Stratified Static Analysis Based on Variable Dependencies*

David Monniaux† Julien Le Guen‡

September 1, 2011

Abstract

In static analysis by abstract interpretation, one often uses *widening operators* in order to enforce convergence within finite time to an inductive invariant. Certain widening operators, including the classical one over finite polyhedra, exhibit an unintuitive behavior: analyzing the program over a subset of its variables may lead a more precise result than analyzing the original program! In this article, we present simple workarounds for such behavior.

1 Introduction

During experiments, we found examples over which classical polyhedral analysis Cousot and Halbwachs (1978), even with alternative widenings Bagnara et al. (2005), would fail to discover some simple program invariants, which could sometimes even be discovered by interval analysis. This would even happen on simple loops, e.g. **for**(**int** i=0; i<N; i++), if the loop contained a nested loop not touching i: the analysis would not discover $\mathtt{i} \geq 0$! It is counter-intuitive that difficulties in analyzing the behavior of the program on other variables should lead to imprecise results for i.

In some of these examples, such as this simple loop, the lost invariants could be easily recovered by syntactic pattern-matching, but such techniques are brittle. We therefore searched for techniques inspired by our intuition that poor results on certain variables should not impact variables not depending on them.

1.1 Generalities and Notations

We consider the strongest invariant of a loop (or, more generally, of a program), defined as the least fixed point lfp Ψ of a monotone operator Ψ over sets of program states Cousot and Cousot (1992a). For instance, in program 2, the strongest invariant of the loop is the least fixed point in $(\mathcal{P}(\mathbb{Z} \times \mathbb{Z}), \subseteq)$ of the operator

$$\Psi(X) = \{(1,0)\} \cup \{(i+1, j+i) \mid (i,j) \in X \wedge i \leq 5\} \tag{1}$$

*This work was partially supported by ANR project "ASOPT

†CNRS / VERIMAG; http://www-verimag.imag.fr/\protect\unhbox\voidb@x\penalty\@M\{}monniaux/; VERIMAG is a joint laboratory of CNRS and Université Joseph Fourier

‡VERIMAG & STMicroelectronics

Explicit-state model-checking computes such invariants as explicitly represented sets of states (that is, for each state there exists some little data structure). Implicit-state model checking uses compact representations of such sets, such as binary decision diagrams, and computes the least solution of $\Psi(X) = X$ by finding the limit of the ascending sequence $X_0 = \emptyset$, $X_{n+1} = \Psi(X_n)$; for systems with at most n states, this limit is reached within at most n iterations. For infinite state systems such as software programs[1] such an approach is infeasible, because (a) the sets of states X_i may be large (or even infinite, if infinite non-determinism is used) (b) the sequence may not converge within a finite number of iterations.

Abstract interpretation Cousot and Cousot (1992a); Cortesi and Zanioli (2011) solves point (a) by replacing arbitrary sets of states by *over-approximations*; for instance, a set of points in $\mathbb{Z}^n$ or $\mathbb{Q}^n$ may be replaced by an enclosing convex polyhedron Halbwachs (1979); Cousot and Halbwachs (1978); Halbwachs et al. (1997). A given analysis thus restricts itself to a given *abstract domain* of sets of states; in this article, we focus, as an example, on the domain of polyhedra, but there exist many other abstract domains, for numerical Miné (2004) or non-numerical states. The operator Ψ on concrete states is replaced by an abstract operator $\Psi^\sharp$, satisfying a soundness condition $\Psi(X^\sharp) \subseteq \Psi^\sharp(X^\sharp)$ for all $X^\sharp$.[2]

Problem (b), that is, failure for the sequence $X^\sharp_{n+1} = \Psi^\sharp(X^\sharp_n)$ to become stationary, remains if the abstract domains contains infinite strictly ascending sequences;[3] this is for instance the case of the domain of convex polyhedra. Some form of convergence acceleration is thus needed. Starting with $u^\sharp_0 = \emptyset$, *upwards iterations with widening* Cortesi and Zanioli (2011); Cousot and Cousot (1992a) compute[4]

$$u^\sharp_{n+1} = u^\sharp_n \triangledown (u^\sharp_n \sqcup \Psi^\sharp(u^\sharp_n)) \tag{2}$$

$x \sqcup y$ is such that $x, y \subseteq x \sqcup y$ (in the case of polyhedra, $\sqcup$ is generally taken to be the convex hull), and $\triangledown$ is a *widening operator*, such that for all $x \subseteq y$, $y \subseteq x \triangledown y$ (*soundness* property), and any sequence of the form $u^\sharp_{n+1} = u^\sharp_n \triangledown v^\sharp_n$, where $v^\sharp_n$ is any other sequence, is stationary: after a certain N, it is constant (*termination* property). Then, $\Psi(u^\sharp_N) \subseteq \Psi^\sharp(u^\sharp_N) \subseteq u^\sharp_N \triangledown (u^\sharp_N \sqcup \Psi^\sharp(u^\sharp_N)) = u^\sharp_N$, thus $\Psi(u^\sharp_N) \subseteq u^\sharp_N$, which means that $u^\sharp_N$ is an inductive invariant of the program, in which the strongest invariant is included.

Once an inductive invariant $u^\sharp_N$ is obtained, it may be refined by *narrowing* iterations, which in practice generally consist in computing ${\Psi^\sharp}^k(u^\sharp_N)$ until the sequence becomes stationary or k exceeds a preset limit.

Widening operators have various unpleasant properties. The best known is that they bring *imprecision*: the result of widening/narrowing iterations may

[1] One of the authors once heard the remark that a program without dynamic allocation or recursion was just a finite-state automaton, thus all properties are decidable, including halting. For the purpose of practical analysis, except for very small and simple programs, such state spaces are so large that they should be treated as infinite.

[2] Some presentations of abstract interpretation distinguish the abstract element $X^\sharp$ from the set of states $\gamma(X^\sharp)$ that it represents. In this article, we chose not to, in order to simplify notations.

[3] Again, for practical purposes, it suffices that there exist exceedingly long finite ascending sequences for analysis to become unfeasible.

[4] Following the usage in APRON Jeannet and Miné (2009), our definition of $u \triangledown v$ assumes that $u \subseteq v$; if this is not the case, use $u \triangledown (u \sqcup v)$ instead.

be strictly larger than the least element of the abstract domain that is an inductive invariant, let alone an invariant (in Sec. 5 we shall list some alternative approaches that do not suffer from this inconvenience, at the expense of generality). The contribution of this article is a generic method to reduce some of the imprecision induced by widening.

1.2 Motivating Example

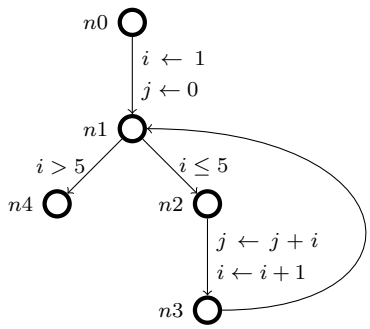
Classical polyhedral analysis Cousot and Halbwachs (1978),[5] when applied to Listing 1, discovers that $i \geq 1 \wedge i \leq 5$ is an invariant at the head of the loop. Yet, running the same analysis on Listing 2 yields $i \leq 5$ but not $i \geq 1$.

Listing 1: Loop until 5

```
int i=1;
while (i<=5) {
  i=i+1;
}
```

Listing 2: $j = i(i+1)/2$

```
int i=1, j=0;
while (i<=5) {
  j=j+i;
  i=i+1;
}
```



This example is not fortuitous: it models how to address consecutive lines of a matrix in lower triangular packed storage mode. In that memory-effective approach, the matrix is stored in memory as a unidimensional array, each line next to the preceding one, and line number i only uses i positions in the array: j is the index of the start of the line in the array.

Program 1 is an abstraction of Program 2: each execution of the latter maps to an execution of the former. Yet, the analysis of the former produces a more precise loop invariant than the analysis of the latter. This is an example of the non-monotonicity of analyzes using widenings, a long-known phenomenon (Cousot and Cousot, 1992b, ex. 11): a more precise abstraction may ultimately lead to less precision in the final analysis result.

Analysis of Program 2 with the basic upwards iteration and widening scheme (widening at every iteration) Cousot and Cousot (1992a), using the standard widening on polyhedra,[6] yields the successive polyhedra

- $i = 1 \wedge j = 0$
- $-i + j \geq -1 \wedge i \geq 1$: draw a line through the first two reachable states and obtain a polyhedron in (i, j) generated by vertex $(1, 0)$ and ray $(1, 1)$;

[5] One may try examples on B. Jeannet's online Interproc analyzer at `http://pop-art.inrialpes.fr/interproc/interprocweb.cgi`

[6] The standard widening on polyhedra $P_1 \triangledown_S P_2$, in intuitive terms, suppresses from P_2 constraints not present in P_1. In reality, its correct definition contains subtleties regarding polyhedra of dimension less than the dimension of the space, and the original definition Cousot and Halbwachs (1978) had to be corrected Halbwachs (1979). Bagnara et al. (2005) recalls the corrected definition.

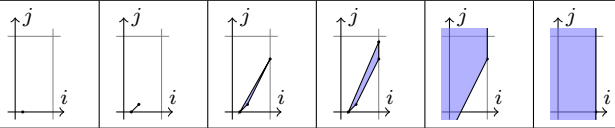
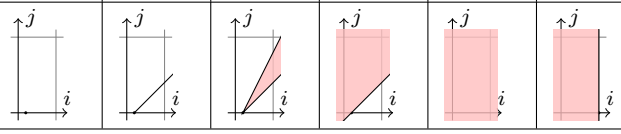
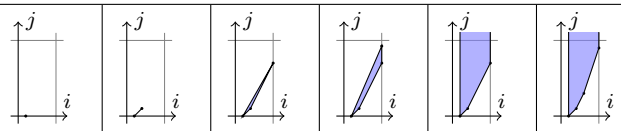
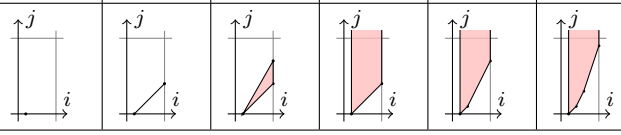
Analysis	Node	$u_0^\sharp$	$u_1^\sharp$	$u_2^\sharp$	$u_3^\sharp$	$u_4^\sharp$	$u_5^\sharp$
Classic	$n1$ (entry)						
	$n1$ (after $\triangledown$ or $\triangle$)						
Stratified	$n1$ (entry)						
	$n1$ (after $\triangledown$ or $\triangle$) and intersection with previous stratum						

Table 1: Comparison of classic static analysis (upward iterations with widening $\triangledown$ followed by descending iterations) and stratified static analysis on Program 2. Classic analysis loses the constraint $i \geq 1$ and finds $\top$ in 5 iterations. The upper bound $i \leq 5$ is found with one narrowing iteration. Stratified analysis on the stratum consisting of variable i first finds $1 \leq i \leq 5$. Then, it analyzes stratum i, j and intersect with result of stratum i. A fixed point is found after 4 iterations ($u_3^\sharp$, last line). The table also shows the polyhedra found after two narrowing iterations. The resulting polyhedron, even without narrowing iterations, is much more precise than the one found by classic analysis.

- $-i + j \geq -1 \wedge 7i - 4j \geq 7$: polyhedron in (i, j) generated by vertex $(1, 0)$ and rays $(1, 1)$ and $(4, 7)$.

So far, so good: such polyhedra still imply $i \geq 1$. At the next iteration, however, this constraint is lost and one gets the polyhedron $-i + j \geq -1$, and finally $\top$, the whole plane. The constraint $i \leq 5$ is recovered by one step of downwards iteration. Analysis with the improved widening proposed by Bagnara et al. Bagnara et al. (2005), as implemented in the Parma Polyhedra Library, yields a different iteration sequence, but still reaches $\top$ at the end.

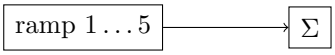
If one runs a polyhedral analysis on Program 1, one gets the inductive invariant $1 \leq i \leq 5$, which is also valid for Program 2. Intersecting this invariant with the output of the widening in the analysis of Program 2 yields a reasonably precise polyhedron (Table 1).

Thus, the basic idea of our method: *run preliminary analyzes over abstractions of the program obtained by removing some of the variables*, in order to refine the analysis of the complete program. In order to further convey our intuition, let us remark that Prog. 2 is the result of *loop fusion* over the following program :

```
for(i=1; i<=5; i++) t[i]=i;
for(i=1; i<=5; i++) j += t[i];
```

Normal forward polyhedral analysis on this program will find good invariants for both loops. In particular, the second loop may not perturb analysis of the first loop. It seems reasonable that the same applies to the code after loop fusion.

The same code could have been the result of the compilation into C of a data-flow program (e.g. Simulink or Lustre) consisting in a ramp generator and an integrator:



Again, it seems natural that the analysis of the integrator should not hamper the analysis of the ramp.

2 Stratified Analysis

We have investigated two approaches. In *stratified analysis*, we successively perform several static analyzes by abstract interpretation, the results from each analysis being used to refine the following ones. In *stratified widening*, a single analysis pass is performed, but with a widening improving on and derived from the traditional widening on polyhedra.

2.1 Dependency Strata

We consider a set $\mathcal{S}$ of subsets of the set of variables $\mathcal{V}$ of the program, such that $\mathcal{V} \in \mathcal{S}$; we order it by inclusion. An *immediate predecessor* of $S \in \mathcal{S}$, denoted by $S' \prec S$, is S' such that $S' \subsetneq S$ and there is no S'' such that $S' \subsetneq S'' \subsetneq S$.

In practice, if we have a relationship $v_1 \to v_2$ meaning "v_1 flows into v_2 through some computation" or "v_2 depends on v_1", then the elements of $\mathcal{S}$ are, in addition to $\mathcal{V}$ itself, subsets S of $\mathcal{V}$ closed by: if $v \in S$ and $v' \to v$, then $v' \in S$. One way to construct such subsets is to compute for each variable v the set $S(v) = \{v' \mid v' \to v\}$, and add this set to $\mathcal{S}$ unless it is already present. For better efficiency, one computes the strongly connected components of $\to$, and takes $S(v)$ for one v in each component.

Note that $\to$ needs not be the semantics dependency relation, which takes into account both data and control dependencies. In intuitive (and imprecise) terms, a variable x is said to be data-dependent on a variable y if x is assigned to by an expression where y appears; a variable x is said to be control-dependent on a variable y if x is assigned in a program branch executed or not executed according to the value of y. Collecting all program elements on which a variable depends, through data or control dependencies, is known as *slicing* Weiser (1984). If $\to$ takes into account all dependencies, then $S(v)$ is the slice of variables on which v depends.

A helpful intuition of our method is that it performs analyzes on program slices of increasing size; but this is somewhat misleading, because we do not make any assumption on $\to$ and thus it does not necessarily reflect all dependencies. In particular, ignoring control dependencies, compared conventional slicing, may produce simpler slices, of a more manageable size — X. Rival, when developing the Astrée static analyzer, observed that, for many variables, the slice corresponded to approximately 80% of the code, thus slicing did not significantly simplify the program Rival (2005a).

2.2 Informal Definition

Let S be a subset of the variables in program P. We note $P_{|S}$ the program P where all references to variables outside S have been replaced by nondet() nondeterministic choices.

Program P

```
int i=1, j=0;
while (i<=5) {
  j=j+i;
  if (j % 2 == 0) i=i+1;
}
```

$P_{|S}$ for $S = \{\mathtt{i}\}$

```
int i=1;
while (i<=5) {
  if (nondet()) i=i+1;
}
```

For any program P, let $C(P)$ be its collecting semantics: the set of reachable states of P. In order to simplify notations, for $S \subseteq S'$, we identify sets of states referring to the variables in S with their completion by all values for variables in $S' \setminus S$. For any S, $P_{|S}$ is a safe abstraction of P: $C(P) \subseteq C(P_{|S})$. More generally, if $S \subseteq S'$, $C(P_{|S'}) \subseteq C(P_{|S})$.

For any program P, let $A(P)$ be the result of static analysis of P. Correctness of the analysis means $C(P) \subseteq A(P)$. Let $A(P, K)$ be the result of the static analysis of P where the semantics of P is restricted to states in K: in other words, all states outside of K are removed from the transition relation. For any $K \supseteq C(P)$, $C(P) \subseteq A(P, K)$.

For each $S \in \mathcal{S}$, we compute the intermediate analysis result $R(S)$ after all $R(S')$, $S' \prec S$, have been computed, as follows:

$$R(S) = A\left(P_{|S}, \bigcap_{S' \prec S} R(S')\right) \tag{3}$$

Remark that in this formula, we could have made S' to range over all predecessors without changing the result; however, this would have been less efficient.

By induction on the length of the $\prec$-chains, for all S, $R(S) \supseteq C(P_{|S})$. At the end, $R(\mathcal{V}) \supseteq C(P)$ is a correct analysis result for the whole program; in fact, any $R(S) \supseteq C(P)$, so one can stop the analysis at any step, for instance because of a time limit.

This is the analysis performed in §1.2, with $\mathcal{S} = \{\{\mathtt{i}\}, \{\mathtt{i}, \mathtt{j}\}\}$.

2.3 Formal Definitions and Variants

Let $S \in \mathcal{S}$. We assume that the result $R(S')$ of the analysis for all $S' \prec S$ has already been computed. Let $K^\sharp = \bigcap_{S' \prec S} R(S')$; we assume that lfp $\Psi \subseteq R(S')$ for all $S' \prec S$ and thus that lfp $\Psi \subseteq K^\sharp$.

The analysis described at Eqn. 3 is defined by the sequence:

$$u^\sharp_{n+1} = u^\sharp_n \triangledown (u^\sharp_n \sqcup (\Psi^\sharp(u^\sharp_n \cap K^\sharp) \cap K^\sharp)) \tag{4}$$

We compute the limit $R(S) = u^\sharp_N$ of that stationary sequence, and output $u^\sharp_N \cap K^\sharp$.

Let us note $\Psi_{|A}(X) = \Psi(X \cap A) \cap A$. In other words, $\Psi_{|A}$ is Ψ with everything outside of A being discarded. The following lemma means that we do not change the strongest invariant by throwing out unreachable states in the definition of the semantics, which is intuitive.

Lemma 1. *lfp* $\Psi =$ *lfp* $\Psi_{|A}$ *for any* $A \supseteq$ *lfp* Ψ.

Proof. lfp $\Psi_{|A}$ is the limit of the ascending sequence defined by $X_0 = \emptyset$, $X_{n+1} = \Psi_{|A}(X_n)$, lfp Ψ that of $Y_0 = \emptyset$, $Y_{n+1} = \Psi(Y_n)$. By induction, for all n, $X_n = Y_n$. □

Corollary 2. $u_N^\sharp$, *and thus* $u_N^\sharp \cap K^\sharp$, *includes lfp* Ψ, *that is, the reachable states.*

Proof. Proof Because $y \subseteq x \triangledown y$ and $y \subseteq x \sqcup y$ for all x, y, $\Psi^\sharp(u_N^\sharp \cap K^\sharp) \cap K^\sharp \subseteq u_N^\sharp$ and thus $\Psi_{K^\sharp}(u_N^\sharp) = \Psi(u_N^\sharp \cap K^\sharp) \cap K^\sharp \subseteq u_N^\sharp$. Thus, lfp $\Psi_{K^\sharp} \subseteq u_N^\sharp$. The result follows from the lemma. □

We conclude that, by induction over $\prec$, for all S, lfp $\Psi \subseteq R(S)$.

We shall now describe a subtly different iteration scheme, which supposes some additional properties of $\triangledown$:

Definition 3. We say that $\triangledown$ satisfies the *"up to" termination condition* if for any fixed $K^\sharp$, any $u_0^\sharp \subseteq K^\sharp$, any sequence $v_n^\sharp \subseteq K^\sharp$ the sequence defined by $u_{n+1}^\sharp = (u_n^\sharp \triangledown v_n^\sharp) \cap K^\sharp$ is stationary if $u_n^\sharp \subseteq v_n^\sharp$ for all n.

This property ensures the correctness of widening "up to" Halbwachs et al. (1997), a well-known improvement to widening, and is true of the standard widening on polyhedra as well as Bagnara et al.'s improved widening (Bagnara et al., 2005, p. 53). Using the same notations and hypotheses as above, we use this iteration:

$$u_{n+1}^\sharp = (u_n^\sharp \triangledown (u_n^\sharp \sqcup (\Psi^\sharp(u_n^\sharp) \cap K^\sharp))) \cap K^\sharp \tag{5}$$

Again, once we get a stationary value $u_N^\sharp$ in this sequence, then it is such that lfp $\Psi \subseteq u_N^\sharp$:

Lemma 4. *If* $u_{N+1}^\sharp \subseteq u_N^\sharp$ *in Eqn. 5,* $u_N^\sharp$ *includes lfp* Ψ, *the set of reachable states.*

Proof. Proof $\Psi(u_N^\sharp) \cap K^\sharp \subseteq \Psi^\sharp(u_N^\sharp) \cap K^\sharp \subseteq u_{N+1}^\sharp \subseteq u_N^\sharp$, from the correctness of $\Psi^\sharp$. Furthermore, by construction, $u_N^\sharp \subseteq K^\sharp$, thus $\Psi(u_N^\sharp) \cap K^\sharp = \Psi_{|K^\sharp}(u_N^\sharp)$. $\Psi_{|K^\sharp}(u_N^\sharp) \subseteq u_N^\sharp$, thus lfp $\Psi_{|K^\sharp} \subseteq u_N^\sharp$. The result follows from Lem. 1. □

3 Stratified Widenings

An alternative to the method described in the preceding section, which runs successive analyzes of increasing precision, is to run a single analysis over a *reduced product* Cortesi and Zanioli (2011) of polyhedral domains, but with a special widening operator. We shall provide two options for that operator.

3.1 Widening with or without Reduction

We distinguish the internal state $(P_S)_{S \in \mathcal{S}}$ of the iteration sequence from the set of states represented, as in Monniaux (2009). The various abstract operations will therefore continue operating on polyhedra as usual: only the widening operator is replaced.

Our widening operators will take a tuple $(P_S)_{S\in\mathcal{S}}$ as a first argument and single polyhedron Q as a second argument. A tuple $(P_S)_{S\in\mathcal{S}}$ represents the polyhedron

$$\gamma\left((P_S)_{S\in\mathcal{S}}\right) = \bigcap_{S\in\mathcal{S}} P_S; \tag{6}$$

the tuples are ordered point-wise, $(P_S)_{S\in\mathcal{S}} \sqsubseteq (Q_S)_{S\in\mathcal{S}}$ if and only if for all S, $(P_S) \subseteq (Q_S)$.

We note $\pi_S(P)$ the projection of polyhedron P onto the variables in S. If $S \subseteq S'$, a polyhedron on the variables in S shall be also considered as a polyhedron on the variables in S' by keeping the same constraints. This means, in particular, that $P \subseteq \pi_S(P)$ for any P and S.

The first widening operator is very simple:

$$(P_S)_{S\in\mathcal{S}} \triangledown_1 Q = (P_S \triangledown \pi_S(Q))_{S\in\mathcal{S}} \tag{7}$$

where $\triangledown$ is any widening on polyhedra. This widening converges because each coordinate converges, since $\triangledown$ is a widening. It is obvious that, if $(P_S)_{S\in\mathcal{S}}$ is the resulting limit, then $\gamma\left((P_S)_{S\in\mathcal{S}}\right)$ is an inductive invariant.

The second widening applies internal reductions. $(R_S)_{S\in\mathcal{S}}$ denotes $(P_S)_{S\in\mathcal{S}}$ $\triangledown_2 (Q_S)_{S\in\mathcal{S}}$. We compute the R_S in ascending order with respect to $\prec$, with the convention that the intersection of zero polyhedra is the full polyhedron:

$$R_S = (P_S \triangledown \pi_S(Q)) \cap \bigcap_{S'\prec S} R_{S'} \tag{8}$$

Theorem 5. *Assuming that $\triangledown$ is a widening satisfying the "up to" termination condition (Def. 3), $\triangledown_2$ is a widening.*

Proof. Proof Let $u^{(n+1)} = u^{(n)} \triangledown_2 v^{(n)}$ be a sequence, with $u^{(n)} \sqsubseteq v^{(n)}$ for all n; each element $u^{(n)}$ consists in $u_S^{(n)}$ for $S \in \mathcal{S}$. We prove that for all $S \in \mathcal{S}$ the sequence $u_S^{(n)}$ is stationary, by induction over $\prec$.

For S with no predecessor, $(u_S^{(n)})$ is of the form $u_S^{(n+1)} = u_S^{(n)} \triangledown v_S^{(n)}$, and the result follows from $\triangledown$ being a widening.

Consider now the property satisfied for all $S' \prec S$. For all $S' \prec S$, $(u_{S'}^{(n)})$ is stationary; thus there is a N such that for $n \geq N$, all $(u_{S'}^{(n)})$ for $S' \prec S$ are constant. $\bigcap_{S'\prec S} u_{S'}^{(n)}$ is thus constant for $n \geq N$. The results follows from $\triangledown$ being a widening satisfying our additional property. □

Instead of polyhedra, one may use other abstract domains fitted with an operation $\sqcap$ such that $a \cap b \subseteq a \sqcap b$ for all a, b. Let us however note that $\triangledown_1$ and $\triangledown_2$ yield the same results as the ordinary widening $\triangledown$ if applied to domains, such as difference bound matrices or octagons Miné (2004) where $\triangledown$ and projection commute: $\pi_S(P) \triangledown \pi_S(Q) = \pi_S(P \triangledown Q)$, and therefore that they bring no improvement for such domains: the P_S are just projections of $P_{\mathcal{V}}$. More precisely:

Lemma 6. *Assume that $\pi_S(P) \triangledown \pi_S(Q) = \pi_S(P \triangledown Q)$ for all P and Q. Any iteration sequence of the form $P^{(n+1)} = P^{(n)} \triangledown Q^{(n)}$ then satisfies, for all n and $S \in \mathcal{S}$, $P_S^{(n)} = \pi_S(P_{\mathcal{V}}^{(n)})$, assuming this equality holds for $n = 0$.*

Proof. Proof Regarding $\triangledown_1$: by induction over n, for any S, $P_S^{(n+1)} = P_S^{(n)} \triangledown \pi_S(Q^{(n)}) = \pi_S(P_{\mathcal{V}}^{(n)}) \triangledown \pi_S(Q) = \pi_S(P_{\mathcal{V}}^{(n)} \triangledown Q) = \pi_S(P_{\mathcal{V}}^{(n+1)})$.

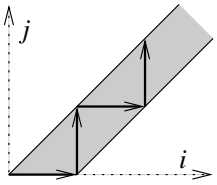
Regarding $\triangledown_2$: by induction over n, then by induction over $\mathcal{S}$ with respect to $\succ$: $(P_S^{(n)} \triangledown \pi_S(Q^{(n)})) \cap \bigcap_{S' \prec S} P_{S'}^{(n+1)} = (\pi_S(P_{\mathcal{V}}^{(n)}) \triangledown \pi_S(Q^{(n)})) \cap \bigcap_{S' \prec S} \pi_{s'}(P_{\mathcal{V}}^{(n+1)}) = \pi_S(P_{\mathcal{V}}^{(n)} \triangledown \pi_S(Q^{(n)})) \cap \bigcap_{S' \prec S} \pi_{s'}(P_{\mathcal{V}}^{(n+1)}) = \pi_S(P_{\mathcal{V}}^{(n+1)}) \cap \bigcap_{S' \prec S} \pi_{s'}(P_{\mathcal{V}}^{(n+1)}) = \pi_S(P_{\mathcal{V}}^{(n+1)})$, since for any X and $S' \succ S$, $\pi_{S'}(X) \cap \pi_S(X) = \pi_S(X)$. □

3.2 Generalized Reduction Leads to Nontermination

Communicating information between several abstract domains used at the same time is sometimes referred to as a *closure* or *reduction* operation. Our $\triangledown_2$ operation includes a partial closure, with information flowing from a to b if $a \prec b$, but not the reverse. One could wonder about applying reductions in all directions. Unfortunately, we would lose the termination property of widening, as demonstrated by the following example. [7]

Listing 3: Alternating increments

```
int i=0, j=0;
while (true) {
  if (i <= j) i++; else j++;
}
```



This loop has different behaviors on odd and even iterations: at iteration $2n$, $i = n$ and $j = n$; at iteration $2n+1$, $i = n+1$ and $j = n$. The results of a static analysis with polyhedra on (i, j), and unions instead of widenings, are, in constraint form: $P_{2n}^\sharp : P^\sharp \wedge i \leq n$ and $P_{2n+1}^\sharp : P^\sharp \wedge j \leq n$, $P^\sharp$ denoting $i \geq j \wedge i \leq j+1 \wedge j \geq 0$ (we identify $P^\sharp$ with the conjunctions of the constraints that define it). If for the iteration $n = 4$ we use widening,[8] we instead obtain $P_4^\sharp = P^\sharp$, which is an inductive invariant.

We have established that this program poses no challenge to "classical" polyhedral analysis. The same is true if we apply one of the analyzes of Sec. 2 or one of the widenings of Sec. 3.1. Let us now see what happens if we modify the $\triangledown_2$ operator of Sec. 3.1 by allowing reductions not following $\prec$.

Instead of the definition given at Eq. 8, we instead initialize all R_S to $P_S \triangledown \pi_S(Q)$, then apply some replacements, or *reductions*, of the form:

$$R_S := R_S \cap \bigcap_{S' \neq S} \pi_S(R_{S'}) \tag{9}$$

If we reach a fixed point for this replacement system, using the terminology from octagons Miné (2004), we say that we have applied the *closure* operation.

Let us first remark that $\gamma((R_S)_{S \in \mathcal{S}})$ is left unchanged any number of such reductions:

Lemma 7. *Let $(R'_S)_{S \in \mathcal{S}}$ be the same as $(R_S)_{S \in \mathcal{S}}$ except that $R'_{S_0} = R_{S_0} \cap \bigcap_{S' \neq S_0} \pi_{S_0}(R_{S'})$. Then, $\gamma((R'_S)_{S \in \mathcal{S}}) = \gamma((R_S)_{S \in \mathcal{S}})$.*

[7] The fact that widenings followed by reductions with cycles (reduce a using b, then reduce b using a) may not ensure termination is already known. For instance, closure in difference-bound matrices and octagons breaks termination. (Miné, 2004, example 3.7.3, p. 85)

[8] Applying unions at n first iterations and then applying widening is a standard technique known as *delayed widening*.

Proof. Proof $\gamma\left((R'_S)_{S\in\mathcal{S}}\right) = \bigcap_{S\in\mathcal{S}} R'_S = \gamma\left((R_S)_{S\in\mathcal{S}}\right) \cap \bigcap_{S'\neq S_0} \pi_{S_0}(R_{S'}) = \gamma\left((R_S)_{S\in\mathcal{S}}\right) \cap \bigcap_{S'\in\mathcal{S}} \pi_S(R_{S'})$. Since $R_{S'} \subseteq \pi_{S_0}(R_{S'})$ for any S', $\bigcap_{S'\in\mathcal{S}} \pi_S(R_{S'}) \supseteq \bigcap_{S'\in\mathcal{S}} R_{S'} = \gamma\left((R'_S)_{S\in\mathcal{S}}\right)$. The result follows. □

Because $\gamma\left((R_S)_{S\in\mathcal{S}}\right)$ does not change, after the reductions, $\gamma\left((R_S)_{S\in\mathcal{S}}\right)$ is still the same as $\gamma(P\triangledown Q)$. Our new "widening" thus verifies the soundness property (see Sec. 2.3); the problem is that it does not verify the termination property!

Let us have $\mathcal{S} = \{\{i\}, \{j\}, \{i,j\}\}$; instead of $P_{\{i\}}$, $P_{\{j\}}$ and $P_{\{i,j\}}$ we shall respectively note $I^\sharp$, $J^\sharp$ and $P^\sharp$. At iteration n, we shall therefore have a polyhedron $I^\sharp_n$ on $\{i\}$ (thus, an interval) and one polyhedron $J^\sharp_n$ on $\{j\}$ in addition to the polyhedron $P^\sharp_n$ on $\{i,j\}$. If using unions instead of widenings, we have $I^\sharp_{2n} = [0,n]$, $I^\sharp_{2n+1} = [0,n+1]$, $J^\sharp_{2n} = [0,n]$ and $J^\sharp_{2n+1} = [0,n]$. Consider now using widening at the iteration $n = 4$. $I^\sharp_4 = I^\sharp_3 = [0,2]$, but $J^\sharp_4 = [0,+\infty)$.

Let us now apply the closure operation: we replace $P^\sharp_4 = P^\sharp$ by its intersection with $I^\sharp_4$ and obtain $P^\sharp \wedge i \leq 2$; then we replace $J^\sharp_4$ by its intersection with the updated $P^\sharp_4$ and obtain $[0,2]$. At the next iteration, with the roles of $I^\sharp$ and $J^\sharp$ reversed, we obtain $I^\sharp_5 = [0,3]$, $J^\sharp_5 = [0,2]$ after closure, and then $I^\sharp_6 = [0,3]$, $J^\sharp_6 = [0,3]$.

The iterations with widening followed by closure behave, on $I^\sharp$ and $J^\sharp$, like those with unions — and *they do not converge within finite time.* Observe that this happens because we alternatively reduce $I^\sharp \to P^\sharp \to J^\sharp$ and $J^\sharp \to P^\sharp \to I^\sharp$, whereas the definitions of Sec. 3.1 only allow $I^\sharp \to P^\sharp$ and $J^\sharp \to P^\sharp$.

4 Experimental Results

The stratified analysis presented in section 2, in both variants (Eqn. 4 and Eqn. 5), was evaluated against the classical analysis described by Eqn. 2 on a set of benchmarks used by STMicroelectronics in the development cycle of its compilers, in addition to a few specific examples such as the one from Sec. 1.2.

LAO Kernels is a set of benchmarks internally used for the evaluation of compilers code generators and optimizations. It is mainly composed of small computational kernels representative of the target applications of STMicroelectronics (audio and video stream processing, embedded device control), associated with a testing harness to be able to run them on the target processor. It contains 63 functions, of which 49 contain at least one loop. Loops have to exhibit some properties, like a non-linear relation between variables in the loop scope, in order to benefit from this method. Stratified analysis finds a more precise invariant for 5 of these functions.

Among these 5 functions, *discrete cosine transform* has three nested loops. The intuition of why stratified analysis performs better is it obtains an invariant for the indices affected by the outer loop before attempting to analyze the inner loop, thus preventing imprecisions during the inner loop analysis to affect the invariant on the outer loop indices.

The dependency relation used to create the strata is based on a modified dataflow graph; strongly connected components (SCC) are reduced to supernodes, while keeping the existing dependency relations. Initial strata stem from the root nodes of this SCC dependency graph, additional ones are created by

following the dependency relations until one stratum encompasses all variables in the dependency graph. In the **while** loop of the listing 2, the variable j depends from i; the SCC nodes simply consist of $\{i\}$ and $\{j\}$, and the analysis creates two strata $\{i\}$ and $\{i, j\}$.

The two variants of stratified analysis described by Eqn. 4 and Eqn. 5 find the same results, and in all cases find invariants equal to or stronger than those obtained by the classical analysis. Bagnara et al.'s alternate widening Bagnara et al. (2005) yields iteration sequences different from those obtained by the classical widening, but ultimately finds the same invariant; thus, our approach improves on theirs on this benchmark set.

Table 2 shows the number of variables in the outermost stratum, along with the number of strata considered by the analysis and its overhead with respect to the standard analysis using only the classic widening. Some programs exhibit a large number of strata, impacting the cost of the analysis. It is possible to run the expensive stratified analysis after a first cheaper standard analysis, while focusing on certain loop nests (those reaching $\top$ for instance).

Function	# of vars	# of strata	Overhead
autocorrelation	9	8	5.55x
binary search	2	2	1.95x
discrete cosine transform	27	17	9.79x
integer power	2	3	2.29x
listing 2	2	2	1.66x

Table 2: Number of variable in the last stratum, number of strata and overhead of stratified analysis for programs that benefit from this method. The baseline for overhead measures is the classic analysis using bare widenings, without delay or widening-up-to).

We rely on the APRON numerical abstract domain library[9] Jeannet and Miné (2009) for all abstract domain computations. APRON implements, among other domains, convex polyhedra with the classical widening, with linearization of nonlinear expressions following Miné's approach Miné (2006). In addition, in order to compare with Bagnara et al.'s alternate widening, we used the Parma Polyhedra Library[10] Bagnara et al. (2008) (with the classical widening, the PPL produces exactly the same results as APRON up to equivalence of constraints, thus providing a means to test for possible bugs in the polyhedral computations).

5 Related Work

It has long been recognized that analysis using polyhedra over all variables in a program, or even all variables in a single function, is unfeasible because of the high complexity of polyhedral operations in higher dimensions. This is also true of weaker domains such as octagons. For this reason, the Astrée analyzer uses relational domains only on "packs" of variables Blanchet et al. (2002, 2003): for instance, if we have four variables a, b, c, d and two packs $\{a, b\}$ and $\{b, c, d\}$, the analysis will track relationships between a, b and b, c, d separately: no direct relation will be established between a and d.

[9] `http://apron.cri.ensmp.fr/library/`
[10] `http://www.cs.unipr.it/ppl/`

A related approach is *factoring of polyhedra* Halbwachs et al. (2006): when a polyhedron P is a Cartesian product $P_1 \times \ldots \times P_n$ of polyhedra in lower dimension, with respectively v_i vertices (or, more generally, generators), it is often advantageous to keep this product representation as much as possible instead of considering it as a polyhedron of $\prod_i v_i$ vertices, because of algorithms that need to work on the generator representation. An alternative is to dispense totally with the generator representation Simon and King (2005); Simon and Chen (2010).

The literature on slicing is abundant, since the early 1980s Weiser (1984). *Syntactic slicing* extracts all program statements, variables etc. that affect the value of variable v, or, rather, a safe superset thereof. The resulting *slice* is executable, which is interesting for testing or debugging methods, but less so for abstract interpretation; this is why we may use lax dependency relations (Sec. 2.1), since we in effect replace any unknown dependency by nondeterministic choice. *Semantic slicing* relaxes the requirement that the resulting program be a syntactic subset of the original program Ward and Zedan (2007). X. Rival considers a form of abstract semantic slicing Rival (2005a,b), where program executions are restricted to those affecting the reachability of undesirable program states (alarms); in contrast, our method does not suppose we have a set of properties (absence of alarms) to prove.

The design of widening operators is surprisingly difficult. The original widening operator on polyhedra Cousot and Halbwachs (1978) was sensitive to syntax: different ways of representing the same polyhedron in constraint form yielded different widened polyhedra; this problem was later fixed Halbwachs (1979). Because the result of iterations with widening is non-monotonic, precision is highly heuristic: in particular, replacing a widening operator by one producing smaller polyhedra at each iteration does not necessarily translate in a smaller invariant in the end (Bagnara et al., 2005, p. 42).

Despite this caveat, many widening operators have been proposed for convex polyhedra (Bagnara et al., 2005, p. 30)Simon and Chen (2010). Many are variants on the classical widening: some apply union in lieu of the classical widening in a way that does not preclude termination Bagnara et al. (2005); the "up to" widening, also known as *widening with thresholds* or *limited widening* Halbwachs et al. (1997), extracts possibly relevant constraints from the program and keeps in $P \triangledown Q$ the constraints from that set satisfied by both P and Q; a related idea is *widening with landmarks*, which uses estimates of the number of supplementary iterations necessary to enable a currently disabled transition Simon and King (2006); *widening with a care set* uses a proof goal and counterexamples in order to guide the widening Wang et al. (2007). Our approach is largely orthogonal to these, and in fact can be combined with them.

In the recent years, there has been much interest in techniques for inferring invariants without doing conventional Kleene iterations. *Policy iteration* (also called *strategy iteration*; the technique is inspired by game theory) exists in two flavors. Descending policy iteration Gaubert et al. (2007) solves a descending sequence of least fixed points of simpler operators; these least fixed points may be solved approximately using widenings, thus this technique is orthogonal to ours. In contrast, ascending policy iteration Gawlitza and Seidl (2007) and other techniques based on constraint programming Sankaranarayanan (2005) or quantifier elimination Monniaux (2010) provide some optimality guarantees, but impose restrictions on the kind of program instructions supported. Such

restrictions may be lifted by abstracting program operations into the supported subset Miné (2004), which may in turn entail an outer loop with widenings.

We finally note that nothing in our approach is specific to polyhedra, or even to numerical domains.

6 Conclusion

Following our intuition that failure to analyze well parts of a program should not negatively influence precision on other parts not depending on them, we proposed four analysis schemes: two proceed by analyzes of restrictions of the program code to variable subsets, the other ones use alternative widening operators. Though we focused on improving the classical polyhedral analysis, two of our methods apply to any abstract domain, and the two other ones make a reasonable assumption on the underlying abstract domain and its widening operator.

References

R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

Roberto Bagnara, Patricia M. Hill, Elisa Ricci, and Enea Zaffanella. Precise widening operators for convex polyhedra. *Science of Computer Programming*, 58(1–2):28–56, October 2005. doi: 10.1016/j.scico.2005.02.003.

Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. Design and implementation of a special-purpose static program analyzer for safety-critical real-time embedded software. In Torben Ægidius Mogensen, David A. Schmidt, and I. Hal Sudborough, editors, *The Essence of Computation: Complexity, Analysis, Transformation*, number 2566 in LNCS, pages 85–108. Springer, 2002.

Bruno Blanchet, Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. A static analyzer for large safety-critical software. In *PLDI*, pages 196–207. ACM, 2003.

Agostino Cortesi and Matteo Zanioli. Widening and narrowing operators for abstract interpretation. *Computer Languages, Systems & Structures*, 37(1): 24–42, 2011. doi: 10.1016/j.cl.2010.09.001.

Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *J. of Logic and Computation*, pages 511–547, August 1992a. ISSN 0955-792X. doi: 10.1093/logcom/2.4.511.

Patrick Cousot and Radhia Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In *PLILP*, vol-

ume 631 of *LNCS*, pages 269–295. Springer, 1992b. ISBN 3-540-55844-6. doi: 10.1007/3-540-55844-6_101.

Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Principles of Programming Languages (POPL)*, pages 84–96. ACM, 1978. doi: 10.1145/512760.512770.

Stéphane Gaubert, Éric Goubault, Ankur Taly, and Sarah Zennou. Static analysis by policy iteration on relational domains. In Rocco de Nicola, editor, *Programming Languages and Systems (ESOP)*, volume 4421 of *LNCS*, pages 237–252. Springer, 2007. ISBN 978-3-540-71316-6.

Thomas Gawlitza and Helmut Seidl. Precise fixpoint computation through strategy iteration. In Rocco de Nicola, editor, *Programming Languages and Systems (ESOP)*, volume 4421 of *LNCS*, pages 300–315. Springer, 2007. ISBN 978-3-540-71316-6. doi: 10.1007/978-3-540-71316-6_21.

Nicolas Halbwachs. *Détermination automatique de relations linéaires vérifiées par les variables d'un programme.* PhD thesis, Université scientifique et médicale de Grenoble, 1979.

Nicolas Halbwachs, Yann-Erick Proy, and Patrick Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.

Nicolas Halbwachs, David Merchat, and Laure Gonnord. Some ways to reduce the space dimension in polyhedra computations. *Formal Methods in System Design*, 29(1):79–95, 2006. doi: 10.1007/s10703-006-0013-2.

Bertrand Jeannet and Antoine Miné. APRON: A library of numerical abstract domains for static analysis. In Ahmed Bouajjani and Oded Maler, editors, *CAV*, volume 5643 of *LNCS*, pages 661–667. Springer, 2009. ISBN 978-3-642-02657-7. doi: 10.1007/978-3-642-02658-4.

A. Miné. Symbolic methods to enhance the precision of numerical abstract domains. In *VMCAI*, volume 3855 of *LNCS*, pages 348–363, Charleston, South Carolina, USA, January 2006. Springer.

Antoine Miné. *Domaines numériques abstraits faiblement relationnels.* PhD thesis, École polytechnique, 2004.

David Monniaux. A minimalistic look at widening operators. *Higher order and symbolic computation*, 22(2):145–154, December 2009. ISSN 1388-3690. doi: 10.1007/s10990-009-9046-8.

David Monniaux. Automatic modular abstractions for template numerical constraints. *Logical Methods in Computer Science*, June 2010. ISSN 1860-5974. doi: 10.2168/LMCS-6(3:4)2010.

Xavier Rival. Understanding the origin of alarms in Astrée. In Chris Hankin and Igor Siveroni, editors, *SAS*, volume 3672 of *LNCS*, pages 303–319. Springer, 2005a. ISBN 3-540-28584-9.

Xavier Rival. *Traces Abstraction in Static Analysis and Program Transformation.* PhD thesis, École polytechnique, 2005b.

Sriram Sankaranarayanan. *Mathematical Analysis of Programs*. PhD thesis, Stanford University, 2005.

Axel Simon and Liqian Chen. Simple and precise widenings for H-polyhedra. In Kazunori Ueda, editor, *APLAS*, volume 6461 of *LNCS*, pages 139–155. Springer, 2010. ISBN 978-3-642-17163-5. doi: 10.1007/978-3-642-17164-2_11.

Axel Simon and Andy King. Exploiting sparsity in polyhedral analysis. In Chris Hankin and Igor Siveroni, editors, *SAS*, volume 3672 of *LNCS*, pages 336–351. Springer, 2005. ISBN 3-540-28584-9. doi: 10.1007/11547662_23.

Axel Simon and Andy King. Widening polyhedra with landmarks. In *APLAS (Programming languages and systems)*, volume 4279 of *LNCS*, pages 166–182. Springer, November 2006. ISBN 3-540-48937-1. doi: 10.1007/11924661_11.

Chao Wang, Zijiang Yang, Aarti Gupta, and Franjo Ivançií. Using counterexamples for improving the precision of reachability computation with polyhedra. In *CAV (Computer aided verification)*, volume 4590 of *LNCS*, pages 352–365. Springer, July 2007. doi: 10.1007/978-3-540-73368-3_40.

Martin Ward and Hussein Zedan. Slicing as a program transformation. *ACM Trans. Program. Lang. Syst.*, 29, April 2007. ISSN 0164-0925. doi: 10.1145/1216374.1216375.

Mark Weiser. Program slicing. *IEEE Trans. Software Eng.*, 10(4):352–357, 1984.