The Gauge Domain: Scalable Analysis of Linear Inequality Invariants

Arnaud J. Venet

Carnegie Mellon University NASA Ames Research Center
Moffett Field, CA 94035
arnaud.j.venet@nasa.gov

Abstract. The inference of linear inequality invariants among variables of a program plays an important role in static analysis. The polyhedral abstract domain introduced by Cousot and Halbwachs in 1978 provides an elegant and precise solution to this problem. However, the computational complexity of higher-dimensional convex hull algorithms makes it impractical for real-size programs. In the past decade, much attention has been devoted to finding efficient alternatives by trading expressiveness for performance. However, polynomial-time algorithms are still too costly to use for large-scale programs, whereas the full expressive power of general linear inequalities is required in many practical cases. In this paper, we introduce the gauge domain, which enables the efficient inference of general linear inequality invariants within loops. The idea behind this domain consists of breaking down an invariant into a set of linear relations between each program variable and all loop counters in scope. Using this abstraction, the complexity of domain operations is no larger than $O(kn)$, where n is the number of variables and k is the maximum depth of loop nests. We demonstrate the effectiveness of this domain on a real 144K LOC intelligent flight control system, which implements advanced adaptive avionics.

1 Introduction

The discovery of numerical relationships among integer variables within a loop is one of the most fundamental tasks in formal software verification. Without this piece of information it would be impossible, for example, to analyze pointer arithmetic as it appears in real C programs. A fully automated solution based on convex polyhedra has been proposed by Cousot and Halbwachs [11] in what probably remains the most spectacular application of Abstract Interpretation. The polyhedral abstraction is precise enough to infer the exact invariants for most program loops in practice. It is based on the double description method [4, 21], which requires enumerating all faces of a convex polyhedron in all dimensions, an operation that has exponential time complexity in the worst case. Unfortunately, the combinatorial explosion almost always occurs in practice and this analysis cannot be reasonably applied to codes involving more than 15 or so variables.

Attempts have been made to improve the performance of the polyhedral domain. They essentially consist in finding more tractable albeit less precise

P. Madhusudan and S.A. Seshia (Eds.): CAV 2012, LNCS 7358, pp. 139–154, 2012.

```
p = &msg;
for (i = 0; i < n; i++) {
  if(*p == ...) {
    ...
    p += 16;
  } else {
    ...
    p += 32;
  }
}
```

Convex polyhedra:

$$\begin{cases} 0 \leq \mathtt{i} \leq \mathtt{n} - 1 \\ 16\mathtt{i} \leq \mathtt{p} \leq 32\mathtt{i} \end{cases}$$

Gauges:

$$\begin{cases} \lambda \leq \mathtt{i} \leq \lambda \\ 16\lambda \leq \mathtt{p} \leq 32\lambda \end{cases}$$

Additional properties:

$$\begin{cases} \lambda \leq \mathtt{n} - 1 \\ \lambda \in [0, +\infty] \end{cases}$$

Fig. 1. Loop invariant expressed with convex polyhedra and gauges

alternatives to those domain operations that may exhibit exponential complexity (join, projection) without modifying the expressiveness of the domain itself [22, 26]. Linear programming techniques are used instead of the double-description method to compute approximate versions of operations on polyhedra. The idea is that the Simplex algorithm exhibits better runtime performance in practice, although still exponential in the worst case. However, available experimental data make it difficult to predict how these techniques would scale to real applications.

Another and more popular approach consists in identifying a subclass of convex polyhedra that possess better algorithmic properties. Notable domains include template polyhedra [24], octahedra [5], subpolyhedra [15], simplices [25], symbolic ranges [23] and the family of two-variables per inequality domains [17–20, 27]. Two members of the latter class, difference-bound matrices [18] and octagons [19], are particularly important since, to the best of our knowledge, they are the only general-purpose relational abstract domains that have been applied to the verification of large applications [1, 3, 10, 28].

Among relational domains that can express inequalities, octagons and difference-bound matrices have the lowest computational complexity: quadratic in space and cubic in time in the worst case. However, due to the nature of the closure algorithm employed to normalize their representation, the worst-case complexity is always attained in practice, which makes this kind of domain unusable for codes with more than a few dozen variables [28]. In order to address this issue, it is necessary to break down the set of program variables into small groups on which the abstract domain can be applied independently. This variable packing can be performed statically before analysis using knowledge on the application [10], or at analysis time, for example, by using dependency information computed on the fly [28].

However, the limited expressiveness of weakly relational domains precludes the direct analysis of pointer arithmetic, which requires more general forms of inequality constraints. This issue is addressed in C Global Surveyor [28] by using templates for access paths in data structures. The parameters appearing in the

template make up for the lack of expressive power of difference-bound matrices. Although effective, these techniques substantially complicate the construction of a static analyzer and they are very dependent on the characteristics of the code analyzed.

In our experience with analyzing large NASA codes, we have observed that most of the time, the value of a scalar variable inside a loop nest was entirely determined by the control structure in terms of symbolic bounds of the form $a_0 + a_1\lambda_1 + \cdots + a_k\lambda_k$, where $\lambda_1, \ldots, \lambda_k$ denote loop counters and $a_1, \ldots, a_k$ are integer coefficients. In this paper, we present an abstract interpretation framework in which each variable is approximated by a pair of such symbolic bounds, which we call a *gauge*. This abstraction generates far fewer constraints than weakly relational domains while providing greater expressiveness.

In Fig. 1 we have shown a code snippet that reads variable-sized data from a buffer of bytes, a common pattern in embedded programs. Gauges represent the implicit loop invariants, which are hard to infer, but do not say anything about loop bounds. The abstraction shall therefore be complemented with additional abstractions, like intervals and symbolic constants. The main idea is that it is far more efficient to combine simpler abstractions rather than have a powerful but inefficient domain take care of all properties at once. The gauge domain is not intended as a replacement for convex polyhedra or weakly relational domains, as it has limitations. However, it provides a simple and efficient way of generating precise loop invariants for a large swath of code without the need for customizing the static analyzer.

The paper is organized as follows. In Sect. 2, we formally define the gauge abstraction and state some of its basic properties. Section 3 introduces the Abstract Interpretation framework in which our analysis is specified. In Sect. 4, we construct an abstract domain that can infer gauge invariants on programs. Section 5 reports experimental results on a large NASA flight system. Section 6 concludes the paper.

2 The Gauge Abstraction

We now give a formal construction of gauges and characterize their natural ordering. Let $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$ be a fixed set of positive counters. Given integer coefficients $a_0, \ldots, a_n$, we call *gauge bound* the expression $a_0 + \sum_{i=1}^{n} a_i\lambda_i$. Given a gauge bound g, we define the upper gauge $\overline{g}$ as

$$\overline{g} = \{(x, l_1, \ldots, l_n) \in \mathbb{Z} \times (\mathbb{Z}^+)^n \mid x \leq a_0 + \sum_{i=1}^{n} a_i l_i\}$$

We define the *lower gauge* $\underline{g}$ dually. Now, given two gauge bounds $g = a_0 + \sum_{i=1}^{n} a_i\lambda_i$ and $g' = a'_0 + \sum_{i=1}^{n} a'_i\lambda_i$, we would like to characterize the inclusion of upper gauges $\overline{g} \subseteq \overline{g}'$. This is equivalent to say that the following system has no integral solution:

$$S : \begin{cases} \lambda_i \geq 0 \quad i \in \{1, \ldots, n\} \\ x \leq a_0 + a_1\lambda_1 + \cdots + a_n\lambda_n \\ x \geq 1 + a'_0 + a'_1\lambda_1 + \cdots + a'_n\lambda_n \end{cases}$$

First, observe that if $a'_0 < a_0$ then S has a trivial solution $\langle x = a'_0 + 1, \lambda_1 = 0, \ldots, \lambda_n = 0\rangle$. Assume for now that $a'_0 \geq a_0$ and S admits a rational solution $\langle x = u, \lambda_1 = l_1, \ldots, \lambda_n = l_n\rangle$ such that all l_i are positive. There exists a nonzero positive integer μ such that $\mu u, \mu l_1, \ldots, \mu l_n$ are all integers (take the lowest common multiplier of all denominators for example). We deduce from S the following inequalities:

$$\begin{cases} \mu u \leq \mu a_0 + a_1(\mu l_1) + \cdots + a_n(\mu l_n) \\ \mu u \geq \mu(1 + a'_0) + a'_1(\mu l_1) + \cdots + a'_n(\mu l_n) \end{cases}$$

which can be rewritten as:

$$\begin{cases} \mu u - (\mu - 1)a_0 \leq a_0 + a_1(\mu l_1) + \cdots + a_n(\mu l_n) \\ \mu u - (\mu + (\mu - 1)a'_0) \geq a'_0 + a'_1(\mu l_1) + \cdots + a'_n(\mu l_n) \end{cases}$$

From $a'_0 \geq a_0$ and $\mu \geq 1$ we deduce that

$$\mu + (\mu - 1)a'_0 > (\mu - 1)a_0$$

and then $\mu u - (\mu - 1)a_0 > \mu u - (\mu + (\mu - 1)a'_0)$. Therefore, the variable assignment

$$\langle x \mapsto \mu u - (\mu - 1)a_0, \lambda_1 \mapsto \mu l_1, \ldots, \lambda_n \mapsto \mu l_n\rangle$$

is a solution of S. We just proved that if S admits a rational solution, then it also admits an integral solution. Therefore, S has no integral solution if and only if it has no rational solution. We can now reason entirely over rationals, which allows us to use a fundamental result of convex geometry, the Farkas lemma [29]:

Theorem 1 (Farkas Lemma). *Let $A \in \mathbb{Q}^{m\times d}$ and a column vector $\mathbf{z} \in \mathbb{Q}^m$. **Either** there exists a point $\mathbf{x} \in \mathbb{Q}^d$ with $A\mathbf{x} \leq \mathbf{z}$, **or** there exists a non-null row vector $\mathbf{c} \in (\mathbb{Q}^+)^m$, such that $\mathbf{c}A = \mathbf{0}$ and $\mathbf{cz} < \mathbf{0}$.*

Note that, although originally established for real numbers, the Farkas lemma can be proven using only elementary linear algebra [12] and therefore holds on rationals. We define the matrix $A \in \mathbb{Q}^{(n+2)\times(n+1)}$ as follows:

$$A = \begin{pmatrix} -1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & \ddots & \vdots & \vdots \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & -1 & 0 \\ -a_1 & \cdots & -a_{n-1} & -a_n & 1 \\ a'_1 & \cdots & a'_{n-1} & a'_n & -1 \end{pmatrix}$$

Let $\mathbf{x}$ be the $(n+1)$-column vector and $\mathbf{z}$ the $(n+2)$-column vector defined as:

$$\mathbf{x} = \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_n \\ x \end{pmatrix} \text{ and } \mathbf{z} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ a_0 \\ -a'_0 - 1 \end{pmatrix}$$

Then, the system S can be equivalently rewritten as:

$$A\mathbf{x} \leq \mathbf{z}$$

According to the Farkas lemma, S has no rational solution if and only if there exists a non-null $(n+2)$-row vector $\mathbf{c} = (c_1 \dots c_{n+2})$ of positive rationals such that $\mathbf{c}A = \mathbf{0}$ and $\mathbf{cz} < \mathbf{0}$. If we unfold the matrix expression, this is equivalent to:

$$\begin{cases} -c_1 - c_{n+1}a_1 + c_{n+2}a'_1 & = 0 \\ \vdots & \vdots \\ -c_n - c_{n+1}a_n + c_{n+2}a'_n & = 0 \\ c_{n+1} - c_{n+2} & = 0 \\ c_{n+1}a_0 - c_{n+2}(a'_0 + 1) & < 0 \end{cases}$$

If $c_{n+1} = 0$, then all c_i's are equal to zero, which contradicts the fact that $\mathbf{c}$ is non-null. Hence $c_{n+1} \neq 0$. Since $c_1, \dots, c_n$ each appear in exactly one equation, we can recast this condition in a much simpler form. The system $A\mathbf{x} \leq \mathbf{z}$ has no rational solution if and only if there exists a rational number $c > 0$ such that

$$\begin{cases} c(a'_1 - a_1) & \geq 0 \\ \vdots & \vdots \\ c(a'_n - a_n) & \geq 0 \\ c(a_0 - (a'_0 + 1)) & < 0 \end{cases}$$

Since $c > 0$, this is equivalent to the following condition

$$\forall i \in \{0, \dots, n\} \quad a_i \leq a'_i$$

We can establish a similar result on lower gauges by duality.

Theorem 2. *If $g = a_0 + \sum_{i=1}^{n} a_i\lambda_i$ and $g' = a'_0 + \sum_{i=1}^{n} a'_i\lambda_i$, then $\overline{g} \subseteq \overline{g}'$ (resp. $\underline{g} \subseteq \underline{g}'$) iff $\forall i \in \{0, \dots, n\}$ $a_i \leq a'_i$ (resp. $a_i \geq a'_i$).*

By analogy with intervals, we define a gauge as a pair $[g, g']$ of gauge bounds and its denotation as $\underline{g} \cap \overline{g}'$. Note that a gauge is not empty if and only if, for all positive values of $\lambda_1, \dots, \lambda_n$, there is an $x \in \mathbb{Z}$ such that

$$a_0 + a_1\lambda_1 + \dots + a_n\lambda_n \leq x \leq a'_0 + a'_1\lambda_1 + \dots + a'_n\lambda_n$$

This condition can be equivalenty restated as

$$a'_0 - a_0 + (a'_1 - a_1)\lambda_1 + \dots + (a'_n - a_n)\lambda_n \geq 0$$

for all positive values of $\lambda_1, \ldots, \lambda_n$. An elementary reasoning shows that this property holds if and only if $a_i \leq a'_i$ for all $i \in \{0, \ldots, n\}$, which is the exact analogue of the non-emptiness condition for intervals. We denote by $\bot_{\mathbf{G}}$ the empty gauge.

Now, given two non-empty gauges $G = [g_l, g_u]$ and $G' = [g'_l, g'_u]$, we need to characterize the inclusion of their denotation. Assume that $G \subseteq G'$. If $g_u = a_0 + \sum_{i=1}^{n} a_i \lambda_i$, then for all $(l_1, \ldots, l_n) \in (\mathbb{Z}^+)^n$, we have $(a_0 + \sum_{i=1}^{n} a_i l_i, l_1, \ldots, l_n) \in G$, because G is not empty. Since $G \subseteq G'$, we have $(a_0 + \sum_{i=1}^{n} a_i l_i, l_1, \ldots, l_n) \in \overline{g}'_u$. By definition of upper gauges, this entails $\overline{g}_u \subseteq \overline{g}'_u$. By duality, we also have $\underline{g}_l \subseteq \underline{g}'_l$. We just proved the following result:

Theorem 3. *Given two non-empty gauges*

$$G = [a_0 + \textstyle\sum_{i=1}^{n} a_i \lambda_i, b_0 + \sum_{i=1}^{n} b_i \lambda_i]$$
$$G' = [a'_0 + \textstyle\sum_{i=1}^{n} a'_i \lambda_i, b'_0 + \sum_{i=1}^{n} b'_i \lambda_i]$$

$G \subseteq G'$ iff $\forall i \in \{0, \ldots, n\}$ $a'_i \leq a_i \wedge b_i \leq b'_i$. The operation $G \sqcup G'$ defined as

$$[min(a_0, a'_0) + \sum_{i=1}^{n} min(a_i, a'_i)\lambda_i, max(b_0, b'_0) + \sum_{i=1}^{n} max(b_i, b'_i)\lambda_i]$$

is the least upper bound of G and G'.

It is quite intriguing that the natural order on gauges defined by the inclusion of denotations is the pointwise extension of the order on intervals. Gauges define a relational numerical domain that has the structure of a non-relational domain. This remarkable property is key to the scalability of the gauge abstraction.

Given a gauge bound g, we denote by $[g, +\infty]$ the upper gauge $\overline{g}$, by $[-\infty, g]$ the lower gauge $\underline{g}$, and by $[-\infty, +\infty]$ the trivial gauge $\mathbb{Z} \times (\mathbb{Z}^+)^n$. The order relation and the join operation defined above are readily extended to these generalized gauges, in the same way as is done for intervals. If we denote by $\mathbf{G}$ the set of all gauges, we have established that:

Theorem 4. *$(\mathbf{G}, \subseteq, \sqcup, [-\infty, +\infty])$ is a $\sqcup$-semilattice. The empty gauge $\bot_{\mathbf{G}}$ is the bottom element.*

Note that, in general, the intersection of two gauges is not a gauge and the greatest lower bound cannot be defined.

3 Abstract Interpretation Framework

We construct our static analysis in the theoretical framework of Abstract Interpretation [8, 9]. A program is represented as a control-flow graph and operates over a set of integer variables $\mathcal{X} = \{x, y, \ldots\}$ and a distinct set of integer non-negative counters $\Lambda = \{\lambda_1, \ldots, \lambda_n\}$. The control-flow graph is given by a set of nodes N, an initial node $start \in N$ and a transition relation $n \to n' : cmd$ labeled by commands. A command is either a sequence $s_1; \cdots; s_k$ of statements,

```
1: x = 0;
2: for (i = 0; i < 10; i++) {
3:   x += 2;
4: }
5: ...
```

$$\begin{cases} 1 \to 2 : x = 0; i = 0; \mathbf{new}(\lambda) \\ 2 \to 3 : i \leq 9 \\ 3 \to 2 : x = x + 2; i = i + 1; \mathbf{inc}(\lambda, 1) \\ 2 \to 4 : 10 \leq i \\ 4 \to 5 : \mathbf{forget}(\lambda) \end{cases}$$

Fig. 2. Representation of a simple C program in the language of the analyzer

a condition $e_1 \leq e_2$, where e_1, e_2 are either variables or integer constants, or a counter removal operation $\mathbf{forget}(\lambda)$, where $\lambda \in \Lambda$. The syntax of statements is defined as follows:

$$\begin{array}{rll} \mathbf{stmt} ::= & x = exp & x \in \mathcal{X} \\ | & \mathbf{new}(\lambda) & \lambda \in \Lambda \\ | & \mathbf{inc}(\lambda, k) & \lambda \in \Lambda, k \in \mathbb{Z}^+ \\ \mathbf{exp} ::= & c & c \in \mathbb{Z} \\ | & x & x \in \mathcal{X} \\ | & exp + exp & \\ | & exp - exp & \\ | & exp * exp & \end{array}$$

The concrete semantics is defined as a transition system on a set of states Σ. A state $\sigma \in \Sigma$ is a pair $\langle n, \varepsilon \rangle$, where n is a node of the control-flow graph and $\varepsilon \in \mathbb{Z}^{\mathcal{X}} \times (\mathbb{Z}^+)^{\Lambda}$ is an environment assigning values to variables in $\mathcal{X}$ and Λ. The semantics $[\![\_]\!]$ of statements and expressions is defined on environments as follows:

$$\begin{aligned} [\![x = e]\!]\varepsilon &= \varepsilon[x \mapsto [\![e]\!]\varepsilon] \\ [\![\mathbf{new}(\lambda)]\!]\varepsilon &= \varepsilon[\lambda \mapsto 0] \\ [\![\mathbf{inc}(\lambda, k)]\!]\varepsilon &= \varepsilon[\lambda \mapsto \varepsilon(\lambda) + k] \end{aligned}$$

The transition relation over states is defined as follows:

- If $n \to n' : s_1; \cdots; s_k$, then $\langle n, \varepsilon \rangle \to \langle n', [\![s_k]\!] \circ \cdots \circ [\![s_1]\!]\varepsilon \rangle$,
- If $n \to n' : x \leq y$ and $\varepsilon(x) \leq \varepsilon(y)$, then $\langle n, \varepsilon \rangle \to \langle n', \varepsilon \rangle$,
- If $n \to n' : x \leq c$ and $\varepsilon(x) \leq c$, then $\langle n, \varepsilon \rangle \to \langle n', \varepsilon \rangle$ (and similarly for a constant on the left-hand side of the condition),
- If $n \to n' : \mathbf{forget}(\lambda)$, then, for any $l \in \mathbb{Z}^+$, $\langle n, \varepsilon \rangle \to \langle n', \varepsilon[\lambda \mapsto l] \rangle$.

The last rule simply expresses that the value of a counter that is removed from scope can be any nonnegative integer. An initial state in the operational semantics is a pair $\langle start, \varepsilon \rangle$, where ε is any environment, as variables are assumed to be uninitialized at the beginning of the program. We denote by $\mathcal{I}$ the set of all initial states. Although simplified, this representation of programs is very close to the actual implementation of the analysis, which is based on LLVM [16].

In Fig. 2 we show how to translate a simple C program into our language. If the original program is structured, it is quite straightforward to introduce the

counter operations, as shown in the figure. In case the input program comes as a control-flow graph, we need to identify the loops and place the counters accordingly. This can be readily done using Bourdoncle's decomposition of a graph into a hierarchy of nested strongly connected components [2]. This efficient algorithm can be used to label each node of the control-flow graph with the sequence of nested strongly connected components in which it belongs. Using this information, loop counters can be assigned to each component and the counter operations can be automatically added to the relevant edges of the control-flow graph. The complexity of Bourdoncle's algorithm is $O(ke)$ where k is the maximum depth of loop nests and e is the number of edges in the control-flow graph.

We are interested in computing a sound approximation of the collecting semantics [6], i.e., the set of all states that are reachable from an initial state. Following the theory of Abstract Interpretation, the collecting semantics can be expressed as the least fixpoint of a semantic transformer $\mathbb{F}$. We denote by $\mathcal{E}$ the set $\mathbb{Z}^{\mathcal{X}} \times (\mathbb{Z}^+)^{\Lambda}$ of all environments. Then, the semantic transformer $\mathbb{F}$ is the function defined over $\wp(\mathcal{E})^N$ as follows:

$$\forall n \neq \mathit{start} \in N : \mathbb{F}(X)(n) = \{\varepsilon \in \mathcal{E} \mid \exists n' \in N, \exists \varepsilon' \in X(n') : \langle n', \varepsilon' \rangle \rightarrow \langle n, \varepsilon \rangle\}$$

with $\mathbb{F}(X)(\mathit{start}) = \mathcal{I}$. In order to obtain a computable approximation of the least fixpoint $\mathbf{lfp}\,\mathbb{F}$, we need to construct an abstract semantic specification [9], i.e.,

- An abstract domain $(D^\sharp, \sqsubseteq)$ together with a monotone concretization function $\gamma : (D^\sharp, \sqsubseteq) \rightarrow (\wp(\mathcal{E}), \subseteq)$,
- An abstract initial state $\mathcal{I}^\sharp \in D^\sharp$ such that $\mathcal{I} \subseteq \gamma(\mathcal{I}^\sharp)$,
- An abstract semantic transformer $\mathbb{F}^\sharp : (D^\sharp)^N \rightarrow (D^\sharp)^N$ such that $\mathbb{F} \circ \gamma \subseteq \gamma \circ \mathbb{F}^\sharp$,
- A widening operator $\nabla : D^\sharp \times D^\sharp \rightarrow D^\sharp$ such that, for any sequence $(x_i^\sharp)_{i \geq 0}$ of elements of $D^\sharp$, the sequence $(y_i^\sharp)_{i \geq 0}$ inductively defined as:

$$\begin{cases} y_0^\sharp = x_0^\sharp \\ y_{i+1}^\sharp = y_i^\sharp \nabla x_{i+1}^\sharp \end{cases}$$

  is ultimately stationary.

Then, it can be shown [9] that the sequence $(\mathbb{F}_i^\sharp)_{i \geq 0}$ iteratively defined as follows using the pointwise extension of ∇:

$$\begin{cases} \mathbb{F}_0^\sharp = \mathcal{I}^\sharp \\ \mathbb{F}_{i+1}^\sharp = \mathbb{F}_i^\sharp \text{ if } \mathbb{F}^\sharp(\mathbb{F}_i^\sharp) \sqsubseteq \mathbb{F}_i^\sharp \\ \qquad\quad = \mathbb{F}_i^\sharp \nabla \mathbb{F}^\sharp(\mathbb{F}_i^\sharp) \text{ otherwise} \end{cases}$$

is ultimately stationary and its limit is a sound approximation of $\mathbf{lfp}\,\mathbb{F}$.

4 The Gauge Domain

In this section we will construct an abstract semantic specification for the gauge abstraction. We cannot use the gauge semilattice $\mathbf{G}$ as is, because gauges are defined for all values of the counters, whereas in the first steps of the abstract iteration sequence only isolated counter values are computed. We need an operation similar to the higher-dimensional convex hull for convex polyhedra, which can build a convex approximation of a discrete set of points. In order to enable this type of induction, we need to keep track of constant counter values that are obtained at the very first steps of the abstract iteration sequence.

We denote by $(\mathbb{Z}_\top, \sqsubseteq)$ the semilattice of constants with greatest element $\top$. For $x, y \in \mathbb{Z}_\top$, $x \sqsubseteq y$ iff $y = \top$ or $x = y$. We define the domain of *sections* $\mathcal{S} = (\mathbb{Z}_\top^\Lambda, \sqsubseteq)$ ordered by pointwise extension of the order on $\mathbb{Z}_\top$. We denote by $\mathcal{E}^\sharp = (\mathbf{G}^\mathcal{X}, \sqsubseteq)$ the set of *abstract environments* ordered by pointwise inclusion. A *gauge section* is a pair $(\rho, \varepsilon^\sharp)$, where $\rho \in \mathcal{S}$ and $\varepsilon^\sharp \in \mathcal{E}^\sharp$, such that only counters in $\rho^{-1}(\top)$ may appear inside a gauge bound of $\varepsilon^\sharp$. The concretization $\gamma(\rho, \varepsilon^\sharp) \in \wp(\mathcal{E})$ of the gauge section is the set of all concrete environments $\varepsilon \in \mathcal{E}$ satisfying the following property:

$$\forall x \in \mathcal{X}, \exists (l_1, \ldots, l_n) \in (\mathbb{Z}^+)^\Lambda : (\varepsilon(x), l_1, \ldots, l_n) \in \varepsilon^\sharp(x)$$
$$\wedge\ \forall i \in \{1, \ldots, n\} : \rho(\lambda_i) \neq \top \Rightarrow l_i = \rho(\lambda_i) \wedge\ \forall i \in \{1, \ldots, n\} : \varepsilon(\lambda_i) = l_i$$

A gauge section is simply an abstract environment where the value of certain counters is set. Working on gauge sections instead of gauges will allow us to construct the invariants incrementally during the abstract iteration sequence. We denote by $(\mathbf{GS}, \sqsubseteq)$ the domain of gauge sections ordered by pointwise extension of the orders on $\mathcal{S}$ and $\mathcal{E}^\sharp$.

We can now construct an abstract semantic specification for the gauge abstraction. We could take $\mathbf{GS}$ as the abstract domain of our specification. However, this choice would yield poor results on nested loops with constant iteration bounds, a very common construct in flight systems and more generally in embedded applications. In order to keep a good level of precision, we need to maintain information on the ranges of the counters. We denote by $\mathbf{I}$ the standard lattice of intervals [7]. The abstract domain $D^\sharp$ is given by $\mathbf{GS} \times \mathbf{I}^\Lambda$ endowed with the pointwise extension of the underlying orders. The concretization $\gamma((\rho, \varepsilon^\sharp), \ell^\sharp)$ of an element of the product domain $D^\sharp$ is defined in the standard way as $\{\varepsilon \in \gamma(\rho, \varepsilon^\sharp) \mid \forall i \in \{1, \ldots, n\} : \varepsilon(\lambda_i) \in \ell^\sharp(\lambda_i)\}$ The abstract initial state $\mathcal{I}^\sharp$ is trivially given by the element of $D^\sharp$ in which all components are set either to $\top$ or to $[-\infty, +\infty]$.

The next thing we need to construct is a widening operator on $D^\sharp$, as it will be needed to define the abstract semantic function later on. We just need to define a widening on the domain of gauge sections, since the widening operator on $D^\sharp$ can be obtained by pointwise application of the widenings on the underlying domains. We first need some auxiliary operations. If $G = [a_0 + \sum_{i=1}^n a_i\lambda_i, b_0 + \sum_{i=1}^n b_i\lambda_i]$ is a gauge, $j \in \{1, \ldots, n\}$ and $l \in \mathbb{Z}^+$, we denote by $G[\lambda_j = l]$ the gauge

$$[a_0 + a_j l + \sum_{i \neq j} a_i\lambda_i, b_0 + b_j l + \sum_{i \neq j} b_i\lambda_i]$$

where we set the value of one counter. Let G and G' be two gauges defined as follows:

$$G = [a_0 + \sum_{i=1}^{n} a_i\lambda_i, b_0 + \sum_{i=1}^{n} b_i\lambda_i]$$
$$G' = [a'_0 + \sum_{i=1}^{n} a'_i\lambda_i, b'_0 + \sum_{i=1}^{n} b'_i\lambda_i]$$

Now, assume there is $\iota \in \{1, \ldots, n\}$ such that $a_\iota = b_\iota = a'_\iota = b'_\iota = 0$. Let $u, v \in \mathbb{Z}^+$ be two distinct non-negative integers. We want to construct a gauge, denoted by $G \nabla_{u,v}^{\lambda_\iota} G'$ such that

$$G[\lambda_\iota = u], G'[\lambda_\iota = v] \subseteq G \nabla_{u,v}^{\lambda_\iota} G'$$

This operation implements the basic induction step with respect to a counter. We have two gauges at two different values of the counter λ_ι and we want to extrapolate a gauge for all possible values of the counter. We choose a simple approach and perform a *linear interpolation*. We compute the slope $\alpha_\iota = \lfloor \frac{a'_0 - a_0}{v-u} \rfloor$ for the lower gauge (resp. $\beta_\iota = \lceil \frac{b'_0 - b_0}{v-u} \rceil$ for the upper gauge), taking care of rounding to the lower (resp. upper) nearest integer. This operation introduces new constants $\alpha_0 = a_0 - \alpha_\iota u$ and $\beta_0 = b_0 - \beta_\iota u$ into the gauge expression. There is no guarantee that the slopes and constants calculated from the upper (resp. lower) gauge will appear on their respective side, i.e., $\alpha_0 \leq \beta_0$ and $\alpha_\iota \leq \beta_\iota$. Therefore, we define $G \nabla_{u,v}^{\lambda_\iota} G'$ as the gauge $[c_0 + \sum_{i=1}^{n} c_i\lambda_i, d_0 + \sum_{i=1}^{n} d_i\lambda_i]$, where

- $c_0 = min(\alpha_0, \beta_0)$
- $d_0 = max(\alpha_0, \beta_0)$
- $c_\iota = min(\alpha_\iota, \beta_\iota)$
- $d_\iota = max(\alpha_\iota, \beta_\iota)$
- For $i \neq \iota$ and $i \neq 0$, $c_i = min(a_i, a'_i)$ and $d_i = max(b_i, b'_i)$

This elementary widening can be defined similarly when one bound of the gauges is $\pm\infty$. We need a variant of the previous operation when one of the gauges is defined over λ_ι. We keep the same notations and we now relax the assumptions, i.e., a'_ι and b'_ι may be nonzero, and $v = \top$. The gauge G' is already defined for all values of λ_ι. There is no need to change the slopes a'_ι and b'_ι, we simply need to adjust the constant coefficients. Hence, we set $\alpha_\iota = a'_\iota$ and $\beta_\iota = b'_\iota$, $\alpha_0 = a_0 - a'_\iota u$ and $\beta_0 = b_0 - b'_\iota u$. Using the previous notations, we define $G \nabla_{u,\top}^{\lambda_\iota} G'$ as the gauge $[c_0 + \sum_{i=1}^{n} c_i\lambda_i, d_0 + \sum_{i=1}^{n} d_i\lambda_i]$

We now construct an interval-like widening ∇_{I} on gauges, which extrapolates unstable bounds. If we denote by L the set $\{0, \ldots, n\}$, this widening is defined as follows:

$$G \nabla_{\mathrm{I}} G' = \begin{cases} G & \text{if } \forall i \in L : a_i \leq a'_i \wedge b'_i \leq b_i \\ [a_0 + \sum_{i=1}^{n} a_i\lambda_i, +\infty] & \text{if } \exists j \in L : b_j < b'_j \wedge \ \forall i \in L : a_i \leq a'_i \\ [-\infty, b_0 + \sum_{i=1}^{n} b_i\lambda_i] & \text{if } \exists j \in L : a'_j < a_j \wedge \ \forall i \in L : b'_i \leq b_i \\ [-\infty, +\infty] & \text{otherwise} \end{cases}$$

Similarly, given $I \subseteq \Lambda$, we define a partial join operation $\sqcup_I$ on gauges as follows: $G \sqcup_I G' = [min(a_0, a'_0) + \sum_{i=1}^{n} \underline{a}_i\lambda_i, max(b_0, b'_0) + \sum_{i=1}^{n} \overline{b}_i\lambda_i]$, where

$$\underline{a}_i = \begin{cases} min(a_i, a'_i) & \text{if } \lambda_i \in I \\ a_i & \text{otherwise} \end{cases} \quad \text{and} \quad \underline{b}_i = \begin{cases} max(b_i, b'_i) & \text{if } \lambda_i \in I \\ b_i & \text{otherwise} \end{cases}$$

The widening and partial join operations on gauges defined above can be extended pointwise to abstract environments in $\mathcal{E}^\sharp$. Now, let $(\rho_1, \varepsilon_1^\sharp)$ and $(\rho_2, \varepsilon_2^\sharp)$ be two gauge sections. Let $\Delta = \{\lambda'_1, \ldots, \lambda'_k\}$ be the set of counters on which the sections ρ_1 and ρ_2 disagree, and $A = \Lambda \backslash \Delta$ the set of counters on which they agree. If $\Delta \neq \emptyset$, we define the widening of the gauge sections as follows:

$$(\rho_1, \varepsilon_1^\sharp) \nabla (\rho_2, \varepsilon_2^\sharp) = \left(\rho_1 \sqcup \rho_2, \left(\cdots \left(\varepsilon_1^\sharp \nabla^{\lambda'_1}_{\rho_1(\lambda'_1), \rho_2(\lambda'_1)} \varepsilon_2^\sharp\right) \cdots \nabla^{\lambda'_k}_{\rho_1(\lambda'_k), \rho_2(\lambda'_k)} \varepsilon_2^\sharp\right) \sqcup_A \varepsilon_2^\sharp\right)$$

If $\Delta = \emptyset$, then $\rho_1 = \rho_2$, and we simply use the interval-like widening as follows:

$$(\rho_1, \varepsilon_1^\sharp) \nabla (\rho_2, \varepsilon_2^\sharp) = \left(\rho_1, \varepsilon_1^\sharp \nabla_{\mathrm{I}} \varepsilon_2^\sharp\right)$$

Note that the definition of the widening depends on the order in which the counters in Δ are arranged, as the linear interpolation widening defined above is commutative but not necessarily associative. In practice, for usual loop constructs, which are the main target of our analysis, the order in which the widening operations are performed has no effect on the result, but this may not always be the case. This is one limitation of our approach as compared to convex polyhedra and weakly relational domains.

We are now ready to define the abstract semantic function $\mathbb{F}^\sharp$. We first define the abstract semantics of expressions. Let G and G' be two gauges defined as follows:

$$G = [a_0 + \textstyle\sum_{i=1}^n a_i\lambda_i, b_0 + \sum_{i=1}^n b_i\lambda_i]$$
$$G' = [a'_0 + \textstyle\sum_{i=1}^n a'_i\lambda_i, b'_0 + \sum_{i=1}^n b'_i\lambda_i]$$

We define $G + G' = [(a_0 + a'_0) + \sum_{i=1}^n (a_i + a'_i)\lambda_i, (b_0 + b'_0) + \sum_{i=1}^n (b_i + b'_i)\lambda_i]$ and $G - G' = [(a_0 - b'_0) + \sum_{i=1}^n (a_i - b'_i)\lambda_i, (b_0 - a'_0) + \sum_{i=1}^n (b_i - a'_i)\lambda_i]$. Since the gauge abstraction is linear, we cannot compute the multiplication exactly. In pratice, multiplication mostly occurs in pointer arithmetic when scaling a byte offset to fit a type of a certain size. Hence, it is sufficient to consider the case when one of the gauges is a singleton, say $G' = [c, c]$. Then we define

$$G * G' = \left[ca_0 + \sum_{i=1}^n ca_i\lambda_i, cb_0 + \sum_{i=1}^n cb_i\lambda_i\right]$$

if c is positive, swapping the bounds when c is negative. Other cases when G is constant, both gauges are constant or one is zero are handled similarly. In all other cases we just return the trivial gauge $[-\infty, +\infty]$. For brevity, we did not go over the cases when one of the gauge bounds is infinite as they are handled similarly. The abstract semantics of expressions is readily defined from the previous operations on gauges.

Now, let $((\rho, \varepsilon^\sharp), \ell^\sharp)$ be an element of $D^\sharp$. We define the abstract semantics of statements as follows. In the case of an assignment operation, we have

$$[\![x = e]\!]^\sharp((\rho, \varepsilon^\sharp), \ell^\sharp) = ((\rho, \varepsilon^\sharp[x \mapsto [\![e]\!]^\sharp \varepsilon^\sharp]), \ell^\sharp)$$

For any counter λ, we denote by $\varepsilon^\sharp|_\lambda$ the abstract environment in which all occurences of a gauge where λ appears with a non-zero coefficient have been

replaced with $[-\infty, +\infty]$. Then, the abstract semantics of a $\mathbf{new}(\lambda)$ operation can be defined as follows:

$$[\![\mathbf{new}(\lambda)]\!]^\sharp((\rho, \varepsilon^\sharp), \ell^\sharp) = ((\rho[\lambda \mapsto 0], \varepsilon^\sharp|_\lambda), \ell^\sharp\,[\lambda \mapsto [0, 0]])$$

Given a gauge $G = [a_0 + \sum_{i=1}^n a_i\lambda_i, b_0 + \sum_{i=1}^n b_i\lambda_i]$, a counter λ_j and $k \in \mathbb{Z}^+$, we define the gauge $inc_{\lambda_j,k}(G)$ as follows:

$$\left[min(a_0 - ka_j, a_0 - kb_j) + \sum_{i=1}^n a_i\lambda_i, max(a_0 - ka_j, a_0 - kb_j) + \sum_{i=1}^n b_i\lambda_i\right]$$

This operation corresponds to incrementing a counter by a constant. The resulting constant coefficients may not satisfy the consistency condition for a non-empty gauge, whence the introduction of *min* and *max* operations. Cases where one of the gauge bounds is infinite are handled similarly. We can extend this operation pointwise to abstract environments. Thus, we can define the semantics of a $\mathbf{inc}(\lambda, k)$ operation as follows:

$$[\![\mathbf{inc}(\lambda, k)]\!]^\sharp((\rho, \varepsilon^\sharp), \ell^\sharp) = ((\rho[\lambda \mapsto \rho(\lambda) + k], inc_{\lambda,k}(\varepsilon^\sharp)), \ell^\sharp[\lambda \mapsto \ell^\sharp(\lambda) + k])$$

Note that for clarity we have overloaded the addition operator, but its semantics depends on the domain on which it applies.

It now remains to define the abstract semantics of commands. For a sequence of statements $s_1 \ldots s_n$, the abstract semantics is obviously given by $[\![s_n]\!]^\sharp \circ \cdots \circ [\![s_1]\!]^\sharp$. The abstract semantics of a condition $x \leq y$ is defined as follows. Assume that $a_0 + \sum_{i=1}^n a_i\lambda_i$ is the lower gauge bound of $\varepsilon^\sharp(x)$ and $b_0 + \sum_{i=1}^n b_i\lambda_i$ is the upper gauge bound of $\varepsilon^\sharp(y)$. We denote by C the linear inequality constraint $a_0 - b_0 + \sum_{i=1}^n (a_i - b_i)\lambda_i \leq 0$. Then we define

$$[\![x \leq y]\!]^\sharp((\rho, \varepsilon^\sharp), \ell^\sharp) = ((\rho, \varepsilon^\sharp), reduce_C(\ell^\sharp))$$

where $reduce_C(\ell^\sharp)$ is the reduction of a collection of variable ranges against a linear inequality constraint, using the algorithm defined in [13]. Since this algorithm is based on constraint propagation, we arbitrarily limit the number of propagation cycles performed (the threshold in our implementation is 5) so as to maintain an $O(|\Lambda|)$ complexity. No impact on precision has been observed in our experiments. The other types of conditions are handled similarly. Note that this operation only affects the loop counter bounds and does not change the gauge invariants.

Now, consider a gauge $G = [a_0 + \sum_{i=1}^n a_i\lambda_i, b_0 + \sum_{i=1}^n b_i\lambda_i]$, a counter λ_j and an interval $[l, u]$. We define the operation $coalesce_{\lambda_j,[l,u]}(G)$ as follows:

$$coalesce_{\lambda_j,[l,u]}(G) = \left[a_0 + a_j l + \sum_{i \neq j} a_i\lambda_i, b_0 + b_j u + \sum_{i \neq j} b_i\lambda_i\right]$$

We can extend this operation pointwise on abstract environments. Then, we can define the semantics of the $\mathbf{forget}(\lambda)$ operation as follows:

$$[\![\mathbf{forget}(\lambda)]\!]((\rho, \varepsilon^\sharp), \ell^\sharp) = ((\rho[\lambda \mapsto \top], coalesce_{\lambda,\ell^\sharp(\lambda)}(\varepsilon^\sharp)), \ell^\sharp[\lambda \mapsto [-\infty, +\infty]])$$

The **forget**(λ) operation is used when exiting the scope of a loop. If we did not inject the range information of the loop counter back into the gauge invariants, we would incur a major loss of accuracy when analyzing loops with constant iteration bounds. This points to a major limitation of the gauge abstraction: it only maintains precise loop invariants *inside* a loop, but most of this information is lost when exiting the loop. The polyhedral domain keeps relational information across loop boundaries and is more precise in this respect.

Finally, we define the abstract semantic transformer $\mathbb{F}^\sharp$ as follows

$$\forall n \neq start \in N : \mathbb{F}^\sharp(X)(n) = \nabla\{[\![cmd]\!]^\sharp(X(n')) \mid n' \to n : cmd\}$$

with $\mathbb{F}^\sharp(X)(start) = \mathcal{I}^\sharp$. Note that the widening operation is used to merge the invariants over a join node. We only need to use the interval-like widening ∇_{I} and the widening on $\mathbf{I}^\Lambda$ when it is the entry node of a strongly connected component, otherwise we can simply use the join operations, which provide better accuracy.

All elementary domain operations only depend on the number of active loop counters and the number of variables in the program. Using a sparse implementation of abstract environments, it is not difficult to see that all operations have an $O(km)$ time complexity in the worst case, where m is the number of program variables and k is the maximum depth of loop nests in the program. If we consider k as a constant, which is a realistic assumption in pratice, all operations are linear in the number of program variables. The gauge domain has a very low complexity in the worst case and is guaranteed to scale for large programs.

In order to illustrate how the abstract semantics operates, we unroll the first few steps of the abstract iteration sequence on the program shown in Fig. 2:

- Node 1: $((\{\}, \{\}), \{\})$
- Node 2:
$$\left(\left(\{\lambda \mapsto 0\}, \left\{\begin{array}{l} x \mapsto [0,0] \\ i \mapsto [0,0] \end{array}\right\}\right), \{\, \lambda \mapsto [0,0] \,\}\right)$$
- Node 3: The reduction operation has no effect on the invariant
$$\left(\left(\{\lambda \mapsto 0\}, \left\{\begin{array}{l} x \mapsto [0,0] \\ i \mapsto [0,0] \end{array}\right\}\right), \{\, \lambda \mapsto [0,0] \,\}\right)$$
- Node 2 through the back edge:
$$\left(\left(\{\lambda \mapsto 1\}, \left\{\begin{array}{l} x \mapsto [2,2] \\ i \mapsto [1,1] \end{array}\right\}\right), \{\, \lambda \mapsto [1,1] \,\}\right)$$
We perform the linear interpolation widening and we obtain:
$$\left(\left(\{\lambda \mapsto \top\}, \left\{\begin{array}{l} x \mapsto [2\lambda, 2\lambda] \\ i \mapsto [\lambda, \lambda] \end{array}\right\}\right), \{\, \lambda \mapsto [0, +\infty] \,\}\right)$$
This is the limit and convergence will be confirmed at the next iteration.
- Node 3: we perform the reduction operation on intervals and we obtain
$$\left(\left(\{\lambda \mapsto \top\}, \left\{\begin{array}{l} x \mapsto [2\lambda, 2\lambda] \\ i \mapsto [\lambda, \lambda] \end{array}\right\}\right), \{\, \lambda \mapsto [0, 9] \,\}\right)$$
The information on the loop bounds has been recovered thanks to the reduction operation.

Analysis	Analysis Time	Precision
Intervals + Complete Inlining	41 min	79%
Commercial Tool	5 hours	91%
Octagons	> 27 hours	N/A
Gauges	10 min 30 sec	91%

Fig. 3. Experimental results

5 Experimental Evaluation

We have implemented the gauge domain described in this paper in a buffer-overflow analyzer for C programs. The gauge domain is well suited for this kind of application, as it is good at discovering invariants that hold inside usual loop constructs. The buffer-overflow analyzer is implemented within an Abstract Interpretation framework developed at NASA Ames Research Center by the author and named IKOS (Inference Kernel for Open Static Analyzers). It is beyond the scope of this paper to describe the design of the buffer-overflow analyzer. We can just say that it is based on the LLVM front-end [16] and computes an abstract representation of objects and pointers in a C program. The analysis is modular and the effect of each function in memory is summarized by numerical constraints on array indices and pointer offsets that are affixed to the abstract memory graph. These numerical constraints are represented by gauges. Symbolic bounds (such as the size of an array passed as an argument to a function) are represented using an elementary domain of symbolic constants, which is used in combination with the gauge abstraction.

We have run the analyzer on a large flight system developed at NASA Dryden Flight Research Center and Ames Research Center. It consists of 144 KLOC of C and implements advanced adaptive avionics for intelligent flight control. It is a very pointer intensive application where matrix operations are pervasive. We have compared the performance of this analyzer with that of (1) a simple interval analysis running on a version of the program where function calls have been completely expanded using the LLVM inliner, (2) a leading commercial static analyzer based on Abstract Interpretation, and (3) a version of our analyzer in which octagons [19] have been subsituted for gauges. In the latter, we used Miné's implementation of the octagon domain from the APRON library [14]. The results of these experiments are presented in Fig. 3. All analyzers ran on a MacBook Air with a 1.86 Ghz Intel Core 2 Duo and 2 GB of memory, except the commercial tool, which is installed on a high-end server with 32 CPU cores and 64 GB of memory. The precision denotes the fraction of all array-bound operations which could be statically verified by the analyzer. This figure is not available for the version of our analyzer based on octagons, as we decided to kill the analysis process after allowing it to run continuously for over 27 hours.

6 Conclusion

We have constructed a numerical relational domain that is able to infer precise loop invariants and is guaranteed to scale thanks to tight bounds on the complexity of the domain operations. An experimental study led on a complex flight system developed at NASA showed that the gauge abstraction is able to deliver accurate loop invariants in a consistent way. This domain is not intended to be a replacement for more costly relational domains like convex polyhedra. It should be seen as a cheap numerical analysis that is able to discharge many simple verification properties, so that more powerful and computationally costly domains can be used to focus on a significantly smaller portion of the program.

Acknowledgement. We are extremely grateful to Tim Reyes for spending many hours getting the code through the commercial static analyzer.

References

1. Bouissou, O., Conquet, E., Cousot, P., Cousot, R., Feret, J., Ghorbal, K., Goubault, E., Lesens, D., Mauborgne, L., Miné, A., Putot, S., Rival, X., Turin, M.: Space software validation using abstract interpretation. In: Proc. of the International Space System Engineering Conference, Data Systems in Aerospace (DASIA 2009), pp. 1–7 (2009)
2. Bourdoncle, F.: Efficient Chaotic Iteration Strategies with Widenings. In: Pottosin, I.V., Bjorner, D., Broy, M. (eds.) FMP&TA 1993. LNCS, vol. 735, pp. 46–55. Springer, Heidelberg (1993)
3. Brat, G., Venet, A.: Precise and scalable static program analysis of NASA flight software. In: Proc. of the IEEE Aerospace Conference (2005)
4. Chernikova, N.V.: Algorithm for discovering the set of all the solutions of a linear programming problem. U.S.S.R. Computational Mathematics and Mathematical Physics 8(6), 282–293 (1968)
5. Clarisó, R., Cortadella, J.: The Octahedron Abstract Domain. In: Giacobazzi, R. (ed.) SAS 2004. LNCS, vol. 3148, pp. 312–327. Springer, Heidelberg (2004)
6. Cousot, P.: Semantic foundations of program analysis. In: Program Flow Analysis: Theory and Applications, ch. 10, pp. 303–342. Prentice-Hall (1981)
7. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: Proc. of the International Symposium on Programming (ISOP 1976), pp. 106–130 (1976)
8. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the Symposium on Principles of Programming Languages (POPL 1977), pp. 238–252 (1977)
9. Cousot, P., Cousot, R.: Abstract interpretation frameworks. Journal of Logic and Computation 2(4), 511–547 (1992)
10. Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: The ASTREÉ Analyzer. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 21–30. Springer, Heidelberg (2005)
11. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Proc. of the Symposium on Principles of Programming Languages (POPL 1978), pp. 84–97 (1978)

12. Dax, A.: An elementary proof of Farkas' lemma. SIAM Rev. 39(3), 503–507 (1997)
13. Harvey, W., Stuckey, P.: Improving linear constraint propagation by changing constraint representation. Constraints 8(2), 173–207 (2003)
14. Jeannet, B., Miné, A.: Apron: A Library of Numerical Abstract Domains for Static Analysis. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 661–667. Springer, Heidelberg (2009)
15. Laviron, V., Logozzo, F.: SubPolyhedra: A (More) Scalable Approach to Infer Linear Inequalities. In: Jones, N.D., Müller-Olm, M. (eds.) VMCAI 2009. LNCS, vol. 5403, pp. 229–244. Springer, Heidelberg (2009)
16. The LLVM Compiler Infrastructure, `http://llvm.org`
17. Logozzo, F., Fähndrich, M.: Pentagons: a weakly relational abstract domain for the efficient validation of array accesses. In: Proc. of the ACM Symposium on Applied Computing (SAC 2008), pp. 184–188 (2008)
18. Miné, A.: A New Numerical Abstract Domain Based on Difference-Bound Matrices. In: Danvy, O., Filinski, A. (eds.) PADO 2001. LNCS, vol. 2053, pp. 155–172. Springer, Heidelberg (2001)
19. Miné, A.: The octagon abstract domain. In: Proc. of the Workshop on Analysis, Slicing, and Transformation (AST 2001), pp. 310–319 (2001)
20. Miné, A.: A Few Graph-Based Relational Numerical Abstract Domains. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 117–132. Springer, Heidelberg (2002)
21. Motzkin, T.S., Raiffa, H., Thompson, G.L., Thrall, R.M.: The double description method. Annals of Mathematics Studies II(28), 51–73 (1953)
22. Sankaranarayanan, S., Colón, M.A., Sipma, H.B., Manna, Z.: Efficient Strongly Relational Polyhedral Analysis. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 111–125. Springer, Heidelberg (2005)
23. Sankaranarayanan, S., Ivančić, F., Gupta, A.: Program Analysis Using Symbolic Ranges. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 366–383. Springer, Heidelberg (2007)
24. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Scalable Analysis of Linear Systems Using Mathematical Programming. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 25–41. Springer, Heidelberg (2005)
25. Seidl, H., Flexeder, A., Petter, M.: Interprocedurally Analysing Linear Inequality Relations. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 284–299. Springer, Heidelberg (2007)
26. Simon, A., King, A.: Exploiting Sparsity in Polyhedral Analysis. In: Hankin, C., Siveroni, I. (eds.) SAS 2005. LNCS, vol. 3672, pp. 336–351. Springer, Heidelberg (2005)
27. Simon, A., King, A., Howe, J.M.: Two Variables per Linear Inequality as an Abstract Domain. In: Logic-Based Program Synthesis and Transformation, pp. 71–89 (2003)
28. Venet, A., Brat, G.P.: Precise and efficient static array bound checking for large embedded C programs. In: Proc. of the Conference on Programming Language Design and Implementation (PLDI 2004), pp. 231–242 (2004)
29. Ziegler, G.M.: Lectures on Polytopes. Graduate Texts in Mathematics. Springer (1995)