf{u}(e) : \ \boldsymbol{x} \ := \ f(\boldsymbol{x})$*.* $\ell_0 \in L$ *is the* start location.

A state of the program is a tuple $\langle \ell, \boldsymbol{a} \rangle$ where $\ell \in L$ is a location and $\boldsymbol{a}$ represents a valuation of the program variables $\boldsymbol{x}$. Given a CFG Π, an *assertion map* $\eta : L \mapsto C$ is a function mapping each location $\ell \in L$ to an assertion $\eta(\ell) \in C$. An assertion map characterizes a set of states $\langle \ell, \boldsymbol{a} \rangle$ such that $\boldsymbol{a} \in [[\eta(\ell)]]$. Let $\eta_1 \models \eta_2$ iff $\forall \ell \in L, \ \eta_1(\ell) \models \eta_2(\ell)$. Given an assertion φ and an edge $e : \ \ell \to m \in E$, the (concrete) post-condition of φ wrt e, denoted $post(\varphi, e)$ is given by the first order assertion $post(\varphi, e) : \ (\exists \ \boldsymbol{x}_0) \ \varphi[\boldsymbol{x}_0] \ \wedge \ \mathsf{c}(e)[\boldsymbol{x}_0] \wedge \boldsymbol{x} = \mathsf{u}(e)[\boldsymbol{x}_0]$.

Definition 2 (Inductive Assertion Map). *An assertion map* η *is* inductive *iff (a)* $\eta(\ell_0) \equiv true$*, and (b) for all edges* $e : \ell \to m$*,* $post(\eta(\ell), e) \models \eta(m)$*.*

A safety property Γ is an assertion map labeling each location with a property to be verified. In order to prove a safety property Γ, we find an inductive assertion map η, such that $\eta \models \Gamma$. "*Concrete interpretation*" can be used to construct the inductive invariant map. Consider an *iterative sequence* of assertion maps $\eta^0, \eta^1, \ldots, \eta^N, \cdots$.

$$\eta^0(\ell) = \begin{cases} \mathit{true}, & \ell = \ell_0, \\ \mathit{false}, & \text{otherwise.} \end{cases} \quad \text{and} \quad \eta^{i+1}(\ell) = \eta^i(\ell) \ \vee \bigvee_{e:\ m \to \ell} \mathit{post}(\eta^i(m), e)$$

Note that $\eta^i \models \eta^{i+1}$. The iteration *converges* if $(\exists N > 0)\ \eta^{N+1} \models \eta^N$. If the iteration converges in $N > 0$ (finitely many) steps, the result η^N is an inductive assertion. However, the iteration may not converge for all programs. Furthermore, detecting convergence is undecidable, in general. As a result, concrete interpretation, as shown above, is impractical for programs. Therefore, we over-approximate the *concrete interpretation* in a suitable *abstract domain* [9, 10].

Abstract domains. An abstract domain is a bounded lattice $\langle A, \sqsubseteq, \sqcap, \sqcup, \top, \bot \rangle$. It is useful to think of A as an assertion language and $\sqsubseteq$ as an entailment relation. The meet $\sqcap$ and the join $\sqcup$ are approximations of the logical conjunction and disjunction respectively. Formally, we require functions $\alpha : C \mapsto A$ and $\gamma : A \mapsto C$ known as the *abstraction* and *concretization* functions resp. that form a *Galois connection* (see [9, 10] for a complete description). An abstract post condition operator $\mathit{post}_A(a, e)$ over-approximates the concrete post condition such that for all $a \in A$, $\mathit{post}(\gamma(a), e) \models \gamma(\mathit{post}_A(a, e))$. An *abstract domain map* $\pi : L \mapsto A$ maps each location $\ell \in L$ to an abstract element $\pi(\ell)$. The concrete iteration sequence is generalized to yield an abstract iteration sequence:

$$\pi^0(\ell) = \begin{cases} \top, & \text{if } \ell = \ell_0 \\ \bot, & \text{otherwise} \end{cases} \quad \mathit{and} \quad \pi^{i+1}(\ell) = \pi^i(\ell) \ \sqcup \bigsqcup_{e:\ m \to \ell} \mathit{post}_A(\pi^i(m), e)\,.$$

Again, $\pi^i \sqsubseteq \pi^{i+1}$, and the iteration converges if $\exists N > 0$ s.t. $\pi^{N+1} \sqsubseteq \pi^N$. If convergence occurs then it follows that $\gamma \circ \pi^N$ is an inductive assertion. If the lattice A is of finite height or satisfies the *ascending chain condition*, convergence is always guaranteed. On the other hand, many of the domains commonly used in program verification do not exhibit these conditions. Convergence, therefore, needs to be forced by the use of *widening*.

Formally, given a_1, a_2, their widening $a_1 \nabla a_2$ satisfies $a_2 \sqsubseteq (a_1 \sqcup a_2)$. Additionally, given an infinite sequence of objects $a_1, \ldots, a_m, \ldots$, the *widened sequence* given by $b_0 = \bot$, and $b_{i+1} = b_i \nabla (b_i \sqcup a_i)$, converges in finitely many steps. In summary, the abstract iteration requires the following operations: (a) *Join* $\sqcup$ (*meet* $\sqcap$) over-approximates the logical or (and), (b) *Abstract post condition* post_A over-approximates *post*, (c) *Inclusion test* $\sqsubseteq$ to check for the termination of the iteration, and (d) *Widening* operator ∇ to force convergence. In practice, we also require other operations such as projection and narrowing.

3 Symbolic Range Constraints

Let $\mathcal{R}$ represent the reals and $\mathcal{R}^+$, the set of *extended reals* ($\mathcal{R} \cup \{\pm\infty\}$). Let $\boldsymbol{x}$ denote a vector of $n > 0$ real-valued variables. The i^{th} component of the vector $\boldsymbol{x}$ is written x_i. We use A, B, C to denote matrices. Throughout this section, we fix a variable ordering given by $x_1 \prec x_2 \prec \cdots \prec x_n$, with the index i of a variable x_i being synonymous with its rank in this ordering.

A *linear expression* is of the form $\mathsf{e} : \boldsymbol{c}^T\boldsymbol{x}+d$ where $\boldsymbol{c}$ is a vector of coefficients over the reals, while $d \in \mathcal{R}^+$ is the constant coefficient. By convention, a linear expression of the form $c^T\boldsymbol{x} \pm \infty$ is identical to $\mathbf{0}^T\boldsymbol{x} \pm \infty$. For instance, the expression $2x_1 + \infty$ is identical to $0x_1 + \infty$. A *linear inequality* is of the form $\mathsf{e} \bowtie 0$, where $\bowtie \in \{\geq, \leq, =\}$. A *linear constraint* is a conjunction of finitely many linear inequalities $\varphi : \bigwedge_i \mathsf{e}_i \geq 0$.

Given an inequality $e \geq 0$, where e is not a constant, its *lead variable* x_i is the least index i s.t. $c_i \neq 0$. We may write such an inequality in the *bounded form* $x_i \widehat{\bowtie} \mathsf{e}_i$, where x_i is the lead variable and $\mathsf{e}_i = \frac{1}{c_i}\mathsf{e} - x_i$. The sign $\widehat{\bowtie}$ denotes the reversal of the direction of the inequality if $c_i < 0$. As an example, consider the inequality $2x_2 + 3x_5 + 1 \leq 0$. Its lead variable is x_2 and bounded form is $x_2 \leq -\frac{3}{2}x_5 - \frac{1}{2}$. We reuse the $\models$ relation to denote entailment among linear constraints in the first order theory of linear arithmetic.

Definition 3 (Symbolic Range Constraint). *A symbolic range constraint (*SRC*) is of the form $\varphi : \bigwedge_{i=1}^n \mathsf{l}_i \leq x_i \leq \mathsf{u}_i$ where for each $i \in [1, n]$, the linear expressions $\mathsf{l}_i, \mathsf{u}_i$ are made up of variables in the set $\{x_{i+1}, \ldots, x_n\}$. In particular, $\mathsf{l}_n, \mathsf{u}_n$ are constants. The linear assertions false and true are also assumed to be* SRC*s.*

The absence of a bound for x_j is modeled by setting the bound to $\pm\infty$. Given an SRC $\varphi : \bigwedge_{j=1}^n \mathsf{l}_j \leq x_j \leq \mathsf{u}_j$, let $\varphi_{[\mathsf{i}]}$ denote the assertion $\bigwedge_{j=i}^n \mathsf{l}_j \leq x_j \leq \mathsf{u}_j$.

Example 1. $\varphi : x_2+4 \leq x_1 \leq 2x_3+x_2+4 \wedge -x_3 \leq x_2 \leq x_3+4 \wedge -\infty \leq x_3 \leq 0$ is a SRC. The variable ordering is $x_1 \prec x_2 \prec x_3$. The bound for x_1 involves $\{x_2, x_3\}$, x_2 involves $\{x_3\}$ and x_3 has constant bounds.

Implied constraints & normalization. Given a symbolic range $\mathsf{l}_i \leq x_i \leq \mathsf{u}_i$, its *implied inequality* is $\mathsf{l}_i \leq \mathsf{u}_i$. Note that the implied inequality $\mathsf{l}_i \leq \mathsf{u}_i$ only involves variables $x_{i+1}, \ldots, x_n$.

Definition 4 (Normalization). *A* SRC *is* normalized *iff for each variable bound $\mathsf{l}_i \leq x_i \leq \mathsf{u}_i$, $\varphi_{[\mathsf{i+1}]} \models \mathsf{l}_i \leq \mathsf{u}_i$. By convention, the empty and universal* SRC *are normalized.*

Example 2. The SRC φ from Example 1 is not normalized. The implied constraint $0 \leq 2x_3$ derived from the range $x_2 + 4 \leq x_1 \leq 2x_3 + x_2 + 4$ is not implied by $\varphi_{[2]}$. The equivalent SRC φ' is normalized:

$$\varphi' : x_2 + 4 \leq x_1 \leq 2x_3 + x_2 + 4 \;\wedge\; -x_3 \leq x_2 \leq x_3 + 4 \;\wedge\; 0 \leq x_3 \leq 0$$

Unfortunately, not every SRC has a normal equivalent. The SRC $\psi: \ x_2 - x_3 \leq x_1 \leq 1 \ \wedge \ 0 \leq x_2 \leq 2 \ \wedge \ 0 \leq x_3 \leq 2$ forms a counter-example. The projection of ψ on the $\{x_2, x_3\}$ is a five sided polygon, whereas any SRC in 2D is a *trapezium*.

Weak optimization algorithms. Optimization is used repeatedly as a primitive for other domain operations including abstraction, join and intersection. Consider the optimization instance $\min.\ (\mathsf{e} : \boldsymbol{c}^T\boldsymbol{x} + d)$ s.t. φ. Let φ be a satisfiable SRC with bound $\mathsf{l}_j \leq x_j \leq \mathsf{u}_j$ for index $0 \leq j < n$. We let $e \xrightarrow{\varphi,j} e'$ denote the replacement of x_j in e by l_j (lower bound in φ) if its coefficient in e is positive, or u_j otherwise.

$$\text{Formally, } \mathsf{e}' = \begin{cases} \mathsf{e} - c_j x_j + c_j \mathsf{l}_j, \ c_j \geq 0, \\ \mathsf{e} - c_j x_j + c_j \mathsf{u}_j, \ c_j < 0. \end{cases}$$

The *canonical sequence*, given by $\mathsf{e} \xrightarrow{\varphi,1} \mathsf{e}_1 \cdots \xrightarrow{\varphi,n} \mathsf{e}_n$, replaces variables in the ascending order of their indices. The canonical sequence, denoted in short by $\mathsf{e} \overset{\varphi}{\twoheadrightarrow} \mathsf{e}_n$, is unique, and yields a unique result. The following lemma follows from the triangularization of SRCs:

Lemma 1. *For the canonical sequence* $\mathsf{e} \xrightarrow{\varphi,1} \cdots \xrightarrow{\varphi,n} \mathsf{e}_n$*, each intermediate expression* e_i *involves only the variables in* $\{x_{i+1}, \ldots, x_n\}$*. Specifically,* $\mathsf{e}_n \in \mathcal{R}^+$*.*

Example 3. Consider the SRC φ' defined in Example 2 and the expression $\mathsf{e} : -3x_1 + 2x_2 + 8x_3$. This yields the sequence $-3x_1 + 2x_2 + 8x_3 \xrightarrow{\varphi',1} -x_2 + 2x_3 - 12 \xrightarrow{\varphi',2} x_3 - 16 \xrightarrow{\varphi',3} -16$.

It follows that e_n under-approximates the minima of the optimization problem, and if φ is normalized, weak optimization computes the exact minima; the same result as any other LP solver.

Theorem 1 (Weak Optimization Theorem). *Given a constraint* φ *and the sequence* $e \overset{\varphi}{\twoheadrightarrow} \mathsf{e}_n$, $\varphi \models \mathsf{e} \geq \mathsf{e}_n$*. Furthermore, if* φ *is normalized then* $\mathsf{e}_n = \min\ \mathsf{e}$ s.t. φ*.*

Weak optimization requires $O(n)$ rewriting steps, each in turn involving arithmetic over expressions of size $O(n)$. Therefore, the complexity of weak optimization for a SRC with n constraints is $O(n^2)$.

Example 4. From Theorem 1, -16 is the exact minimum in Example 3. Consider the equivalent constraint φ from Example 1. The same objective minimizes to $-\infty$ (unbounded) if performed w.r.t. φ.

Optimization provides an inference mechanism: given $d = \min\ \mathsf{e}$ s.t. φ, we infer $\varphi \models \mathsf{e} \geq d$. By Theorem 1, an inference using weak optimization is always sound. It is also complete, if the constraint φ is also normalized. Given SRC φ, we write $\varphi \models_W \mathsf{e} \geq 0$ to denote inference of $\mathsf{e} \geq 0$ from φ by weak optimization. Similarly, $\varphi \models_W \bigwedge_i \mathsf{e}_i \geq 0$ iff $(\forall\ i)\ \varphi \models_W \mathsf{e}_i \geq 0$.

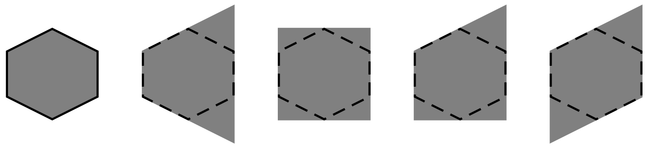

Fig. 2. Four possible SRC abstractions of a 2D hexagon (among many others).

Optimization for SRCs can also be solved by efficient algorithms such as SIMPLEX or interior point techniques. We will henceforth refer to such techniques as *strong optimization* techniques. In practice, however, we prefer weak optimization since (a) it out-performs LP solvers, (b) is less dependent on floating point arithmetic, and (c) allows us to draw sound inferences wherever required. As a curiosity, we also note that well-known examples such as Klee-Minty cubes and Goldfarb cubes that exhibit worst case behavior for SIMPLEX algorithms happen to be SRCs [5]. It is unclear if such SRCs will arise in practical verification problems. For the rest of the paper, we will assume optimization is always performed using weak optimization. Nevertheless, any call to weak optimization can be substituted by a call to strong optimization. Experimental results shown in Section 6 provide further justification for this choice.

We also use optimization to compare expressions wrt a given SRC φ. We write $\mathsf{e}_1 \gg_\varphi \mathsf{e}_2$ iff $\varphi \models_W \mathsf{e}_1 \geq \mathsf{e}_2$. Expressions are equivalent, written $\mathsf{e}_1 \equiv_\varphi \mathsf{e}_2$, if $\varphi \models \mathsf{e}_1 = \mathsf{e}_2$, and incomparable, denoted $\mathsf{e}_1 \Diamond_\varphi \mathsf{e}_2$, if neither inequality holds.

Abstraction. The abstraction function converts arbitrary first-order formulae to symbolic ranges. In practice, programs we analyze are first *linearized*. Therefore, abstraction needs to be defined only on polyhedra. Abstraction is used as a primitive operation that organizes arbitrary linear constraints into the form of SRCs.

Let ψ be a polyhedron represented as a conjunction of linear inequalities $\bigwedge_i \mathsf{e}_i \geq 0$. We seek a SRC $\varphi : \alpha(\psi)$ such that $\psi \models \varphi$. Unfortunately, this SRC abstraction $\alpha(\psi)$ may not be uniquely defined. Figure 2 shows possible SRC abstractions of a hexagon in 2 dimensions that are all semantically incomparable.

Abstraction of a given polyhedron ψ is performed by sequentially inserting the inequalities of ψ into a target SRC, starting initially with the SRC *true*. The result is an SRC $\alpha(\psi)$.

Inequality Insertion. Let φ be a SRC and $\mathsf{e}_j \geq 0$ be an inequality. As a primitive we consider the problem of deriving an abstraction $\alpha(\varphi \wedge \mathsf{e}_j \geq 0)$. We consider the case wherein $x_j \leq \mathsf{b}_j$ is the bounded form of e_j. The case where the bounded form is $x_j \geq \mathsf{b}_j$ is handled symmetrically. Also, let $\mathsf{l}_j \leq x_j \leq \mathsf{u}_j$ be the existing bounds for x_j in φ.

Using expression comparison, we distinguish three cases, (a) $\mathsf{b}_j \gg_{\varphi_{[j+1]}} \mathsf{u}_j$, (b) $\mathsf{u}_j \gg_{\varphi_{[j+1]}} \mathsf{b}_j$ and (c) $\mathsf{u}_j \Diamond_{\varphi_{[j+1]}} \mathsf{b}_j$, as depicted in Figure 3. For case (a), the bound $x_j \leq \mathsf{u}_j$ entails $x_j \leq \mathsf{b}_j$, therefore we need not replace u_j. The reverse holds for case (b), and u_j is replaced. However, for case (c), neither bound entails the other. We call this a *conflict*.

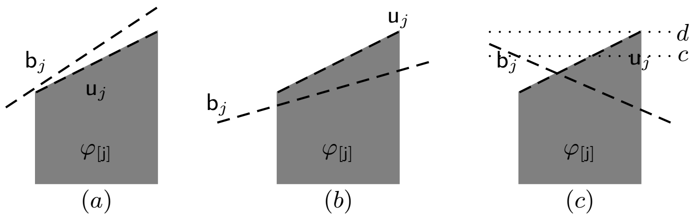

Fig. 3. Three cases encountered during abstraction. (a) $\mathsf{b}_j \gg_\varphi \mathsf{u}_j$, (b) $\mathsf{u}_j \gg_\varphi \mathsf{b}_j$ and (c) $\mathsf{u}_j \Diamond \mathsf{b}_j$ showing a conflict.

A *conflict* forces us to choose between two bounds $\mathsf{u}_j, \mathsf{b}_j$ where neither is semantically stronger than the other. Conflicts are due to the lack of a unique SRC abstraction. We handle conflicts using *conflict resolution heuristics* provided by the user. We describe a few possible heuristics below.

Interval Heuristic We consider the worst case interval bound on x_j resulting from either choice of bounds. Let $c = \max\ \mathsf{b}_j\ \mathsf{s.t.}\ \ \varphi_{[j+1]}$ and similarly, $d = \max\ \mathsf{u}_j\ \mathsf{s.t.}\ \ \varphi_{[j+1]}$. If $c < d$, we replace u_j by b_j, and retain u_j otherwise. Figure 3(c) shows a geometric interpretation.

Metric Choose the bound that minimizes the volume of the resulting SRC, or alternatively, the distance from a reference set.

LexOrder Choose syntactically according to lexicographic order.

Fixed Always choose to retain the original bound u_j, or replace it with b_j.

The result of abstraction is not guaranteed to be normalized. If there are no conflicts in the abstraction process then semantic equivalence of the SRC to the original polyhedron follows. In summary, the abstraction algorithm is parameterized by the conflict resolution heuristic. Our implementation uses the interval heuristic to resolve conflicts and the lexicographic order to break ties. Let α denote the abstraction function that uses some conflict resolution strategy.

Lemma 2. *For a constraint ψ, $\alpha(\psi)$ is a* SRC *and $\psi \models \alpha(\psi)$.*

Each inequality insertion requires us to solve finitely many optimization problems. Weak optimization requires time $O(n^2)$. Therefore, the SRC abstraction a polyhedron with m inequalities can be computed in time $O(n^2 m)$.

4 Domain Operations

The implementation of various operations required for static analysis over SRCs is discussed in this section.
Forced normalization. A SRC φ may fail to be normalized in the course of our analysis as a result of abstraction or other domain operations. Failure of normalization can itself be detected in $O(n^3)$ time using weak optimization using the lemma below:

Lemma 3. *A* SRC *φ is normalized iff for each bound $\mathsf{l}_i \le x_i \le \mathsf{u}_i$, $0 \le i < n$, $\varphi_{[i+1]} \models_W \mathsf{l}_i \le \mathsf{u}_i$. Note that the $\models_W$ relation is sufficient to test normalization.*

$\mathsf{Bottom\text{-}up}$: In general, a SRC that is not normalized may not have a normal equivalent. However, it is frequently the case that normalization may be achieved by simply propagating missing information from lower order indices up to the higher order indices. We consider each bound $\mathsf{l}_j \leq x_j \leq \mathsf{u}_j$, for $j = n-1, \ldots, 1$, and insert the implied inequality $\mathsf{l}_j \leq \mathsf{u}_j$ into $\varphi_{[\mathsf{j+1}]}$ using the abstraction procedure described in Section 3. This process does not always produce a normalized constraint. However, the procedure itself is useful since it can sometimes replace missing bounds for variables by using a bound implied by the remaining constraints.

Example 5. Recall the SRC $\varphi: \ x_2 + 4 \leq x_1 \leq 2x_3 + x_2 + 4 \ \wedge \ -x_3 \leq x_2 \leq x_3 + 4 \ \wedge \ -\infty \leq x_3 \leq 0$ from Example 1. The implied inequality $x_2 + 4 (\leq x_1) \leq 2x_3 + x_2 + 4$ simplifies to $x_3 \geq 0$. When inserted, this yields the normalized SRC φ' from Example 2.

Even though bottom-up normalization is not always guaranteed to succeed, it generally improves the result of the weak optimization algorithm. We therefore employ it after other domain operations as a pre-normalization step.

$\mathsf{Top\text{-}down}$: Add constant offsets $\alpha_j, \beta_j > 0$ to bounds $\mathsf{l}_j, \mathsf{u}_j$ such that the resulting bounds $\mathsf{l}_j - \alpha_j \leq x_j \leq \mathsf{u}_j + \beta_j$ are normalized. In practice, α_j, β_j may be computed by recursively normalizing $\varphi_{[\mathsf{j+1}]}$ and then using weak optimization. As a corollary of Lemma 3, top-down technique always normalizes.

Lemma 4. *Let φ be an* SRC *and φ_1, φ_2 be the results of applying bottom-up and top-down techniques, respectively to φ. It follows that $\varphi \models \varphi_1$ and $\varphi \models_W \varphi_2$. However, $\varphi \models_W \varphi_1$ does not always hold.*

Following other numerical domains, we note that normalization should never be forced after a widening operation to ensure termination [21].

Intersection & join. Given two SRCs $\varphi_1 \wedge \varphi_2$ their intersection can be performed by using the abstraction procedure, i.e., $\varphi_1 \sqcap \varphi_2 = \alpha(\varphi_1 \wedge \varphi_2)$. In general, the best possible join $\varphi_1 \sqcup \varphi_2$ for SRCs φ_1, φ_2 can be defined as the abstraction of the polyhedral convex hull φ_1, φ_2. However, convex hull computations are expensive, even for SRCs. We describe a direct generalization of the interval join used for value ranges. Let $\mathsf{l}_j \leq x_j \leq \mathsf{u}_j$ be a bound in φ_1 (similar analysis is used for bounds in φ_2). Consider the following optimization problems: $c_j^1 = \mathsf{min.} \ \ x_j - \mathsf{l}_j \ \mathsf{s.t.} \ \ \varphi_2, \ d_j^1 = \mathsf{max.} \ \ x_j - \mathsf{u}_j \ \mathsf{s.t.} \ \ \varphi_2$.

Note that $\varphi_2 \models \mathsf{l}_j + c_j^1 \leq x_j \leq \mathsf{u}_j + d_j^1$, while $\varphi_1 \models \mathsf{l}_j + 0 \leq x_j \leq \mathsf{u}_j + 0$. As a result, $(\varphi_1 \sqcup \varphi_2) \models \mathsf{l}_j + \min(c_j^1, 0) \leq x_j \leq \mathsf{u}_j + \max(0, d_j^1)$. We call such a constraint the *relaxation* of x_j in φ_1. Let φ_{12} be the result of relaxing each bound in φ_1 wrt φ_2. Similarly, let φ_{21} be obtained by relaxing each bound in φ_2 wrt φ_1. We define the range join as $\varphi_1 \sqcup_{\mathrm{R}} \varphi_2 : \ \varphi_{12} \sqcap \varphi_{21}$.

Lemma 5. *Given any* SRC *φ_1, φ_2, $\varphi_i \models_W \varphi_1 \sqcup_{\mathrm{R}} \varphi_2, i = 1, 2$. Also, $\varphi_1 \sqcap \varphi_2 \models \varphi_i$. However, this containment may not be provable using $\models_W$.*

Relaxing each constraint requires $O(n)$ optimization, each requiring $O(n^2)$ time. Finally, abstraction itself requires $O(n^3)$ time. As a result join can be achieved in time $O(n^3)$.

Example 6. Consider the SRCs φ_1, φ_2 shown below:

$$\varphi_1 : \left\{ \begin{array}{l} x_2 \leq x_1 \leq 2x_2 + 4 \\ x_3 \leq x_2 \leq 5 \\ -4 \leq x_3 \leq 4 \end{array} \right\} \quad \varphi_2 : \left\{ \begin{array}{r} -\infty \leq x_1 \leq x_2 \\ 0 \leq x_2 \leq x_3 + 1 \\ 0 \leq x_3 \leq 2 \end{array} \right\}$$

The relaxed constraints are given by

$$\varphi_{12} : \left\{ \begin{array}{r} -\infty \leq x_1 \leq 2x_2 + 4 \\ x_3 - 2 \leq x_2 \leq 5 \\ -4 \leq x_3 \leq 4 \end{array} \right\} \quad \varphi_{21} : \left\{ \begin{array}{r} -\infty \leq x_1 \leq x_2 + 9 \\ -4 \leq x_2 \leq x_3 + 9 \\ -4 \leq x_3 \leq 4 \end{array} \right\}$$

The join is computed by intersecting these constraints:

$$\varphi : \ -\infty \leq x_1 \leq 2x_2 + 4 \ \wedge \ x_3 - 2 \leq x_2 \leq 5 \ \wedge \ -4 \leq x_3 \leq 4 \,.$$

Projection. Projection is an important primitive for implementing the transfer function across assignments and modeling scope in interprocedural analysis. The "best" projection is, in general, the abstraction of the projection carried out over polyhedra. However, like convex hull, polyhedral projection is an exponential time operation in the worst case.

Definition 5 (Polarity). *A variable z occurring in the RHS of a bound $x_j \bowtie \mathsf{b}_j$ has* positive polarity *if b_j is a lower bound and z has a positive coefficient, or b_j is an upper bound and z has a negative coefficient. The variable has negative polarity otherwise. Variable z with positive polarity in a constraint is written z^+, and negative polarity as z^- (see Example 7 below).*

Direct projection. Consider the projection of x_j from SRC φ. Let $\mathsf{l}_j \leq x_j \leq \mathsf{u}_j$ denote the bounds for the variable x_j in φ. For an occurrence of x_j in a bound inequality of the form $x_i \bowtie \mathsf{b}_i : \ \boldsymbol{c}^T\boldsymbol{x} + d$ (note $i < j$ by triangulation), we replace x_j in this expression by one of $\mathsf{l}_j, \mathsf{u}_j$ based on the *polarity replacement rule*: occurrences of x_j^+ are replaced by the lower bound l_j, and x_j^- are by u_j. Finally, x_j and its bounds are removed from the constraint. Direct projection can be computed in time $O(n^2)$.

Lemma 6. *Let φ' be the result of a simple projection of x_j from φ. It follows that φ' is an SRC and $(\exists x_j)\ \varphi \models \varphi'$.*

Example 7. Direct projection of z from $\varphi : \ z^+ \leq x \leq z^- + 1 \ \wedge \ z^+ - 2 \leq y \leq z^- + 3 \ \wedge \ -\infty \leq z \leq 5$, replaces z^+ with $-\infty$ and z^- with 5 at each occurrence, yielding $\varphi' : \ -\infty \leq x \leq 6 \ \wedge \ -\infty \leq y \leq 8$.

Indirect projection. Direct projection can be improved by using a simple modification of *Fourier-Motzkin* elimination technique.

A *matching pair* for the variable x_j consists of two occurrences of variable x_j with opposite polarities in bounds $x_i \bowtie \alpha_j x_j^+ + \mathsf{e}_i$ and $x_k \bowtie \alpha_j x_j^- + \mathsf{e}_k$ with $i \neq k$. The matching pairs for the SRC φ from Example 7 are:

$$\varphi : \left\{ z^+ \leq x \leq z^- + 1 \ \wedge \ z^+ - 2 \leq y \leq z^- + 3 \ \wedge \ -\infty \leq z \leq 5 \right\}$$

There are two matching pairs for the variable z shown using arrows. The matching pair $z^+ \leq x$ and $y \leq z^- + 3$ can be used to rewrite the former constraint as $y - 3 \leq x$. Similarly the other matching pair can be used to rewrite the upper bound of x to $x \leq y + 2$. An indirect projection of the constraint in Example 7, using matching pairs yields the result $y - 3 \leq x \leq y + 3 \;\wedge\; -\infty \leq y \leq 8$.

Matching pairs can be used to improve over direct projection, especially when the existing bounds for the variables to be projected may lead to too coarse an over-approximation. They are sound and preserve the triangular structure.

Substitution. The substitution $x_j \mapsto \mathsf{e}$ involves the replacement of every occurrence of x_j in the constraint by e. In general, the result of carrying out the replacements is not a SRC. However, the abstraction algorithm can be used to reconstruct a SRC as $\varphi' :\ \alpha(\varphi[x \mapsto \mathsf{e}])$.

Transfer function. Consider a SRC φ and an assignment $x_j \ := \ \mathsf{e}$, where $e \equiv \boldsymbol{c}^T\boldsymbol{x} + d$. The assignment is *invertible* if $c_j \neq 0$, on the other hand the assignment is non-invertible or *destructive* if $c_j = 0$. An invertible assignment can be handled using a substitution $\psi :\ \ \varphi[x_j \mapsto \frac{1}{c_j}(x_j - (\mathsf{e} - c_j x_j))]$. A destructive update is handled by first using the projection algorithm to compute $\varphi' :\ \exists x_j\ \varphi$ and then computing the intersection $\psi :\ \alpha(\varphi' \wedge x_j = e)$ using the abstraction algorithm.

Widening. An instance of widening consists of two SRCs φ_1, φ_2 such that $\varphi_1 \models \varphi_2$. Using standard widening [9], we simply drop each constraint in φ_1 that is not entailed by φ_2. Let $x_j \leq \mathsf{u}_j$ be an upper bound in φ_1. We first compute $c_j = \mathsf{max.}\ \ (x_j - \mathsf{u}_j)\ \mathsf{s.t.}\ \ \varphi_2$. If $c_j > 0$ then $\varphi_2 \not\models_W x_j \leq \mathsf{u}_j$. Therefore, we need to drop the constraint. This may be done by replacing the bound u_j with ∞. A better widening operator is obtained by first replacing each occurrence of x_j^- (x_j occurring with negative polarity) by a matching pair before replacing u_j. Lower bounds such as $x_j \geq \mathsf{l}_j$ are handled symmetrically.

Lemma 7. *The* SRC *widening* ∇_R *satisfies (a)* $\varphi_1, \varphi_2 \models_W \varphi_1 \nabla_R \varphi_2$*; (b) any ascending chain eventually converges (even if* $\models_W$ *is used to detect convergence), i.e., for any sequence* $\psi_1, \ldots, \psi_n, \ldots$*, the widened sequence* $\varphi_1, \ldots,$ *satisfies* $\varphi_{N+1} \models_W \varphi_N$*, for some* $N > 0$.

Narrowing. The SRC narrowing is similar to the interval narrowing on Cousot et al. [10]. Let $\varphi_2 \models \varphi_1$. The narrowing $\varphi_1 \triangle_{\mathrm{R}} \varphi_2$ is given by replacing every $\pm\infty$ bound in φ_1 by the corresponding bound in φ_2.

Lemma 8. *For any* SRC*s* φ_1 *and* φ_2*, s.t.* $\varphi_2 \models \varphi_1$*,* $\varphi_1 \triangle_{\mathrm{R}} \varphi_2 \models_W \varphi_1$*. Furthermore, the narrowing iteration for* SRC *domain converges.*

Equalities. While equalities can be captured in the SRC domain itself, it is beneficial to compute the equality constraints separately. An equality constraint can be stored as $A\boldsymbol{x} + \boldsymbol{b} = 0$ where A is a $n \times n$ matrix. In practice, we store A in its triangulated form assuming some ordering on the variables. Therefore, it is possible to construct the product domain of SRC and linear equalities wherein both domains share the same variable ordering. The equality part is propagated using Karr's analysis [19].

Using the same variable ordering allows us to share information between the two domains. For instance, $\pm\infty$ bounds for the SRC component can be replaced with bounds inferred from the equality constraints during the course of the analysis. The equality invariants can also be used to delay widening. Following the polyhedral widening operator of Bagnara et al., we do not apply widening if the equality part has decreased in rank during the iteration [1].

Variable Ordering

We now consider the choice of the variable ordering. The variable ordering used in the analysis has a considerable impact on its precision. The ideal choice of a variable ordering requires us to assign the higher indices to variables which are likely to be unbounded, or have constant bounds. Secondly, if a variable x is defined in terms of y in the program flow, it is more natural to express the bounds of x in terms of y than the other way around. We therefore consider two factors in choosing a variable ordering: (a) ordering based on variable type or its purpose in the code; and (b) ordering based on variable dependencies.

The determination of the "type" or "purpose" of a variable is made using syntactic templates. For instance, variables used as loop counters, or array indices are assigned lower indices than loop bounds or those that track array/pointer lengths. Similarly, variables used as arguments to functions have higher indices than local variables inside functions. These variables are identified in the front end during CFG construction using a simple variable dependency analysis.

Variables of a similar type are ordered using data dependencies. A dataflow analysis is used to track dependencies among a variable. If the dependency information between two variables is always uni-directional we use this information to determine a variable ordering. Finally, variables which cannot be otherwise ordered in a principled way are ordered randomly.

5 Implementation

We have implemented an analysis tool to prove array accesses safe as part of the ongoing F-SOFT project [18]. Our analyzer is targeted towards proving numerous runtime safety properties of C programs including array and pointer access checks. The analyzer is *context sensitive*, by using call strings to track contexts. While recursive functions cannot be handled directly, they may be abstracted by unrolling to some fixed length and handling the remaining calls context insensitively. Our abstract interpreter supports a combination of different numerical domains, including constant folding, interval, octagon, polyhedron and SRC domains. For our experiments, we used off-the-shelf implementations of the octagon abstract domain library [20], and the Parma Polyhedron Library [2]. Each library was used with the same abstract interpreter to carry out the program analysis.

The tool constructs a CFG representation from the program, which is simplified using program slicing [29], constant propagation, and optionally by interval analysis. A linearization abstraction converts operations such as multiplication

and integer division into non-deterministic choices. Arrays and pointers are modeled by their allocated sizes while array contents are abstracted away. Pointer aliasing is modeled soundly using a flow insensitive alias analysis.
Variable clustering. The analysis model size is reduced by creating small *clusters* of related variables. For each cluster, statements that involve variables not belonging to the current cluster are abstracted away. The analysis is performed on these abstractions. A property is considered proved only if it can be proved in each context by some cluster abstraction. Clusters are detected heuristically by a backward traversal of the CFG, collecting the variables that occur in the same expressions or conditions. The backward traversal is stopped as soon as the number of variables in a cluster first exceeds 20 variables for our experiments. The number of clusters ranges from a few hundreds to nearly 2000 clusters.
Iteration Strategy. The fixpoint computation is performed by means of an upward iteration using widening to converge to some fixed point followed by a downward iteration using narrowing to improve the fixed point until no more improvements are possible. To improve the initial fixed point, the onset of widening is delayed by a fixed number of iterations (2 iterations for our experiments). The iteration strategy used is *semi-naive*. At each step, we minimize the number of applications of post conditions by keeping track of nodes whose abstract state changed in the previous iteration. In the case of the polyhedral domain, the narrowing phase is cut off after a fixed number of iteration to avoid potential non termination.

6 Experiments

Our experiments involved the verification of C programs for runtime errors such as buffer overflows, null pointer accesses, and string library usage checks. The domains are compared simply based on their ability to prove properties.
Small Benchmarks. We first compare the domains on a collection of small example programs [24]. These programs are written in the C language, and range from 20-400 lines of code. The examples typically consist of statically or dynamically allocated arrays accessed inside loops using aliased pointers, and passed as parameters to string/standard library functions.

Table 1 summarizes the results on these examples. The table on the left shows the total running times and the number of properties established. The properties proved by the domains are compared pairwise. The pairwise comparison summarizes the number of properties that each domain could (not) prove as compared to other domains. In general, the SRC domain comes out slightly ahead in terms of proofs, while remaining competitive in terms of time. An analysis of the failed proofs revealed that roughly 25 are due to actual bugs (mostly unintentional) in the programs, while the remaining were mostly due to modeling limitations.
Comparison of Implementation Choices. Our implementation of SRCs requires heuristics for optimization, variable ordering and conflict resolution while abstracting. Table 2 compares the proofs and running times for some alternative strategies for these operations. Each experiment in the table changes one option at a time, leaving the others unchanged. The choices we made for these strategies

Table 1. Comparison results on small examples. **Prog.**: Number of programs, **#Prp.**: total number of properties, **Prf**: number of proofs, **T**: time taken in seconds. Detailed pairwise comparison of the proofs is shown on the right.

Prog.	#Prp.	Int.		Oct.		Poly.		SRCs	
		Prf	T	Prf	T	Prf	T	Prf	T
48	480	316	9	340	29	356	413	360	81

	vs. Int.	vs. Oct.	vs. SRC
Oct.	$+29/-5$		
SRC	$+45/-1$	$+23/-3$	
Poly	$+46/-6$	$+24/-8$	$+7/-11$

Table 2. Comparison of different implementation choices for SRC. **SIMP**: simplex instead of weak optimization, **Random**: Random var. ordering, **Reverse**: reversal of the implemented var. ordering, **Random**: Resolve conflicts randomly, **Lex:** choose expr. with lower lex order, **Arg1**: Always retain, **Arg2**: Always replace existing expr.

opt.		Var. Ordering				Conflict Resolution							
SIMP		Random		Reverse		Random		Lex		Arg1		Arg2	
Prf	T	Prf	T	Prf	T	Prf	T	Prf	T	Prf	T	Prf	T
0/0	906	+4/-23	114	+1/-29	132	+1/-13	80	+2/-4	81	+6/-8	80	+6/-8	68

perform better than the more ad-hoc strategies used in these experiments. In particular, the difference is most pronounced when the variable ordering used is exactly the reverse of that suggested by our heuristic.

Network Controller Study. We studied the performance of our analyzer on a commercial network controller implementation. The analysis is started at different root functions assuming an unknown calling environment. Root functions are chosen based on their position in the global call graph. Each analysis run first simplifies the model using slicing, constant folding and interval analysis. Table 3 shows each of these functions along with the number of properties sliced away as a result of all the front-end simplifications. Also note that a large fraction of the properties can be handled simply by using interval analysis and constant folding. Slicing the CFG to remove these properties triggers a large reduction in the CFG size.

Table 4 compares the performance of the SRC domain with the octagon and polyhedral domains on the CFG simplified by slicing, constant folding and intervals. The interval domain captures many of the easy properties including the common case of static arrays accessed in loops with known bounds. While the SRC and octagon domains can complete on all the examples even in the absence of such simplifications, running interval analysis as a pre-processing step nevertheless lets us focus on those properties for which domains such as octagons, SRC and polyhedra are really needed. In many situations, the domains produce a similar bottom line. Nevertheless, there are cases where SRCs capture proofs missed by octagons and polyhedra. The SRC domain takes roughly 2.5× more time than the octagon domain. On the other hand, the polyhedral domain proves much fewer properties than both octagons and SRCs in this experiment, while requiring significantly more time. We believe that the iteration strategy used,

Table 3. Front end statistics for network controller. **#BB**: number of basic blocks, **#Fun**: number of functions, **#BC**: $\sum_{\text{block } n} \#Contexts(n)$, # of CFG blocks weighted by the # of contexts for each block , **#Prop**: number of properties, **Proof**: number of proofs by constant folding + intervals, **Time**: simplification time (sec), **#BC_Simpl**: Block-contexts after simplification.

Name	KLOC	#BB $\times 10^3$	#Fun	#BC $\times 10^3$	#Prop	Simplifications Proof	Time	#BC_Simpl
F1	5.9	1.6	11	2.0	441	208	24	1.6
F2	6.4	1.7	9	2.2	545	223	77	1.9
F3	7.2	2.1	11	2.6	613	424	58	1.5
F4	9.4	3.3	12	4.8	995	859	128	1.6
F5	11.3	3.8	16	4.5	1133	644	268	3.2
F6	15.0	5.3	15	10.0	1611	1427	451	2.1
F7	14.5	2.1	5	2.5	733	354	30	1.5
F8	25.7	9.0	5	29.6	2675	2641	1266	2.4
F9	23.0	8.1	8	11.9	2461	2391	1350	2.0
F10	45.4	16.6	59	60.6	4671	4627	2h30m	6.6
10	164				15878	13798	12850	

Table 4. Comparing the performance of abstract domains on simplified CFG. P_{Oct}: number of octagon proofs, T_{Oct}: octagon analysis time (seconds), P_{SRC}, T_{SRC}: SRC proof and time taken, P_{Poly}, T_{Poly}: polyhedron time and proof.

Function	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	Tot
P_{Oct}	56	0	23	56	146	56	28	14	0	0	379
T_{Oct}	11	30.7	9	3.2	105	7.7	10.4	1.3	.9	.4	180
P_{SRC}	56	**12**	22	56	146	56	28	14	0	**14**	**404**
T_{SRC}	18.2	59.1	21.0	7.6	291.7	17	20.7	0.7	1.5	0.5	439
P_{Poly}	42	0	23	0	62	0	0	0	0	0	127
T_{Poly}	63	684	75	29	1697	63.5	51.1	2.7	4.2	1.4	2672

especially the fast onset of widening and the narrowing cutoff for polyhedra may account for the discrepancy. On the other hand, increasing either parameter only serve to slow the analysis down further. In general, precise widening operators [1] along with techniques such as *lookahed widening* [16], *landmark-based widening* [26] or widening with *acceleration* [15] can compensate for the lack of a good polyhedral narrowing.

7 Conclusion

We have presented an abstract domain using symbolic ranges that captures many properties that are missed by other domains such as octagons and intervals. At the same time, our domain does not incur the large time complexity of the polyhedral domain. In practice, we hope to use the SRC domain in conjunction with intervals, octagons and polyhedra to prove more properties with a reasonable time overhead.

Many interesting avenues of future research suggest themselves. One interesting possibility is to allow for a conjunction of many SRC constraints, each using a different variable ordering. Apart from checking overflows, the SRC domain may also be useful for analyzing the numerical stability of floating point loops [17]. The constraint handling techniques presented in this paper can be directly applied to practical tools such as ARCHER [31] and ESP [12].

Acknowledgments. We gratefully acknowledge Ilya Shlyakhter for his useful insights and the anonymous reviewers for their comments. We acknowledge the efforts of Antoine Miné for his Octagon library and the Parma Polyhedra library team [2] for making our experiments possible.

References

1. Bagnara, R., Hill, P. M., Ricci, E., and Zaffanella, E. Precise widening operators for convex polyhedra. In *Static Analysis Symposium* (2003), vol. 2694 of *LNCS*, Springer–Verlag, pp. 337–354.
2. 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.
3. 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.
4. Blume, W., and Eigenmann, R. Symbolic range propagation. In *Proceedings of the 9th International Parallel Processing Symposium* (April 1995).
5. Chvátal, V. *Linear Programming*. Freeman, 1983.
6. Clarisó, R., and Cortadella, J. The octahedron abstract domain. In *Static Analysis Symposium* (2004), vol. 3148 of *LNCS*, Springer–Verlag, pp. 312–327.
7. Costan, A., Gaubert, S., Goubault, E., Martel, M., and Putot, S. A policy iteration algorithm for computing fixed points in static analysis of programs. In *CAV* (2005), vol. 3576 of *LNCS*, Springer–Verlag, pp. 462–475.
8. Cousot, P., and Cousot, R. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming* (1976), Dunod, pp. 106–130.
9. 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.
10. Cousot, P., and Cousot, R. Comparing the Galois connection and widening/narrowing approaches to Abstract interpretation. In *PLILP* (1992), vol. 631 of *LNCS*, Springer–Verlag, pp. 269–295.
11. Cousot, P., and Halbwachs, N. Automatic discovery of linear restraints among the variables of a program. In *ACM POPL* (Jan. 1978), pp. 84–97.
12. 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.
13. 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.
14. Gawlitza, T., and Seidl, H. Precise fixpoint computation through strategy iteration. In *Proc. of European Symp. on Programming (ESOP)* (2007), vol. 4421 of *LNCS*, pp. 284–289.

15. Gonnord, L., and Halbwachs, N. Combining widening and acceleration in linear relation analysis. In *SAS* (2006), vol. 4134 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 144–160.
16. Gopan, D., and Reps, T. W. Lookahead widening. In *Computer Aided Verification: Proceedings of the 18th International Conference* (Seattle, Washington, USA, 2006), T. Ball and R. B. Jones, Eds., vol. 4144 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 452–466.
17. Goubault, E., and Putot, S. Static analysis of numerical algorithms. In *SAS* (2006), vol. 4134 of *LNCS*, pp. 18–34.
18. Ivančić, F., Shlyakhter, I., Gupta, A., Ganai, M. K., Kahlon, V., Wang, C., and Yang, Z. Model checking C programs using f-soft. In *ICCD* (2005), pp. 297–308.
19. Karr, M. Affine relationships among variables of a program. *Acta Inf. 6* (1976), 133–151.
20. Miné, A. Octagon abstract domain library. `http://www.di.ens.fr/~mine/oct/`.
21. Miné, A. A new numerical abstract domain based on difference-bound matrices. In *PADO II* (May 2001), vol. 2053 of *LNCS*, Springer–Verlag, pp. 155–172.
22. Nielson, F., Nielson, H. R., and Hankin, C. *Principles of Program Analysis.* Springer–Verlag, 1999.
23. 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.
24. Sankaranarayanan, S. NEC C language static analysis benchmarks. Available by request from `srirams@nec-labs.com`.
25. Sankaranarayanan, S., Colón, M., Sipma, H. B., and Manna, Z. Efficient strongly relational polyhedral analysis. In *VMCAI* (2006), LNCS, Springer–Verlag.
26. Simon, A., and King, A. Widening Polyhedra with Landmarks. In *Fourth Asian Symposium on Programming Languages and Systems* (November 2006), vol. 4279 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 166–182.
27. Simon, A., King, A., and Howe, J. M. Two Variables per Linear Inequality as an Abstract Domain. In *Proceedings of Logic Based Program Development and Transformation (LBPDT)* (2002), vol. 2664 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 71–89.
28. Su, Z., and Wagner, D. A class of polynomially solvable range constraints for interval analysis without widenings. *Theor. Comput. Sci. 345*, 1 (2005), 122–138.
29. Tip, F. A survey of program slicing techniques. *J. Progr. Lang. 3*, 3 (1995).
30. 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.
31. Xie, Y., Chou, A., and Engler, D. ARCHER: Using symbolic, path-sensitive analysis to detect memory access errors. *SIGSOFT Softw. Eng. Notes 28*, 5 (2003).
32. Zaks, A., Cadambi, S., Shlyakhter, I., Ivančić, F., Ganai, M. K., Gupta, A., and Ashar, P. Range analysis for software verification. In *Proc. Workshop on Software Validation and Verification (SVV)* (2006).