Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science 287 (2012) 29–40

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Access-Based Localization for Octagons

Eva Beckschulze, Stefan Kowalewski,[1,2]

Embedded Software Laboratory
RWTH Aachen University
Aachen, Germany

Jörg Brauer[3]

Verified Systems International GmbH
Bremen, Germany

Abstract

Access-based localization is a two-step process. First, the set of abstract memory locations that are accessed in a procedure are determined. Then, in a subsequent fixed point iteration, the input to the respective procedure is reduced to those variables that are indeed accessed, thereby saving time and memory. The topic of this paper is access-based localization for the octagon abstract domain. For the frequently occurring scenario that only one out of two variables in some octagonal constraint is contained in the access-set of a procedure, there is a variety of opportunities how localization could be implemented. This paper presents three different approaches on how to deal with such constraints. Albeit applied to a subset of the abstract state space, two of these approaches preserve precision, i.e., the abstract state space is as precise as in the case that no localization is performed.

Keywords: Access-based localization, octagon domain, interprocedural analysis

1 Introduction

The key idea in abstract interpretation [4] is to systematically abstract away from the complex, concrete semantics of a program. Given concrete and abstract domains C and D that describe concrete values and abstract descriptions, respectively, the key idea in abstract interpretation is to simulate the execution of each concrete operation $f : C \to C$ in a program using an abstract analogue $g : D \to D$. Since computing the corresponding abstract semantics amounts to fixed point iteration until the results stabilize.

[1] Supported by DFG EXC 89.

[2] Email: beckschulze, kowalewski@embedded.rwth-aachen.de

[3] Email: brauer@verified.de

doi:10.1016/j.entcs.2012.09.004

1.1 Efficiency Issues in Fixed-Point Iteration

Efficiency is a great concern when designing fixed point computations in abstract interpretation. Of course, the chosen abstract domain strongly influences the number of iterations required to converge onto a fixed point. Yet, independently of the chosen abstract domain, poor efficiency often originates from two other factors:

- Since abstract operation g transforms an abstract state $d \in D$ on input into an abstract output $d' \in D$, each d' (or variations thereof) is propagated from each location in a program to its successors. If a program constitutes a large number of variables, each state is of considerable size, requiring large amounts of memory.
- To detect a fixed point, it is necessary to compare a newly computed state to the state obtained in the previous iteration. In the worst case, fixed-point detection thus amounts to comparing two entire states, which may be of significant size.

Recently, there has thus been increasing interest in *localizing* abstract interpretations, a technique that is based on the idea of restricting states to a subset that is sufficient to compute sound abstractions in a given context, based on the subset of states that is required *locally* in the respective context.

1.2 Improving Performance by Localization

Access-based localization [8] is a two step procedure: First, memory locations accessed in code blocks are estimated conservatively using an efficient flow-insensitive preanalysis. Second, the actual analysis is run restricting input states of code blocks to the determined sets of accessed memory locations. Propagating smaller states improves performance of fixed-point computation w.r.t. the issuses discussed above. Additionally, smaller input states lower the number of necessary iterations as a reanalysis of a procedure is unnecessary if only those parts of the abstract state were changed that are not accessed in the procedure.

In [8], access-based localization was applied to the non-relational interval domain. With quadratic memory cost per abstract element and cubic time cost for some operations efficency is also a great concern in the octagon domain [7]. Compared to intervals, however, the relational nature of octagons require different techniques to avoid a loss of precision. In this paper, we discuss three different approaches to access-based localization for octagons, two of which do not incur a loss in precision.

1.3 Contributions & Outline

This paper specifically provides the following contributions to access-based localization for the octagon abstract domain:

- We discuss three different strategies to determine those constraints that may be relevant within a context.
- We formalize these strategies within a single framework and provide an optimality criterion.
- We present an experimental evaluation that compares both, the runtime and the

```
1   int f(void){
2     if (...){
3       ...
4       g();
5     }
6     else{
7       ...
8       h();
9     }
10  }
11  void g(){
12    x++;
13  }
14
15
16  void h(){
17    x = 0;
18  }
```

Fig. 1. Example C program

precision of these approaches for a set of benchmarks written in C.

The presentation of these contributions is structured as follows. First, Sect. 2 introduces the different localization strategies by means of a worked example. Afterwards, in Sect. 3, we introduce the formal notion of *significance* for constraints in localization. This notion refines the classical access-based view on dependencies. The different localization strategies are then formalized in Sect. 4, before experimental evidence is presented in Sect. 5. Finally, the paper concludes with a survey of related work in Sect. 6 and a conclusion in Sect. 7.

2 Worked Example

The crucial difference between interval analysis compared to octagon analysis w.r.t. localization is the fact that the former propagates non-relational intervals, whereas the latter propagates constraints that relate a variable to another variable. To illustrate this important difference and its implications on localization, we discuss the example in Fig. 1. We assume a pre-analysis determines the sets of variables accessed in functions, denote Acc. Suppose this step yields the following sets:

$$\mathsf{Acc}(f) = \{x, y\}, \qquad \mathsf{Acc}(g) = \mathsf{Acc}(h) = \{x\}$$

It has been shown that an interval-based analysis of function `g` taking into account this form of access-information [8] needs to consider only `x` to converge onto a sound over-approximation. Likewise, it is sufficient to restrict the input state of `h` to the interval that describes `x`. Variable `y` is not accessed in these functions; its corresponding interval prior to `g` (resp. `f`) is thus identical to the intervals after analyzing `g` (resp. `h`).

2.1 Access-Sets and Octagons

Using octagons, states consists of conjunctions of constraints of the form $\pm v_1 \pm v_2 \leq c$, where v_1 and v_2 are program variables and $c \in \mathbb{Z}$ is a constant value. In the remainder, we assume all variables to be integral, though our results are likewise applicable to real-valued or rational octagons. To make the presentation accessible, we restrict the example to few constraints and assume $\pm v_1 \pm v_2 = \infty$ for all unspecified constraints. Especially, we do not derive all implicit constraints but deal with them separately in Sect. 3. Assume the input state of call site `g` (line 4 in Fig. 1) is represented by the

following octagon:

$$(\mathtt{x} \leq 4) \wedge (\mathtt{y} \leq 0) \wedge (\mathtt{x} + \mathtt{y} \leq 2)$$

Clearly, there is no need to transfer the bounds of y for the constraint $(\mathtt{y} \leq 0)$ to g as y is not accessed within g. By way of contrast, since $\mathtt{x} \in \mathsf{Acc}(\mathtt{g})$, we propagate $(\mathtt{x} \leq 4)$ into g. Hence, for the constraint $(\mathtt{x} \leq 4)$, the situation is no different from the interval-case since the octagon prescribes no relation between x and y. However, with $\mathtt{x} \in \mathsf{Acc}(\mathtt{g})$ and $y \notin \mathsf{Acc}(\mathtt{g})$, it is unclear how to proceed for constraints that involve both, x and y. Of course, there are different possible strategies to approach this situation, three of which we sketch in what follows.

2.2 Projecting Octagons onto Access-Sets

Using this strategy, only those octagonal constraints $\pm v_1 \pm v_2 \leq c$ that satisfy $\{v_1, v_2\} \subseteq \mathsf{Acc}(\mathtt{g})$ are propagated into g. Hence, since $\mathtt{y} \notin \mathsf{Acc}(\mathtt{g})$, the constraint $(\mathtt{x} + \mathtt{y} \leq 2)$ is dismissed. However, incrementing x in g necessitates updating all constraints that involve x. Yet, without the constraint $(\mathtt{x} + \mathtt{y} \leq 2)$ present at that position, the update of the relation between x and y is lost. The drive for soundness thus forces us to delete the constraint $(\mathtt{x} + \mathtt{y} \leq 2)$ after g has been analyzed, causing loss of information. Assuming the same input state for call site h, deleting the constraint would not result in loss of information since $(\mathtt{x} + \mathtt{y} \leq 2)$ is invalidated by the assignment `x = 0`. For g, the approach is exemplified in Fig. 2(a).

2.3 Passing Dependent Constraints

Now suppose all constraints that involve x — here $(\mathtt{x} + \mathtt{y} \leq 2)$ — are also transferred into g (cp. Fig. 2(b)). In this case, precision is maintained at the expense of a larger input state, and the analysis proceeds as if it were performed without localization. Indeed, both strategies discussed so far highlight unsatisfactory aspects of access-based localization for relational domains. With the first approach, all relational constraints that involve variables not contained in the access-sets of the respective function are dismissed, thereby reducing the precision of octagonal analysis to that of intervals. In contrast, transferring constraints that exhibit dependencies to accessed variables yields only minor reductions of the size of the input space, provided there are many such dependencies. A practical middle ground between both approaches can be found by introducing slack variables, which we discuss in what follows.

2.4 Introducing Slack Variables

In the following, assume that the octagonal state on input to g consists of $(\mathtt{x} \leq 4)$ paired with n additional constraints $\mathtt{x} + \mathtt{y}_i \leq c_i$ for $i \leq n \in \mathbb{N}$. Denote this octagon o. Observe that we still have $\mathsf{Acc}(\mathtt{g}) = \{\mathtt{x}\}$. For each $v \in \mathsf{Acc}(\mathtt{g})$, we introduce a fresh *slack variable* v_{slack}, replace v in o by v_{slack}, and augment the octagon with $(v_{\mathsf{slack}} - v \leq 0) \wedge (v - v_{\mathsf{slack}} \leq 0)$, being equivalent to $v = v_{\mathsf{slack}}$. We thus obtain a

transformed octagon o_{slack} defined as:

$$o_{\mathsf{slack}} = (\mathtt{x}_{\mathsf{slack}} \leq 4) \wedge \textstyle\bigwedge_{i=1}^{n} (\mathtt{x}_{\mathsf{slack}} + \mathtt{y}_i \leq c_i) \wedge (\mathtt{x} - \mathtt{x}_{\mathsf{slack}} \leq 0) \wedge (\mathtt{x}_{\mathsf{slack}} - \mathtt{x} \leq 0)$$

At this point, we propagate o_{slack} reduced onto constraints that involve $\mathsf{Acc}(g)$ and the corresponding slack variables into $\mathtt{g}$. The key idea of this construction is to propagate only those variables into a function that are included in its access set, but use slack variable to keep track of changes to these variables. Hence, the octagon propagated into $\mathtt{g}$ ranges over $2 \cdot |\mathsf{Acc}(\mathtt{g})|$ variables. Since $|\mathsf{Acc}(g)|$ is typically small compared to the overall number of variables (and can be reduced further using techniques such as bypassing [10]), this approach reduces the size of the octagon significantly. Incrementing $\mathtt{x}$ in $\mathtt{g}$ then yields the modified octagon:

$$(\mathtt{x} \leq 5) \wedge (\mathtt{x} - \mathtt{x}_{\mathsf{slack}} \leq 1) \wedge (\mathtt{x}_{\mathsf{slack}} - \mathtt{x} \leq -1)$$

This octagon is the output of the analysis after $\mathtt{g}$ (cp. Fig 2(c)). Conjoining this result with o yields an octagon that implicitly relates $\mathtt{x}$ on output of $\mathtt{g}$ with $\mathtt{y}_1, \ldots, \mathtt{y}_n$ through $\mathtt{x}_{\mathsf{slack}}$. The remaining step is thus to eliminate $\mathtt{x}_{\mathsf{slack}}$ from the conjoined octagon using existential quantification, thereby making the implicit relations between $\mathtt{x}$ and $\mathtt{y}_1, \ldots, \mathtt{y}_n$ explicit, to give:

$$(\mathtt{x} \leq 5) \wedge \bigwedge_{i=1}^{n} (\mathtt{x} + \mathtt{y}_i \leq c_i + 1)$$

Elimination can be implemented as closure [3]. It is important to appreciate that this method also works if $\mathtt{x}$ is assigned a completely unrelated value, as in $\mathtt{h}$ from Fig. 1.

We call the approach presented first the *access-based approach* as the transferred input state is derived directly from access-information. Taking into account dependencies, we name the approach presented secondly the *dependency-based approach*. In the last approach, $\mathtt{x}_{\mathsf{slack}}$ is used as an anchor that relates $\mathtt{x}$ to its input value. Therefore, we shall refer to this approach as *anchoring*.

3 Dependency and Significance

In the last section, we assumed $\pm v_i \pm v_j = \infty$ for unspecified constraints. In practice, however, usually few constraints are unbounded as transfer functions require that all implicit constraints have been made explicit by closing the respective octagon. This is done using a modified Floyd-Warshall algorithm that checks iteratively for all constraints $v_j - v_k \leq c$ and $v_k - v_i \leq d$ whether $c + d$ is smaller than the upper bound previously computed for $v_j - v_i$ [7]. Indeed, it is well-known that closed octagons often contain significant amounts of redundant constraints (cp. Fig. 3(a)). Using dependency-based localization, all constraints that depend on a variable in the access-set of a called function are passed to the callee. Redundant constraints in the base octagon thus manifest themselves in redundant dependent constraints that

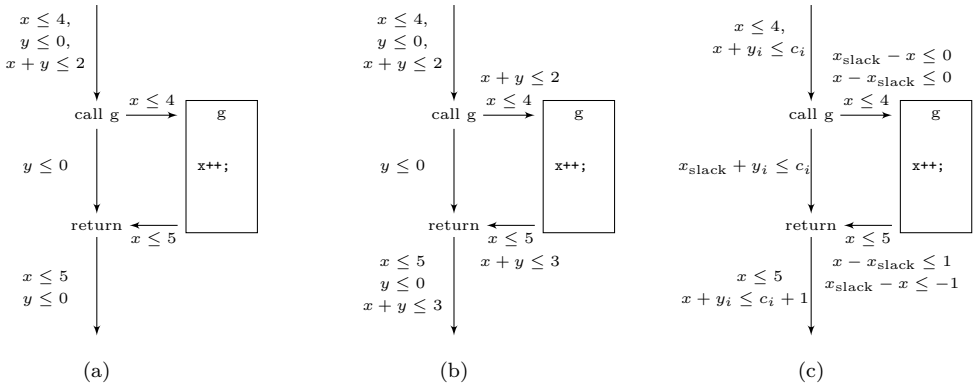


Fig. 2. Transferred constraints when calling g: (a) considering the accessed set only, (b) transferring also dependent constraints, (c) introducing a slack variable.

are passed to the callee, too, even though they may be insignificant for the output of the analysis. We thus refine the notion of *dependency* by means of *significance* so as to obtain reductions in the number of involved constraints.

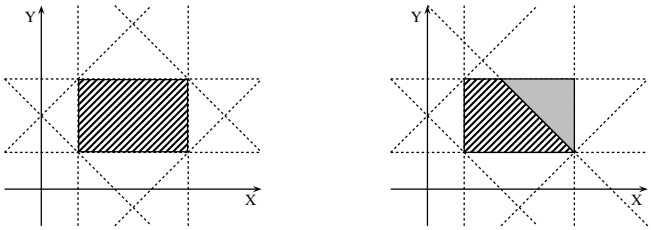
An algorithm that removes redundancy in difference bound matrices (DBMs) was presented in [6] and adapted in [1] to coherent DBMs as used in the octagon domain. However, in our case redundancy is obstructive only with respect to accessed variables. In order to determine whether a constraint is significant, it suffices to determine if a constraint is tighter than the constraint obtained by combining intervals. Intuitively, a constraint is significant if it cuts the rectangle defined by the intervals as shown in Fig. 3(b); here, $x + y$ is smaller than the sum of the upper bounds of x and y. In the following, let $V = \{v_1, \ldots, v_n\}$ denote the set of program variables. We interpret an octagon $o = \{\sigma_1, \ldots, \sigma_m\}$ as a collection of m linear constraints of the form $\sigma_i = (\lambda_{i,1} \cdot v_{i,1} + \lambda_{i,2} \cdot v_{i,2} \leq c_i)$, where $v_{i,1}, v_{i,2} \in V$, $\lambda_{i,1}, \lambda_{i,2} \in \{0, -1, 1\}$, and $c_i \in \mathbb{Z}$. Further, we assume o to be closed, and define:

Definition 3.1 (**Dependent Constraint**) Let $\mathsf{Acc}(\mathsf{g})$ denote the access-set of a callee g. Let o denote the octagon on input. Then, g depends on a constraint $\sigma_i \in o$ iff $\{v_{i,1}, v_{i,2}\} \cap \mathsf{Acc}(\mathsf{g}) \neq \emptyset$.

Definition 3.2 (**Significant Constraint**) For each $v \in V$, let v_l and v_u denote the boundaries of v as characterized by o. A constraint $\sigma_i \in o$ is called *significant* iff $\lambda_{i,1} \cdot \beta(v_{i,1}, \lambda_{i,1}) + \lambda_{i,2} \cdot \beta(v_{i,2}, \lambda_{i,2}) > c_i$. The map $\beta : (V \times \mathbb{Z}) \to \mathbb{Z}$ is defined as:

$$\beta(v, \lambda) = \begin{cases} v_l & : \lambda = -1 \\ v_u & : \lambda = 1 \end{cases}$$

From now on, we refer to the set of constraints in o that are significant for f as $\mathsf{Sig}_f(o)$. Without taking significance into account, dependency is a purely syntactic criterion to determine whether an octagonal constraint may be relevant in a callee. By considering significance as well, the constraints passed to a callee depend on the state of the octagonal abstraction. Intuitively, an octagonal constraint is propagated into a callee using the dependency-based strategy iff it is dependent *and* significant.



(a) (b)

Fig. 3. (a) Diagonal constraints do not offer more precision than the interval bounds. (b) The constraint $X + Y \leq c$ reduces solution space by the gray triangle.

4 Formal Model

In principle, analyzing a concrete procedure f subject to an octagonal input state o amounts to applying the abstract counterpart $f^{\#}$ of f to o. The output of the analysis is thus $f^{\#}(o) = o_{\mathsf{out}}$. Clearly, $f^{\#}$ can be designed as the composition of the abstract counterparts of all operations that constitute f. Localization refines this approach by applying $f^{\#}$ to a sub-state of o that is (often) strictly smaller than o. This section formally discusses our approaches to localization and provides arguments for correctness and optimality criteria in terms of maintaining precision.

4.1 Classifying Constraints

To detect all dependencies, we require that the octagonal input o to f is closed. As before, let $\mathsf{Acc}(f)$ be the set of variables directly or transitively accessed in f. Based on $\mathsf{Acc}(f)$, the input $o = \{\sigma_1, \ldots, \sigma_m\}$ to f is partitioned into three disjoint sets, depending on whether a constraint is relevant, unaffected, or can be discarded. Whereas the latter two sets are independent of the localization strategy, the definition of relevant constraints distinguishes the different methods of localization.

- $\mathsf{Relevant}_f(o)$ denotes the set of constraints in o relevant when evaluating f. Formal definitions for each method are given in subsequent paragraphs.
- $\mathsf{Unaffected}_f(o)$ consists of constraints in $o = \{\sigma_1, \ldots, \sigma_m\}$ that are unaffected by the call to f, and are therefore still valid at return from the callee, i.e., $\mathsf{Unaffected}_f(o) = \{\sigma \in o \mid \mathsf{vars}(\sigma) \cap \mathsf{Acc}(f) = \emptyset\}$.
- $\mathsf{Discard}_f(o)$ describes the remaining constraints in o that can be discarded as they are either redundant or will be outdated after return. Put formally, we thus obtain $\mathsf{Discard}_f(o) = \{\sigma \in o \setminus (\mathsf{Relevant}_f(o) \cup \mathsf{Unaffected}_f(o))\}$.

These sets of constraints partition o with respect to f, i.e., $o = \mathsf{Unaffected}_f(o) \cup \mathsf{Relevant}_f(o) \cup \mathsf{Discard}_f(o)$. Further, if $f^{\#}$ denotes the abstract transformer of f, then the output o_{out} of $f^{\#}$ subject to o is defined as (cp. Fig 4):

$$o_{\mathsf{out}} = \mathsf{Unaffected}_f(o) \cup f^{\#}(\mathsf{Relevant}_f(o))$$

We state the following result with respect to precision of localized analysis:

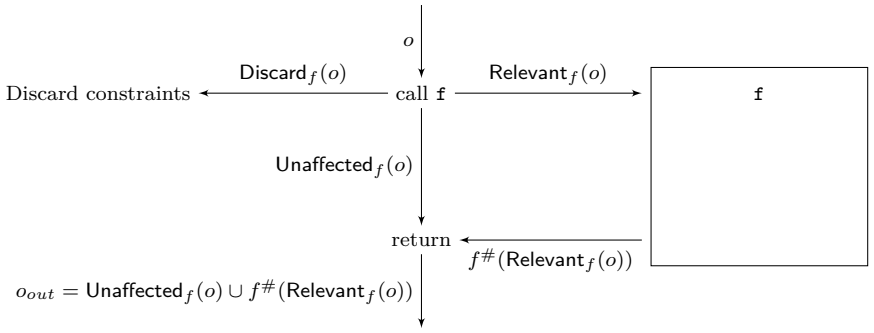


Fig. 4. Schematic illustration of localization applied to a call site given the octagon o on input

Proposition 4.1 *Whenever the closed forms of* $\mathsf{Unaffected}_f(o) \cup f^\#(\mathsf{Relevant}_f(o))$ *and* $f^\#(o)$ *coincide, localization maintains the precision of the original analysis.*

Within this setting, we now study the different localization methods formally, especially the definition of relevance.

4.2 Access-based Localization

Analyzing f using access-based localization requires all octagonal constraints that relate variables accessed in f to be propagated into f. We therefore define the set of relevant constraints as $\mathsf{Relevant}_f^{\mathsf{Acc}}(o) = \{\sigma \in o \mid \mathsf{vars}(\sigma) \subseteq \mathsf{Acc}(f)\}$ and obtain an immediate consequence regarding precision:

Corollary 4.2 *Assume* $\mathsf{Discard}_f(o) \cap \mathsf{Sig}_f(o) = \emptyset$. *Then:*

$$f^\#(o) = \mathsf{Unaffected}_f(o) \cup f^\#(\mathsf{Relevant}_f^{\mathsf{Acc}}(o))$$

Intuitively, access-based localization for octagons is optimal iff all variables having significant dependencies are redefined in f.

4.3 Dependency-based Localization

In order to avoid precision loss inherent to access-based localization, we consider significant constraints in the definition of relevance:

$$\mathsf{Relevant}_f^{\mathsf{Dep}}(o) = \{\sigma \in o \mid \mathsf{vars}(\sigma) \cap \mathsf{Acc}(f) \neq \emptyset \wedge \sigma \in \mathsf{Sig}_f(o)\}$$

This construction entails that only redundant constraints are discarded. Yet, these constraints can be re-established and updated respectively by closing the output of the analysis of f, and thus, precision is maintained.

Proposition 4.3 *Dependency-based localization is optimal.*

4.4 Localization using Anchors

For this approach, we first determine the set of variables for which copies (*anchors*) are introduced. To do so, define the set $\mathsf{sigVars}_f(o)$ of variables that appear in

significant constraints as follows:

$$\mathsf{sigVars}_f(o) = \bigcup_{\sigma \in \mathsf{Sig}_f(o)} \mathsf{vars}(\sigma)$$

Using the above set, we obtain the set of variables to be anchored by restricting $\mathsf{sigVars}_f(o)$ to those variables that are found in the access-set of f. For each variable $v \in \mathsf{sigVars}_f(o) \cap \mathsf{Acc}(f)$, we thus introduce a slack (or anchor) variable v_{slack} and an auxiliary constraint $v_{\mathsf{slack}} = v$ to give:

$$o_{\mathsf{slack}} = o \wedge \bigwedge_{v \in \mathsf{sigVars}_f(o) \cap \mathsf{Acc}(f)} ((v - v_{\mathsf{slack}} \leq 0) \wedge (v_{\mathsf{slack}} - v \leq 0))$$

To ensure that the same constraints are valid for v_{slack} and v, we compute the strong closure of the augmented system. Partitioning is applied to the enhanced input state:

$$\mathsf{Relevant}_f^{\mathsf{Anc}}(o_{\mathsf{slack}}) = \left\{ \sigma \in o_{\mathsf{slack}} \;\middle|\; \begin{array}{l} v_i, v_j \in \mathsf{vars}(\sigma) : v_i, v_j \in \mathsf{Acc}(f) \vee \\ (v_i \in \mathsf{Acc}(f) \wedge v_j \notin \bigcup_{\sigma \in o} \mathsf{vars}(\sigma)) \end{array} \right\}$$

We conclude the elaborations on different localization strategies by observing that the precision of anchor-based localization is identical to the precision of the dependency-based approach.

Proposition 4.4 *Anchor-based localization is optimal.*

5 Experiments

We have implemented the different localization strategies for octagons in Java. Our framework provides a flow- and context-sensitive abstract interpretation in the octagon domain. The benchmarks we analyzed consist of a number of programs which, e.g., implement sorting algorithms or perform integer arithmetic. Without localization, i.e., considering all variables at all points in program, our analyzer runs out of memory after a few minutes, consuming approximately 1GB of RAM.

The overall results with localization are given in Tab. 1. There, *Acc* refers to the access-based approach, *Dep* to the dependency-based approach, and *Anc* to anchoring. Beside runtime, we determined the number of different procedure contexts found during the analysis, the average sizes of DBMs used to represent octagonal constraints, the total number of constraints, and also the portion of significant constraints. Interestingly, the numbers of variables traced along procedure calls using *Dep* and *Anc* is only slightly larger compared to *Acc*. Yet, interprocedural analyses using *Dep* and *Anc* are more precise.

Analogously to results reported by Oh et al. [8], we observe that localization decreases the number of necessary re-analyses of procedures. This is because it is often the case that only constraints in the octagon change that are irrelevant for the analysis of the callee. The smaller number of different contexts analyzed

Method	Runtime	# contexts	∅ matrix size	#constraints	sign. constraints
Acc	26 s	137	18.71	567898	0.63 %
Dep	36 s	211	23,55	1031211	1.21 %
Anc	34 s	132	20.61	625459	0.67 %

Table 1
Results for localization

shows that the access-based localization and anchoring benefit from localized input states. By way of comparison, the dependency-based approach on top of dependent variables has larger input states, and procedures need to be traced significantly more often than using one of the other strategies. A noteworthy byproduct of our experiments is the overall small number of semantically significant constraints. This result confirms the observation of Bagnara et al. [1], who noted that large parts of (closed) octagonal systems are often redundant. Consequently, localization based on semantic dependencies leads to fewer constraints than a criterion that determines dependencies on a syntactic basis.

6 Related Work

Early approaches to localization mainly focused on reachability, which is a criterion much coarser than considering access-sets, with the expected drawbacks in terms of effectiveness. For a survey of related work in this field, we refer to [8] who give a thorough discussion in the context of introducing access-based localization. The authors [8] also extend localization for intervals from procedure boundaries to arbitrary code blocks. Adapting this approach is also possible for the octagon domain, using merely identical techniques.

Independently of the chosen numeric domain, the efficiency of access-based localization suffers from the fact that often global variables are carried along many nodes unnecessarily until they are finally accessed in a sub-routine, deep down in the call-tree. Most importantly, all global variables accessed in the program are considered in the input state of the main function. In [10], the notion of bypassing is introduced to mitigate this effect. The idea is to bypass the abstract values of variables to transitively called procedures instead of propagating them through the procedure bodies. This idea implemented for the interval domain can also be transferred to the octagon domain. However, due to the relational nature of octagons, this is technically more involved. At each call site, all implicit constraints need to be derived in order to propagate updated constraints and dependencies to callees.

Other related work can be found in work handling the large abstract states of complex abstract domains. Variable packing where only few variables can be related to each other is a way to make relational domains feasible with respect to memory consumption and analysis time [5]. For the octagon abstract domain an *octagon packing* [7], i.e., a collection of small packs of variables, is determined based on observable dependencies in the program. For example, the control variable in an `if` statement will be assigned to the pack that includes a variable accessed in the corresponding `then` block (and `else` block, respectively) as there is a direct dependency

between these variables. Octagon packing has been successfully integrated into static analyzers, thereby reducing memory consumption to a reasonable amount while still achieving satisfying precision [2]. Using localization to reduce the size of abstract states can be seen as a method orthogonal to octagon packing. Both approaches reduce memory consumption and computational cost. By way of comparison, our approach guarantees maximal precision, whereas octagon packing does not as not all constraints are derived.

In [11], Venet and Brat report that packs of variables determined statically based on syntactic criterion do no always provided satisfactory precision. They consider the problem of using DBMs for the analysis that involves pointer variables and array indices. The evaluation of a dereference is then based on a base address and an offset, the values of both of which are extracted from the abstract state. Their solution is to create and merge DBMs dynamically, during the analysis, whenever an operation causes new relations between variables associated with different DBMs. This way all relations are derived at the cost of reorganizing DBMs perpetually. Furthermore, the procedure does not include localization so that many constraints are unnecessarily propagated along many paths.

The sparse analysis framework presented in [9] applies a stricter form of localization that is based on (approximate) data dependencies (similar to def-use-chains) instead of control flow. The key to designing a sparse analysis lies in the appropriate choice of approximate definition- and use-sets for the particular abstract domain. For sparse relational analysis, definition- and use-sets consist of packs determined by a pre-analysis as described above. Identifying the two steps localization and size reduction, their procedure differs in the order of execution from our approach: Sparse relational analysis in [9] starts with determining an octagon packing that provides small octagons. Subsequently, localization in terms of data dependency is applied to packs. By way of contrast, we start with localizing variables and then reduce the size of the octagon on-the-fly, based on data dependencies. Although both approaches are heading for more efficiency they differ with respect to the acceptance of loss of information.

7 Conclusion

In this paper, we have revisited access-based localization and identified drawbacks of approaches that use either only access-sets or syntactic dependencies between variables. Even though access-based localization has proven precise for intervals, (weakly-) relational domains such as octagons impose different needs on the localization strategy to capture the exposed relations without propagating too many redundant constraints. We have thus provided a dependency-based technique, which is based on the notion of significant constraints. The anchoring approach is, in turn, based on the idea of augmenting the input state with auxiliary (slack or anchor) variables to track the differences between values on input and output of a procedure.

References

[1] Bagnara, R., P. Hill and E. Zaffanella, *Weakly-Relational Shapes for Numeric Abstractions: Improved Algorithms and Proofs of Correctness*, Formal Methods in System Design **35** (2009), pp. 279–323.

[2] Blanchet, B., P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux and X. Rival, *A Static Analyzer for Large Safety-Critical Software*, in: *PLDI* (2003), pp. 196–207.

[3] Bozga, M., C. Gîrlea and R. Iosif, *Iterating Octagons*, in: *TACAS*, LNCS **5505** (2009), pp. 337–351.

[4] Cousot, P. and R. Cousot, *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*, in: *POPL* (1977), pp. 238–252.

[5] Cousot, P., R. Cousot and L. Mauborgne, *A Scalable Segmented Decision Tree Abstract Domain*, in: *Time for Verification, Essays in Memory of Amir Pnueli* (2010), pp. 72–95.

[6] Larsen, K., F. Larsson, P. Pettersson and W. Yi, *Efficient Verification of Real-Time Systems: Compact Data Structure and State-Space Reduction*, in: *RTSS* (1997), pp. 14–24.

[7] Miné, A., *The Octagon Abstract Domain*, Higher-Order and Symbolic Computation **19** (2006), pp. 31–100.

[8] Oh, H., L. Brutschy and K. Yi, *Access Analysis-based Tight Localization of Abstract Memories*, in: *VMCAI*, LNCS **6538** (2011), pp. 356–370.

[9] Oh, H., K. Heo, W. Lee, W. Lee and K. Yi, *Design and Implementation of Sparse Global Analyses for C-like Languages*, in: *PLDI* (2012), to appear.

[10] Oh, H. and K. Yi, *Access-based Localization with Bypassing*, in: *APLAS*, LNCS **7078** (2011), pp. 50–65.

[11] Venet, A. and G. Brat, *Precise and efficient static array bound checking for large embedded C programs*, in: *PLDI* (2004), pp. 231–242.