A summary on the Octagon abstract domain

Simon Coumes
ENS Rennes with help from team CELTIC

March 9, 2019

Abstract

Static analysis is the study of computer programs without executing them. In this paper we will try to summarize the work of Antoine Miné on the Octagon abstract domain, a work in which he aims to prove invariants of the form $(\pm x + \pm y \leq c)$ for a variety of programs.

1. Introduction

Static analysis is the study of computer programs without executing them. To do so, a possibility is to use a process called abstract execution to prove invariants at differetn program points. In this paper we will try to summarize the work of Antoine Miné [1] on the Octagon abstract domain, a work in which he aims to prove invariants of the form $(\pm x \pm y \leq c)$ for a variety of programs.
Parts 2 and 3 will go back on a few notions useful in static analysis whereas parts 4, 5, and 6 will talk directly about Miné's work on the Octagon domain.
In part 2 we will explain what abstract domains are and introduces a few useful concepts and examples. In part 3 we will do the same for Difference-Bound Matrices (DBM). Parts 4 and 5 will introduce the Octagon domain proper and the operators used to apply that domain to real code. Lastly, part 6 will show an application on an example and discuss the state of implementations of the Octagon Domain at the time of the redaction of Miné's article [1].

As in Miné's article all proofs of theorems will be omitted in this paper. Proofs related to the Octagon Domain can be found in Antoine Miné's master thesis [3].

2. Static analysis and abstract domains

Because it will be useful in this section, we remind the definition of a lattice.

Definition 1 *A lattice is a partially ordered set $(E, \leq)$ where every pair of elements has a least upper bound and a greater lower bound.*

2.1 Execution on state properties

Let us consider a program on a set ν of variables, all taking their values in Γ. A state of the program is simply an element of $\nu \to \Gamma$. It corresponds to giving a value to each variable of ν. A state property is a limitation put on the set of possible states. Having a specific property at a certain point of the program's execution simply means that the current state can only be in a certain subset of $\nu \to \Gamma$.

Definition 2 *A state property is an element of $P(\nu \to \Gamma)$*

The idea of abstract interpretation is to run programs on state properties rather than states. Some information is lost in the process as state properties are less precise than states. An execution will rely on a specific set of properties and operators. The new operators introduced are there to mimic the basic operators of the program at hand on the chosen properties. The set of properties used for such an execution will be called the abstract domain.

Definition 3 *An abstract domain is a subset of $P(\nu \to \Gamma)$*

Exemple 1 *If the program uses only one variable x which takes its values in $\mathbb{Z}$ we might only need to know its sign. A suitable abstract domain might then be $\{+, -, 0, \top\}$ where $\top$ simply represents the property "both signs are possible".*

2.2 Abstract operators and least fixpoints

When analyzing a program we consider the set of all its program points. A program point is simply a specific point in the program for which we want the to know the possible states. We consider that such points are present :

- at the beginning and the end of the program,
- before and after a loop,
- after each time a variable can change its value,
- before and after an if,
- etc

Each time we reach a program point we might know of new possible states for that point. In term of properties that means we need an operator on our abstract domain to act as an "or". In term of set theory that means we need an operator that acts as $\cup$. Obviously, it might not be possible to find a perfect equivalent to $\cup$ in our abstract domain. We will then use an approximation, which we will write $\cup\#$. For similar reasons we will introduce $(\cup\#, \cap\#, \subseteq\#, \bot, \top)$ to act as $(\cup, \cap, \subseteq, \emptyset, Env)$, where Env simply means all possibilities. Because we want to be able to define $\cup\#$ and $\cap\#$, a good abstract domain should be a lattice for $\subseteq$.

Having defined those operators and possibly a few others related to the program and the abstract domain at hand, we can write our program as a set of equations on the properties at all program points.

We define $(X_0, X_1, ..., X_n)$ the properties at all different program points. We then write every portion of code separating two subsequent program points i and i+1 as an equation between X_i and X_{i+1}. Loops are slightly different but the general idea remains the same.

Theorem 1 *The least solution to that system of equations is the least fixpoint reached by abstract execution initialized at $X_0 = \top$ and $X_i = \bot$ for $i \neq 0$. With X_0 the first program point.*

2.3 Widenings

When our abstract domain is infinite the abstract execution might never finish. To ensure convergence we define a widening operator.

Definition 4 *A widening on an abstract domain L is an operator* $\triangle : L \times L \to L$ *such that :*

- $\forall (x, y) \in L^2, \quad x \cup y \subseteq x \triangle y$
- *for* $(x_i)_{i \in \mathbb{N}}$ *an increasing chain the chain defined by* $y_0 = x_0$ *and* $y_{n+1} = y_n \triangle x_{n+1}$ *stabilizes after a finite number of steps.*

By replacing $U\#$ by a widening in strategic places we can ensure our abstract execution will finish in a finite number of steps.

2.4 Examples

The following domains are abstractions for variables with values in $\mathbb{R}$.

We call interval abstract domain the domain obtained by approximating the value of each variable with an interval.

We call polyhedron abstract domain the domain of properties that are systems of linear equations. It is named the polyhedron abstract domain because a system of equations of n variables with bound solutions can be represented as a polyhedron in n dimensions.

Because the polyhedron abstract domain contains the interval abstract domain it is stronger (meaning more precise) than it. It however generally requires more calculations to converge.

3. Introducing and extending Difference-Bound Matrices

3.1 Difference-Bound Matrices

Again, we consider $\nu = (v_1, v_2, ..., v_n)$ a set of variables taking their values in $\mathbb{I} = \mathbb{Q}$, $\mathbb{Z}$, or $\mathbb{R}$. We will add $+\infty$ and $-\infty$ to $\mathbb{I}$ and work on $\bar{\mathbb{I}}$.
Difference bound matrices (DBM) allow to represent properties that are sets of linear equations of the form $v_i - v_j \leq c$. Formally speaking, a DBM is only a matrix of size n∗n taking its values in $\mathbb{I}$. The element $m_{i,j}$ represents the inequation $v_i - v_j \leq m_{i,j}$. If no information is available on $v_i - v_j$ then $m_{i,j} = +\infty$.

The operators $\cup\#$ and $\cap\#$ are simply min and max element per element. Also, the total order on $\mathbb{I}$ allows us to define a simple partial order on the set of all DBM :

$$u \preceq v \equiv \forall (i, j) \in [1, n]^2 \; u_{i,j} \leq v_{i,j}$$

Given a DBM m, the set of elements of $\nu \to \bar{\mathbb{I}}$ that satisfy all conditions described by m is called its ν domain and is written $D(m)$.

3.2 First extension : ν^0

We want DBMs to be able to represent conditions of the form $v_i \leq c$. To do so we simply add an element 0 to ν. We get a new set : $\nu^0 = (0, v_1, v_2, ..., v_n)$. Given a DBM m^0 over that new set of variables, the element $m_{i,0}$ simply represents the inequation $v_i \leq m_{i,0}$.

Thus, the addition of only one element to our set of variables allows us to represent conditions of the form $v_i \leq m_{i,0}$ and $v_i \geq m_{i,0}$.

3.3 Toward the Octagon domain : ν^+

The goal of the Octagon domain is to represent properties that are sets of conditions of the form $(\pm v_i + \pm v_j \leq c)$. Not all such conditions can be written using DBMs on ν^0, which is why we need the addition of other variables.
We will consider a set of size $2 * n$:

$$\nu^+ \equiv (v_1^+, v_1^-, v_2^+, v_2^-, ..., v_n^+, v_n^-)$$

In this set v_i^+ represents v_i and v_i^- represents $-v_i$. Thus, a DBM on ν^+ allows us to represent our target conditions.

Remark 1 *We didn't need to add an element 0 because the inequation* $v_i \leq c$ *can be written as* $v_i^+ - v_i^- \leq 2c$.

Let us consider a DBM m^+ over ν^+. Because we aren't interested in the set ν^+ but only in ν, we can't simply use $D(m^+)$. Thus, we define the ν^+ domain of a DBM m^+ as follows :

Definition 5 *The* ν^+ *domain of a DBM* m^+ *is :*
$D^+(m^+) \equiv \{(s_1, ..., s_n) \setminus (s_1^+, s_1^-, ..., s_n^+, s_n^-) \in D(m^+)\}$

We will now talk of constraints over ν^+ to talk about constraints on the ν^+ domain.

We also define an operator on subscripts :

Definition 6 $\bullet \to \bar{\bullet}$ *is the operator that swaps the sign of the linked variable. If i corresponds to* v_j^+ *then* $\bar{i}$ *corresponds to* v_j^- *and if i corresponds to* v_j^- *then* $\bar{i}$ *corresponds to* v_j^+.

Remark 2 $u^+ \preceq v^+ \Rightarrow D^+(v^+) \subseteq D^+(u^+)$

4. Defining the Octagon Domain

In this section we will refine our ν^+ domains and DBMs into workable canonical forms (which we will now call normal forms). Because a set of conditions of the form $(\pm x + \pm y \leq c)$ with bound solutions can only form octagons in two dimensions we will now call our ν^+ domains octagons. This follows the same logic used to name the polyhedron domain.

4.1 Coherence and closure

We remark that the same constraints over ν^+ can correspond to different constraints over ν (see figure 1). We introduce the notion of coherence to say that a DBM shows all the constraints that are equivalent to one of its constraints.

Definition 7 *A DBM m^+ will be said to be coherent if for every pair of potential constraints over ν corresponding to the same constraint over ν^+ they are either both absent or both present in m^+.*

The following theorem will help us to characterize coherence in DBMs :

Theorem 2 *m^+ is coherent* $\Leftrightarrow \forall i, j\ m^+_{i,j} = m^+_{\bar{i},\bar{j}}$

We also notice that some of the conditions in a DBM can sometimes be combined to produce stronger conditions than some of the other conditions found in the DBM.

Exemple 2 *$v_1 - v_2 \leq c$ and $v_2 - v_3 \leq c$ can be combined to produce $v_1 - v_3 \leq 2c$ which is stronger than $v_1 - v_3 \leq 3c$ if $c \geq 0$.*

To remedy to that situation we introduce the concept of closure :

Definition 8 *The closure m^* of a DBM m is defined by :*

- $m^*_{i,i} = 0\ \forall i$
- $m^*_{i,j} = min\ \{\Sigma_{k=1}^{M-1} m_{i_k,i_{k+1}}\ \backslash M \geq 1, i_1 = i, i_M = j,$ *with $(i_1, ..., i_M)$ a string of subscripts* $\}$

Intuitively, the closure simply replaces every condition by the strongest similar condition that can be obtained by combination of other conditions. This definition leads us to a closure algorithm that runs in $\bigcirc(n^3)$ (see fig 2).

Theorem 3

- $m = m^* \Leftrightarrow \forall i, j, k\ m_{i,j} \leq m_{i,k} + m_{k,j}$ *and* $m_{i,i} = 0$.
- $m^* = inf_{\preceq}\{n \backslash D(n) = D(m)\}$

Theorem 3 tells us that cloture is a way to get a normal form of a DBM.

constraint over $\mathcal{V}^+$	constraint(s) over $\mathcal{V}$
$v_i - v_j \leq c \quad (i \neq j)$	$v_i^+ - v_j^+ \leq c, \quad v_j^- - v_i^- \leq c$
$v_i + v_j \leq c \quad (i \neq j)$	$v_i^+ - v_j^- \leq c, \quad v_j^+ - v_i^- \leq c$
$-v_i - v_j \leq c \quad (i \neq j)$	$v_j^- - v_i^+ \leq c, \quad v_i^- - v_j^+ \leq c$
$v_i \leq c$	$v_i^+ - v_i^- \leq 2c$
$v_i \geq c$	$v_i^- - v_i^+ \leq -2$

Figure 1: Examples of translation between constraints on the ν domain and constraints on ν^+.

$$\begin{cases} \mathbf{m}_0 & \triangleq & \mathbf{m}, \\ \mathbf{m}_{k+1} & \triangleq & C_k(\mathbf{m}_k) \quad \forall k,\ 0 \leq k < N, \\ \mathbf{m}^* & \triangleq & \mathbf{m}_N, \end{cases}$$

where C_k is defined, $\forall k$, by:

$$\begin{cases} [C_k(\mathbf{n})]_{ii} & \triangleq & 0, \\ [C_k(\mathbf{n})]_{ij} & \triangleq & \min(\mathbf{n}_{ij}, \mathbf{n}_{ik} + \mathbf{n}_{kj}) \quad \forall i \neq j\ . \end{cases}$$

Figure 2: Closure algorithm

4.2 Strong closure

Because two DBM can have the same ν domain but different ν^+ domains, the notion of closure is insufficient to give us normal forms for octagons. We now introduce the notion of strong closure :

Definition 9 *m^+ is strongly closed if and only if :*

- *m^+ is closed*
- *m^+ is coherent*
- $\forall i, j\ m^+_{i,j} \leq (m^+_{i,\bar{i}} + m^+_{j,\bar{j}})/2$

Remark 3 *In definition 9 the third condition is there to extend the idea of closure to conditions of the form $v_i \leq c$, which are represented as was explained in remark 1.*

Again, we offer a strong closure algorithm (see fig 3). We define $m^+ \rightarrow (m^+)^\bullet$ the corresponding operator.

The following theorem ensures that strong closure provides a normal form for $\mathbb{I} = \mathbb{R}$ or $\mathbb{Q}$.

Theorem 4

- $m^+ = (m^+)^\bullet \Leftrightarrow m^+$ *is strongly closed*
- $(m^+)^\bullet = inf_{\preceq}\{n^+ \backslash D^+(n^+) = D^+(m^+)\}$

Remark 4 *In the previous theorem the second condition gives us the normal form for octagon domains.*

Strong closure does not gives normal forms for $\mathbb{I} = \mathbb{Z}$. For example, $v_i^+ - v_i^- \leq c$ implies that c i even, which isn't taken in account by our strong closure algorithm. It is recommended to approximate $\mathbb{I} = \mathbb{Z}$ by $\mathbb{I} = \mathbb{R}$.

$$\begin{cases} \mathbf{m}_0^+ & \triangleq & \mathbf{m}^+, \\ \mathbf{m}_{k+1}^+ & \triangleq & S^+(C_{2k}^+(\mathbf{m}_k^+)) \quad \forall k,\ 0 \le k < N, \\ (\mathbf{m}^+)^\bullet & \triangleq & \mathbf{m}_N^+, \end{cases}$$

where C_k^+ is defined, $\forall k$, by:

$$\begin{cases} \left[C_k^+(\mathbf{n}^+)\right]_{ii} \triangleq 0, \\ \left[C_k^+(\mathbf{n}^+)\right]_{ij} \triangleq \min(\ \mathbf{n}_{ij}^+,\ (\mathbf{n}_{ik}^+ + \mathbf{n}_{kj}^+), \\ \qquad\qquad\qquad (\mathbf{n}_{i\bar{k}}^+ + \mathbf{n}_{\bar{k}j}^+), \\ \qquad\qquad\qquad (\mathbf{n}_{ik}^+ + \mathbf{n}_{k\bar{k}}^+ + \mathbf{n}_{\bar{k}j}^+), \\ \qquad\qquad\qquad (\mathbf{n}_{i\bar{k}}^+ + \mathbf{n}_{\bar{k}k}^+ + \mathbf{n}_{kj}^+) \ ) \end{cases}$$

and S^+ is defined by:

$$\left[S^+(\mathbf{n}^+)\right]_{ij} \triangleq \min(\ \mathbf{n}_{ij}^+,\ (\mathbf{n}_{i\bar{i}}^+ + \mathbf{n}_{\bar{j}j}^+)/2 \ ) \ .$$

Figure 3: Strong closure algorithm

5. Operator and transfer functions

In this section we list some operators and test used to implement the octagon domain in a real life abstract analyzer. Details on Miné's analyzer will follow in section 6.

5.1 Tests

We need to define inclusion tests and equality tests. Those are immediately given by the following theorem.

Theorem 5 *Assuming no ν^+ domains are empty :*

- $D^+(m^+) \subseteq D^+(m^+) \Leftrightarrow (m^+)^\bullet \preceq n^+$
- $D^+(m^+) = D^+(m^+) \Leftrightarrow (m^+)^\bullet = (n^+)^\bullet$

5.2 Projections

The goal of a projection is to get the smallest interval containing all possible values for a variable given all the information at hand. Such an interval can easily be extracted from a DBM m^+ using the following theorem.

Theorem 6
$\{t \setminus \exists (s_0, ..., s_n) \in D^+(m^+)$ *such that* $s_i = t\} =$
$\left[-(m^+)^\bullet_{2i,2i+1}/2; (m^+)^\bullet_{2i+1,2i}/2\right]$

5.3 Union and intersection

As expected, unions and intersections are replaced by mins and maxs.

Definition 10

- $(m^+ \wedge n^+)_{i,j} \equiv min\ (m_{i,j}^+, n_{i,j}^+)$
- $(m^+ \vee n^+)_{i,j} \equiv max\ (m_{i,j}^+, n_{i,j}^+)$

Theorem 7

- $D^+(m^+ \wedge n^+) = D^+(m^+) \cap D^+(n^+)$
- $D^+(m^+ \vee n^+) \supseteq D^+(m^+) \cup D^+(n^+)$

5.4 Widening

Finally, we define a widening $\triangledown$.

Definition 11
$(m^+ \triangledown n^+)_{i,j} = m_{i,j}^+$ *if* $n_{i,j}^+ \le m_{i,j}^+$ *and* $+\infty$ *otherwise.*

Theorem 8 *The previous operator is a widening on DBMs.*

6. Applications and example

In this section we present the results obtained by Miné in his real life implementation of the octagon domain. The second part of this section will consist in an example on a toy code taken directly from Miné's original article [1].

6.1 Presentation of Miné's analyzer

Miné's analyzer was implemented in OCamel. It is designed for programs that work only with numerical variables -no pointer or arrays. Said programs must also only use value assignments, while loops, and if-then-else conditions.

The analyzer works by associating a DBM of size $2n$ to each program state, with n the number of variables. It then initializes the starting DBM at $\top$ (or at any other value if information on the input is given) and all the others at $\bot$. Once the initialization is done it runs an abstract execution as follows.

- Assignments change the value of every condition that uses the variable being changed. Conditions that concern no other variables can be updated directly. To update the other conditions that use the variable, we use a projection (see subsection 5.2) to learn the possible previous values of the variable. We then update the conditions considering the worst case scenario.
- Both parts of an if-then-else are handled separately before being merged using a $\cup\#$.
- Loops are seen as equations of the form $X = X \cup f(X)$ and are solved using the provided widening.

Miné's implementation was able to prove that the bubble sort, the heap sort, and the toy program of figure 4 do not perform out of bounds errors. While the bubble sort and the heap sort were already within the reach of the interval domain, the program of figure 4 was not. The details of the test of Miné's analyzer on that program can be found in subsection 6.2.

Most operators have a $\bigcirc(n^3)$ worst case time cost, with n the number of variables. The analysis is always more precise than the one offered by the interval domain and less costly than the one offered by the polyhedron domain. According to Miné, a more precise analysis could be obtained by using sets of DBMs rather than DBMs during the analysis. This would offer exact unions and better ways to discriminate between the two branches of an if test.

```
1   int tab[−m ... m];
2   for i = −m to m   tab[i] = 0;   {−m ≤ i ≤ m}
3   for j = 1 to M do
4     int a = 0;
5     for i = 1 to m
6       { 1 ≤ i ≤ m; −i + 1 ≤ a ≤ i − 1 }
7       if rand(2) = 0
8           then a = a + 1;   { −i + 1 ≤ a ≤ i }
9           else a = a − 1;   { −i ≤ a ≤ i − 1 }
10    tab[a] = tab[a] + 1;   { −m ≤ a ≤ m }
11  done;
```

Figure 4: Example of code outside the reach of the interval domain (we want to prove accesses to tab won't cause out of bounds errors).

6.2 Example

Figure 5 shows the results of a step by step execution of Miné's analyzer on the lines 5 to 9 of the program shown in figure 4. The goal of the analysis is to prove that no out of bound errors are raised when accessing tab in line 10 of the program. Line 7 of figure 5 simply means that a random coin toss is used.

7. Conclusion

The Octagon domain provides an extension on the canonical abstract domain based on DBM for little cost in performances. It allows to prove invariants of the form $(\pm x + \pm y \leq c)$ with a $\bigcirc(n^2)$ memory cost in the worst scenario per abstract state and a $\bigcirc(n^3)$ time cost in the worst scenario per abstract operation, with n the number of numerical variables. It can be seen as an intermediary between interval analysis and polyhedron analysis in terms of power and cost.

Miné wasn't able to test this domain on real life programs at the time of the redaction of [1] and it was his hope that it would be integrated in static analyzers in the future. It has since been integrated in the APRON library [5].

```
4   (l0) a ← 0; i ← 1 (l1)
    while i ≤ m do (l2)
7     if ?
8       then (l3) a ← a + 1 (l4)
9       else (l5) a ← a − 1 (l6)
      fi (l7)
      i ← i + 1 (l8)
11  done (l9)
```

$\mathbf{m}_0^+ = \top$
$\mathbf{m}_1^+ = \{i = 1;\ a = 0;\ 1 - i \leq a \leq i - 1\}$

First iteration of the loop
$\mathbf{m}_{2,0}^+ = \{i = 1;\ a = 0;\ 1 - i \leq a \leq i - 1;\ i \leq m\}$
$\mathbf{m}_{3,0}^+ = \mathbf{m}_{5,0}^+ = \mathbf{m}_{2.0}^+$
$\mathbf{m}_{4,0}^+ = \{i = 1;\ a = 1;\ 2 - i \leq a \leq i;\ i \leq m\}$
$\mathbf{m}_{6,0}^+ = \{i = 1;\ a = -1;\ -i \leq a \leq i - 2;\ i \leq m\}$
$\mathbf{m}_{7,0}^+ = \{i = 1;\ a \in [-1, 1];\ -i \leq a \leq i;\ i \leq m\}$
$\mathbf{m}_{8,0}^+ = \{i = 2;\ a \in [-1, 1];\ 1 - i \leq a \leq i - 1;\ i \leq m + 1\}$

Second iteration of the loop
$\mathbf{m}_{2,1}^+ = \mathbf{m}_{3,1}^+ = \mathbf{m}_{5,1}^+ = \mathbf{m}_{2,0}^+ \triangledown (\mathbf{m}_{8,0}^+)_{(i \leq m)}$
$= \{1 \leq i \leq m;\ 1 - i \leq a \leq i - 1\}$
$\mathbf{m}_{4,1}^+ = \{1 \leq i \leq m;\ 2 - i \leq a \leq i\}$
$\mathbf{m}_{6,1}^+ = \{1 \leq i \leq m;\ -i \leq a \leq i - 2\}$
$\mathbf{m}_{7,1}^+ = \{1 \leq i \leq m;\ -i \leq a \leq i\}$
$\mathbf{m}_{8,1}^+ = \{2 \leq i \leq m + 1;\ 1 - i \leq a \leq i - 1\}$

Third iteration of the loop
$\mathbf{m}_{2,2}^+ = \mathbf{m}_{2,1}^+$ *(fixpoint reached)*

$\mathbf{m}_2^+ = \mathbf{m}_{2,1}^+ \qquad \mathbf{m}_8^+ = \mathbf{m}_{8,1}^+$
$\mathbf{m}_9^+ = \{i = m + 1;\ 1 - i \leq a \leq i - 1\}$

Figure 5: Detailed analysis of lines 5-9 from figure 4. DBMs are shown as sets of constraints for the sake of lisibility.

References

[1] Antoine Miné *The Octagon abstract domain* In Proc. of the Workshop on Analysis, Slicing, and Transformation (AST'01), 310-319, Stuttgart, Germany, Oct. 2001. IEEE CS Press. doi: 10.1007/s10990-006-8609-1 available at "https://www-apr.lip6.fr/ mine/publi/article-mine-ast01.pdf"

[2] Thomas Jensen *Abstract Interpretation.* Given for classes in the "Université Rennes 1" for "Module SOS".

[3] Antoine Miné *The Octagon abstract domain* In Higher-Order and Symbolic Computation (HOSC), 19(1), 31–100, 2006. Springer. doi: 10.1007/s10990-006-8609-1. available at "https://www-apr.lip6.fr/ mine/publi/article-mine-HOSC06.pdf"
Despite sharing the same title this is not the same paper as [1].

[4] Antoine Miné *A new numerical abstract domain based on difference-bound matrices* in PADO II may 2001 vol. 2053 of LNCS, pp 155-172, Springer-Verlag

[5] APRON project "http://apron.cri.ensmp.fr/library/"