Widening Polyhedra with Landmarks

Axel Simon and Andy King

Computing Laboratory, University of Kent, Canterbury, UK
{a.simon,a.m.king}@kent.ac.uk

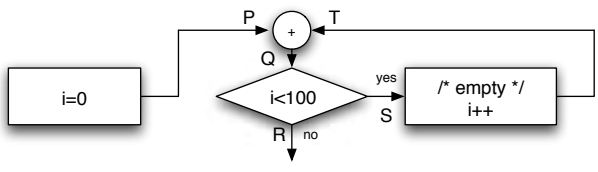
Abstract. The abstract domain of polyhedra is sufficiently expressive to be deployed in verification. One consequence of the richness of this domain is that long, possibly infinite, sequences of polyhedra can arise in the analysis of loops. Widening and narrowing have been proposed to infer a single polyhedron that summarises such a sequence of polyhedra. Motivated by precision losses encountered in verification, we explain how the classic widening/narrowing approach can be refined by an improved extrapolation strategy. The insight is to record inequalities that are thus far found to be unsatisfiable in the analysis of a loop. These so-called landmarks hint at the amount of widening necessary to reach stability. This extrapolation strategy, which refines widening with thresholds, can infer post-fixpoints that are precise enough not to require narrowing. Unlike previous techniques, our approach interacts well with other domains, is fully automatic, conceptually simple and precise on complex loops.

1 Introduction

In the last decade, the focus of static analysis has shifted from program optimisations towards program verification [5]. In this context, the abstract domain of polyhedra [2, 10] has attracted much interest due to its expressiveness, as have sub-classes of polyhedra [18, 19, 21, 22] that solve specific analysis tasks more efficiently. However, an inherent problem in polyhedral analysis is the ability to finitely reason about loops. Since the values of variables may differ in each iteration, each iterate may well be described by a different polyhedron. In order to quickly analyse a large or potentially infinite number of iterations, special acceleration techniques are required. One such acceleration framework is provided by the widening/narrowing approach to abstract interpretation [9, 10].

1.1 A Primer On Widening/Narrowing

In order to illustrate the widening/narrowing approach on the domain of polyhedra and to discuss the implications of applying narrowing in an actual analyser, consider the control flow graph of `for (i=0; i<100; i++) {/*empty*/}`:



The analysis amounts to characterising the values that can arise on the edges of the control flow graph. To this end, each edge is decorated with a polyhedron describing the relationships between the values of the variables on that edge. Given that the program contains only a single variable i, the polyhedra P, Q, R, S, T coincide with intervals over the reals. In the example, the polyhedron $P = \{i \in \mathbb{R} \mid 0 \leq i \leq 0\}$ describes the value of i at the beginning of the program. The +-node joins the polyhedra P and T to obtain $Q = P \sqcup T$. This join corresponds to the smallest convex polyhedron that includes the set of points $P \cup T$. Due to the integrality of i, the polyhedra that characterise the two outcomes of the test $i < 100$ are $R = Q \sqcap \{i \in \mathbb{R} \mid i \geq 100\}$ and $S = Q \sqcap \{i \in \mathbb{R} \mid i \leq 99\}$ where $\sqcap = \cap$ denotes the intersection of two polyhedra. The last polyhedron T is characterised by the affine map $T = \{i + 1 \mid i \in S\}$.

A solution of these equations can be found by applying Jacobi iteration [8], which calculates new polyhedra $P_{j+1}, Q_{j+1}, R_{j+1}, S_{j+1}, T_{j+1}$ from the polyhedra of the previous iteration P_j, Q_j, R_j, S_j, T_j. To ensure rapid convergence, a widening point must be inserted into the Q, S, T cycle. Widening at Q amounts to replacing the equation for Q with $Q_{j+1} = Q_j \nabla (P_j \sqcup T_j)$ where ∇ is a widening operator that removes unstable bounds [9]. The possible values of `i` are given below where $\perp$ denotes the empty set; the updated entries are shown in bold:

j	P_j	Q_j	R_j	S_j	T_j
1	$[\mathbf{0}, \mathbf{0}]$	$\perp$	$\perp$	$\perp$	$\perp$
2	$[0, 0]$	$[\mathbf{0}, \mathbf{0}]$	$\perp$	$\perp$	$\perp$
3	$[0, 0]$	$[0, 0]$	$\perp$	$[\mathbf{0}, \mathbf{0}]$	$\perp$
4	$[0, 0]$	$[0, 0]$	$\perp$	$[0, 0]$	$[\mathbf{1}, \mathbf{1}]$
5	$[0, 0]$	$[\mathbf{0}, \infty]$	$\perp$	$[0, 0]$	$[1, 1]$

j	P_j	Q_j	R_j	S_j	T_j
6	$[0, 0]$	$[0, \infty]$	$[\mathbf{100}, \infty]$	$[\mathbf{0}, \mathbf{99}]$	$[1, 1]$
7	$[0, 0]$	$[0, \infty]$	$[100, \infty]$	$[0, 99]$	$[\mathbf{1}, \mathbf{100}]$
8	$[0, 0]$	$[0, \infty]$	$[100, \infty]$	$[0, 99]$	$[1, 100]$
1'	$[0, 0]$	$[\mathbf{0}, \mathbf{100}]$	$[100, \infty]$	$[0, 99]$	$[1, 100]$
2'	$[0, 0]$	$[0, 100]$	$[\mathbf{100}, \mathbf{100}]$	$[0, 99]$	$[1, 100]$

In iteration 5, the output of the +-node is $P_4 \sqcup T_4 = [0, 1]$. The widening operator compares $P_4 \sqcup T_4$ against $Q_4 = [0, 0]$ and removes the unstable upper bound, yielding $Q_5 = [0, \infty]$. Stability is reached in iteration 8. The calculated post-fixpoint is now refined. This is realised by replacing widening with narrowing, i.e. $Q_{j+1} = Q_j \triangle (P_j \sqcup T_j)$. For polyhedra, it is sufficient to put $\triangle = \sqcap$ and to bound the number of iterations [9, page 290]. Hence, let $Q_{j+1} = Q_j \sqcap (P_j \sqcup T_j)$ which yields a refined state 1' and a further refinement $2'$ which, in this case, coincides with the least fixpoint of the original equations.

1.2 The Limitations of Narrowing

To illustrate one drawback of narrowing, consider a re-analysis of the above example where the widening is applied on S rather than on Q. In particular, let $S_{j+1} = S_j \nabla (Q_i \sqcap \{i \in \mathbb{R} \mid i \leq 99\})$. The analyses differ after the first 4 iterations:

j	P_j	Q_j	R_j	S_j	T_j
5	$[0, 0]$	$[0, 1]$	$\perp$	$[0, 0]$	$[1, 1]$
6	$[0, 0]$	$[0, 1]$	$\perp$	$[\mathbf{0}, \infty]$	$[1, 1]$
7	$[0, 0]$	$[0, 1]$	$\perp$	$[0, \infty]$	$[\mathbf{1}, \infty]$
8	$[0, 0]$	$[\mathbf{0}, \infty]$	$\perp$	$[0, \infty]$	$[\mathbf{1}, \infty]$
9	$[0, 0]$	$[0, \infty]$	$[\mathbf{100}, \infty]$	$[0, \infty]$	$[1, \infty]$

j	P_j	Q_j	R_j	S_j	T_j
10	$[0, 0]$	$[0, \infty]$	$[100, \infty]$	$[0, \infty]$	$[1, \infty]$
1'	$[0, 0]$	$[0, \infty]$	$[100, \infty]$	$[\mathbf{0}, \mathbf{99}]$	$[1, \infty]$
2'	$[0, 0]$	$[0, \infty]$	$[100, \infty]$	$[0, 99]$	$[\mathbf{1}, \mathbf{100}]$
3'	$[0, 0]$	$[\mathbf{0}, \mathbf{100}]$	$[100, \infty]$	$[0, 99]$	$[1, 100]$
4'	$[0, 0]$	$[0, 100]$	$[\mathbf{100}, \mathbf{100}]$	$[0, 99]$	$[1, 100]$

In the first analysis, only the polyhedra Q and R are larger before narrowing commences. In the second analysis, S and T are also larger before narrowing. To illustrate the impact of this in the context of verification, suppose `/*empty*/` is replaced by `b = array[i]` where `array` has 100 elements. To avoid an avalanche of false warning messages it is common practise to intersect S with the legal range of the index i [5], in this case $0 \leq i \leq 99$, yielding the polyhedron S', and thereafter use S' instead of S. Moreover, since the out-of-bounds check amounts to the subsumption test $S \not\subseteq S'$, it is straightforward to perform the check during fixpoint calculation; the test could be postponed until a fixpoint is reached, but this would require S' to be recalculated unnecessarily. However, this technique does not combine well with narrowing since a warning is issued if S is nominated for widening rather than Q, i.e. the placement of the widening point can determine whether a warning is issued or not.

Another implication of reducing a post-fixpoint with narrowing relates to domain interaction. Assume that the array above is embedded into a C structure declared as `struct { int[100] array; int* p } s;` and that the loop body is changed to `b = s.array[i]`. Consider again the second analysis in which S is widened to $[0, \infty]$ so that the upper bound of the array index i is lost. In this case, a points-to analysis [17, 23] would generate a spurious l-value flow from `s.p` to `b`. Once narrowing infers $0 \leq i < 100$ it is desirable to remove this spurious flow. Alas, points-to analyses are typically formulated in terms of either closure operations [17] or union-find algorithms [23], none of which support the removal of flow information. Thus, even if narrowing can recover precision in one domain, the knock-on precision loss induced in other domains may be irrecoverable.

Furthermore, narrowing on polyhedra [9] cannot recover precision if the loop invariant is expressed as a disequality [5]. For instance, narrowing has no effect if the loop invariant in the example is changed from `i<100` to the equivalent `i!=100`. Since it is unrealistic to modify the program under test, a substitute for narrowing is required to analyse programs with disequalities as loop conditions.

1.3 Our Contribution to Widening/Narrowing

Rather than recovering inequalities through narrowing that were widened away, our contribution is to use unsatisfiable inequalities as oracles to guide the fixpoint acceleration. Specifically, we propose widening with landmarks, which records inequalities that were found to be unsatisfiable in two consecutive iterates. We then extrapolate to the first iterate that makes any of these inequalities satisfiable. If this extrapolation is not a fixpoint, we continue until no unsatisfiable inequalities remain, at which point standard widening is applied [1, 14]. The rationale for observing unsatisfiable inequalities is that the transition from unsatisfiable to satisfiable indicates a change in the behaviour of a program. Widening with landmarks is similar in spirit to widening with thresholds [5]. In this related approach, the value of an unstable variable is extrapolated to the next threshold from a set of user-supplied values. Rather than guiding widening with thresholds on individual variables, our approach automatically extracts linear inequalities from the program which bound the degree of extrapolation.

After introducing notation for polyhedra manipulation, Section 3 presents a worked example of a string buffer analysis that conveys the ideas behind widening with landmarks. Sections 4 and 5 formalise the notion of landmarks which are used in Section 6 to define an extrapolation strategy. Section 7 comments on our implementation and explains how widening with landmarks can be added to an existing analysis. We discuss related work in Section 8 and conclude in Section 9.

2 Preliminaries

Let $\boldsymbol{x} = \langle x_1, \dots x_n \rangle$ denote an ordered set of variables, let *Lin* denote the set of linear expressions of the form $\boldsymbol{a} \cdot \boldsymbol{x}$ where $\boldsymbol{a} \in \mathbb{Z}^n$ and let *Ineq* denote the set of linear inequalities $\boldsymbol{a} \cdot \boldsymbol{x} \leq c$ where $c \in \mathbb{Z}$. Moreover, let e.g. $6x_3 \leq x_1 + 5$ abbreviate $\langle -1, 0, 6, 0, \dots 0 \rangle \cdot \boldsymbol{x} \leq 5$ and let e.g. $x_2 = 7$ abbreviate the two opposing inequalities $7 \leq x_2$ and $x_2 \leq 7$. Each inequality $\boldsymbol{a} \cdot \boldsymbol{x} \leq c \in \mathit{Ineq}$ induces a half-space $[\![\boldsymbol{a} \cdot \boldsymbol{x} \leq c]\!] = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a} \cdot \boldsymbol{x} \leq c\}$. Each finite set of inequalities $I = \{\iota_1, \dots \iota_m\} \subseteq \mathit{Ineq}$ induces a closed, convex polyhedron $[\![I]\!] = \bigcap_{i=1}^{m} [\![\iota_i]\!]$. Let $\mathit{Poly} = \{[\![I]\!] \mid I \subseteq \mathit{Ineq}, |I| \in \mathbb{N}\}$ denote the set of all (finitely generated) polyhedra. Given two polyhedra $P_i = [\![I_i]\!]$, $i = 1, 2$, define $P_1 \sqcap P_2 = [\![I_1 \cup I_2]\!]$ and let $P_1 \sqsubseteq P_2$ iff $[\![I_1]\!] \subseteq [\![I_2]\!]$. Let $P_1 \sqcup P_2 = \sqcap\{P \in \mathit{Poly} \mid P_1 \sqsubseteq P \wedge P_2 \sqsubseteq P\}$; equivalently let $P_1 \sqcup P_2 = \mathit{cl}(\mathit{hull}(P_1 \cup P_2))$ where *cl* denotes topological closure and *hull* is the convex hull operation on sets of points [10]. A set of inequalities $I \subseteq \mathit{Ineq}$ is said to be unsatisfiable if $[\![I]\!] = \emptyset$, otherwise it is satisfiable. The lattice $\langle \mathit{Poly}, \sqsubseteq, \sqcap, \sqcup \rangle$ contains infinite ascending chains $P_1 \sqsubseteq P_2 \sqsubseteq P_3 \dots$ so that standard Kleene iteration [9] may not converge onto a fixpoint in finite time. To guarantee convergence, widening operators $\nabla : \mathit{Poly} \times \mathit{Poly} \to \mathit{Poly}$ have been proposed for *Poly* which are required to satisfy the following properties [10]:

1. $\forall x, y \in \mathit{Poly} \,.\, x \sqsubseteq x \nabla y$
2. $\forall x, y \in \mathit{Poly} \,.\, y \sqsubseteq x \nabla y$
3. for all increasing chains $x_0 \sqsubseteq x_1 \sqsubseteq \dots$, the increasing chain defined by $y_0 = x_0$ and $y_{i+1} = y_i \nabla x_{i+1}$ is ultimately stable.

Besides the standard lattice operations, we introduce a family of projection operators $\exists_{x_i} : \mathit{Poly} \to \mathit{Poly}$ such that $\exists_{x_i}(Q) = \{\langle x_1, \dots, x_{i-1}, x, x_{i+1}, \dots x_n \rangle \mid \langle x_1, \dots x_n \rangle \in Q, x \in \mathbb{R}\}$. Intuitively, $\exists_{x_i}(Q)$ removes any information pertaining to x_i from the polyhedron $Q \in \mathit{Poly}$. This is useful to model assignment, e.g. $\exists_{x_i}(Q) \sqcap [\![\{x_i = 42\}]\!]$ updates the value of x_i to 42. Finally, in order to find the minimum value of an expression $\boldsymbol{a} \cdot \boldsymbol{x}$ such that $\boldsymbol{x} \in P$, we introduce the operation $\min : \mathit{Lin} \times \mathit{Poly} \to (\mathbb{Z} \cup \{-\infty\})$. To this end, let $C = \{c \in \mathbb{Z} \mid P \sqcap [\![\{\boldsymbol{a} \cdot \boldsymbol{x} \leq c\}]\!] \neq \emptyset\}$, that is, C contains all constants c such that the half-space defined by $\boldsymbol{a} \cdot \boldsymbol{x} \leq c$ intersects with P, and define

$$\min(\boldsymbol{a} \cdot \boldsymbol{x}, P) = \begin{cases} \min(C) & \text{if } \min(C) \text{ exists} \\ -\infty & \text{otherwise.} \end{cases}$$

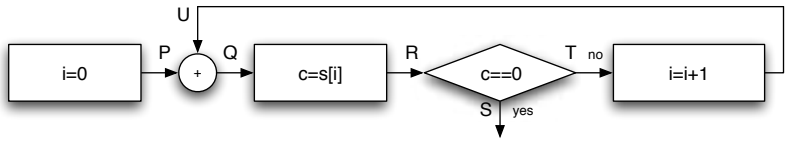
Observe that $\min(\boldsymbol{a} \cdot \boldsymbol{x}, P)$ can be realised with Simplex: if there exists $\boldsymbol{y} \in \mathbb{R}^n$ that minimises the expression $\boldsymbol{a} \cdot \boldsymbol{y}$ over P, then put $\min(\boldsymbol{a} \cdot \boldsymbol{x}, P) = \lceil \boldsymbol{a} \cdot \boldsymbol{y} \rceil$, otherwise put $\min(\boldsymbol{a} \cdot \boldsymbol{x}, P) = -\infty$.

3 Worked Example from String Buffer Analysis

In this section we explain the ideas behind widening with landmarks in the context of an example drawn from string buffer analysis. Consider the following loop which is naturally produced by a C compiler translating `while (*s) s++;`.

```
char s[32] = "the␣string";
int i = 0;
while (true) {
  c = s[i];
  if (c==0) break;
  i = i+1;
};
```

The task is to check that the string buffer `s` is only accessed within bounds. This program is challenging for automatic verification because the loop invariant is always satisfied and the extra exit condition within the loop does not mention the loop counter `i`. In C, a string is merely an array of bytes, in this case `s` is an array of 32 bytes. The string literal initialises the first ten characters whilst the eleventh position is set to 0 (the NUL character). The analysis of this function follows the ideas of [11, 20, 25] in representing only the position of the first NUL character, thereby ignoring the content of `"the␣string"`. Thus, a single variable per array suffices to express the relevant information. Specifically, let n represent the index of the NUL position in `s`. The control flow graph of the string buffer example is decorated with polyhedra P, Q, R, S, T, U as follows:



The initial values of the program variables is described by $P = [\![\{i = 0, n = 10\}]\!]$. The merge of this polyhedron and the polyhedron on the back edge, U, defines $Q = P \sqcup U$. To verify that the array access `s[i]` is within bounds, we compute $Q' = Q \sqcap [\![\{0 \leq i \leq 31\}]\!]$ and issue a warning if $Q' \neq Q$. The analysis continues under the premise that the access was within bounds and hence R is defined in terms of Q' rather than Q as follows:

$$\begin{aligned} R = &(\exists_c(Q') \sqcap [\![\{i \leq n-1, 1 \leq c \leq 255\}]\!]) \\ &\sqcup (\exists_c(Q') \sqcap [\![\{i = n, c = 0\}]\!]) \\ &\sqcup (\exists_c(Q') \sqcap [\![\{n+1 \leq i, 0 \leq c \leq 255\}]\!]) \end{aligned}$$

The projection operator $\exists_c$ removes all information pertaining to c in Q' so that c can be updated. Since the contents of `s` are ignored in our model, the new value of c only depends on the relationship between the index i and n which describes the position of the *first* NUL character. The value of c is restricted to

$[1, 255]$ if $i < n$, it is set to 0 if $i = n$ and to $[0, 255]$ if $i > n$. Note that this model is valid for platforms where the C `char` type is unsigned. The last three equations that comprise the system are given by the following:

$$S = R \sqcap [\![\{c = 0\}]\!]$$
$$T = (R \sqcap [\![\{c \leq -1\}]\!]) \sqcup (R \sqcap [\![\{c \geq 1\}]\!])$$
$$U = \{\langle n, i + 1, c\rangle \mid \langle n, i, c\rangle \in T\}$$

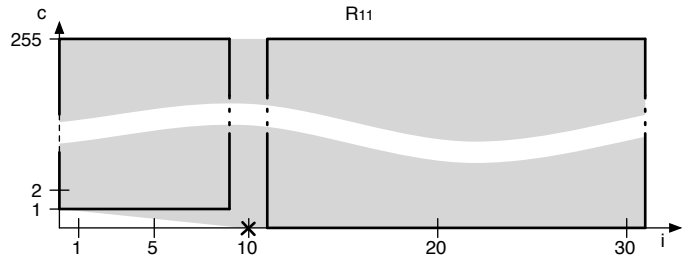
The affine transformation in the last equation defining U assumes that the variables in the polyhedron are ordered as in the sequence n, i, c.

3.1 Applying the Widening/Narrowing Approach

As before, we solve these equations iteratively, nominating Q as the widening point to ensure convergence in the cycle Q, R, T, U. Thus, when the equations are reinterpreted iteratively, the equation Q is replaced with $Q_{j+1} = Q_j \nabla (P_j \sqcup U_j)$. Applying the standard widening/narrowing approach results in the iterates shown in Figure 1. Again, we apply widening when Q is evaluated the third time, so that widening is applied on Q_9 and $P_9 \sqcup U_9$ to obtain Q_{10}. The resulting polyhedron $Q_{10} = [\![\{0 \leq i\}]\!]$ is intersected with the verification condition to yield $Q'_{10} = [\![\{0 \leq i \leq 31\}]\!]$, thereby raising a warning since $Q_{10} \neq Q'_{10}$. Before proceeding to the evaluation of R_{11}, observe that $\exists_c(Q'_j) = Q'_j$ in all iterations j since P_j does not constrain c and consequently neither does $Q_j = U_j \sqcup P_j$. Given that Q'_{10} allows i to take on any value in $[0, 31]$, the three cases in the definition of R that are guarded by $i \leq n - 1$, $i = n$ and $n + 1 \leq i$ all contribute to the result R_{11}. This result is depicted as the grey region in Figure 1 which shows the relationship between i and c. The three regions whose join form the polyhedron R_{11} are marked with two rectangles and a small cross for the $c = 0$ case. Observe that applying narrowing, that is, replacing $Q_{j+1} = Q_j \nabla (U_j \sqcup P_j)$ with $Q_{j+1} = Q_j \triangle (U_j \sqcup P_j)$, yields another iterate $1'$ in which the value of i ranges over $[0, 32]$ which still violates the array bound check since $Q'_{1'} \neq Q_{1'}$ where $Q'_{1'} = Q_{1'} \sqcap \{0 \leq i \leq 31\}$ corresponds to $Q_{1'}$ restricted to valid array indices.

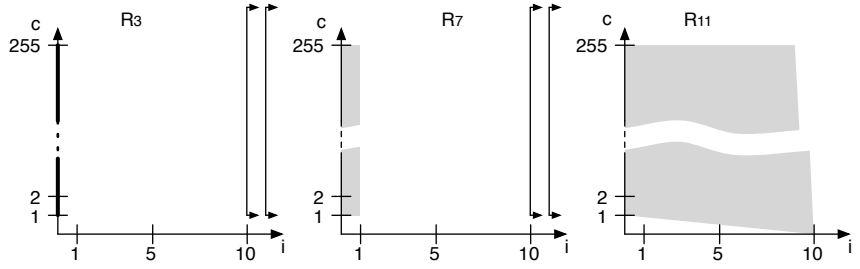
3.2 The Rationale behind Landmarks

Now consider the same fixpoint calculation using widening with landmarks as shown in Figure 2. We omit the first nine iterates before widening is applied since they coincide with those given in Figure 1. While landmarks are gathered throughout the fixpoint calculation, we focus on the calculation of the polyhedron R as this gives rise to the only landmarks that are of relevance in this example. The three graphs in Figure 2 depict the relation between i and c in the polyhedra R_3, R_7, R_{11}, which are the three iterates in which R_j changes. The polyhedron R_3 is derived from $\exists_c(Q'_3) = Q'_3 = [\![\{0 \leq i \leq 0\}]\!]$. During this computation, Q'_3 is intersected with $[\![\{i \leq n-1, 1 \leq c \leq 255\}]\!]$, $[\![\{n \leq i \leq n, 0 \leq c \leq 0\}]\!]$ and $[\![\{n+1 \leq i, 0 \leq c \leq 255\}]\!]$ which represent three different behaviours of the program.



j	Q_j	R_j	S_j	T_j	U_j
1	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
2	$\{\mathbf{0 \le i \le 0}\}$	$\bot$	$\bot$	$\bot$	$\bot$
3	$\{0 \le i \le 0\}$	$\left\{\begin{array}{l}\mathbf{0 \le i \le 0},\\ \mathbf{1 \le c \le 255}\end{array}\right\}$	$\bot$	$\bot$	$\bot$
4	$\{0 \le i \le 0\}$	$\left\{\begin{array}{l}0 \le i \le 0,\\ 1 \le c \le 255\end{array}\right\}$	$\bot$	$\left\{\begin{array}{l}\mathbf{0 \le i \le 0},\\ \mathbf{1 \le c \le 255}\end{array}\right\}$	$\bot$
5	$\{0 \le i \le 0\}$	$\left\{\begin{array}{l}0 \le i \le 0,\\ 1 \le c \le 255\end{array}\right\}$	$\bot$	$\left\{\begin{array}{l}0 \le i \le 0,\\ 1 \le c \le 255\end{array}\right\}$	$\left\{\begin{array}{l}\mathbf{1 \le i \le 1},\\ \mathbf{1 \le c \le 255}\end{array}\right\}$
6	$\{\mathbf{0 \le i \le 1}\}$	$\left\{\begin{array}{l}0 \le i \le 0,\\ 1 \le c \le 255\end{array}\right\}$	$\bot$	$\left\{\begin{array}{l}0 \le i \le 0,\\ 1 \le c \le 255\end{array}\right\}$	$\left\{\begin{array}{l}1 \le i \le 1,\\ 1 \le c \le 255\end{array}\right\}$
7	$\{0 \le i \le 1\}$	$\left\{\begin{array}{l}\mathbf{0 \le i \le 1},\\ \mathbf{1 \le c \le 255}\end{array}\right\}$	$\bot$	$\left\{\begin{array}{l}0 \le i \le 0,\\ 1 \le c \le 255\end{array}\right\}$	$\left\{\begin{array}{l}1 \le i \le 1,\\ 1 \le c \le 255\end{array}\right\}$
8	$\{0 \le i \le 1\}$	$\left\{\begin{array}{l}0 \le i \le 1,\\ 1 \le c \le 255\end{array}\right\}$	$\bot$	$\left\{\begin{array}{l}\mathbf{0 \le i \le 1},\\ \mathbf{1 \le c \le 255}\end{array}\right\}$	$\left\{\begin{array}{l}1 \le i \le 1,\\ 1 \le c \le 255\end{array}\right\}$
9	$\{0 \le i \le 1\}$	$\left\{\begin{array}{l}0 \le i \le 1,\\ 1 \le c \le 255\end{array}\right\}$	$\bot$	$\left\{\begin{array}{l}0 \le i \le 1,\\ 1 \le c \le 255\end{array}\right\}$	$\left\{\begin{array}{l}\mathbf{1 \le i \le 2},\\ \mathbf{1 \le c \le 255}\end{array}\right\}$
10	$\{\mathbf{0 \le i}\}$	$\left\{\begin{array}{l}0 \le i \le 1,\\ 1 \le c \le 255\end{array}\right\}$	$\bot$	$\left\{\begin{array}{l}0 \le i \le 1,\\ 1 \le c \le 255\end{array}\right\}$	$\left\{\begin{array}{l}1 \le i \le 2,\\ 1 \le c \le 255\end{array}\right\}$
11	$\{0 \le i\}$	$\left\{\begin{array}{l}\mathbf{0 \le i \le 31},\\ \mathbf{0 \le c \le 255},\\ \mathbf{-i - 10c \le -10}\end{array}\right\}$	$\bot$	$\left\{\begin{array}{l}0 \le i \le 1,\\ 1 \le c \le 255\end{array}\right\}$	$\left\{\begin{array}{l}1 \le i \le 2,\\ 1 \le c \le 255\end{array}\right\}$
12	$\{0 \le i\}$	$\left\{\begin{array}{l}0 \le i \le 31,\\ 0 \le c \le 255,\\ -i - 10c \le -10\end{array}\right\}$	$\left\{\begin{array}{l}\mathbf{10 \le i},\\ \mathbf{i \le 31},\\ \mathbf{0 \le c \le 0}\end{array}\right\}$	$\left\{\begin{array}{l}\mathbf{0 \le i \le 31},\\ \mathbf{1 \le c \le 255}\end{array}\right\}$	$\left\{\begin{array}{l}1 \le i \le 2,\\ 1 \le c \le 255\end{array}\right\}$
13	$\{0 \le i\}$	$\left\{\begin{array}{l}0 \le i \le 31,\\ 0 \le c \le 255,\\ -i - 10c \le -10\end{array}\right\}$	$\left\{\begin{array}{l}10 \le i,\\ i \le 31,\\ 0 \le c \le 0\end{array}\right\}$	$\left\{\begin{array}{l}0 \le i \le 31,\\ 1 \le c \le 255\end{array}\right\}$	$\left\{\begin{array}{l}\mathbf{1 \le i \le 32},\\ \mathbf{1 \le c \le 255}\end{array}\right\}$
14	$\{0 \le i\}$	$\left\{\begin{array}{l}0 \le i \le 31,\\ 0 \le c \le 255,\\ -i - 10c \le -10\end{array}\right\}$	$\left\{\begin{array}{l}10 \le i,\\ i \le 31,\\ 0 \le c \le 0\end{array}\right\}$	$\left\{\begin{array}{l}0 \le i \le 31,\\ 1 \le c \le 255\end{array}\right\}$	$\left\{\begin{array}{l}1 \le i \le 32,\\ 1 \le c \le 255\end{array}\right\}$
1'	$\{\mathbf{0 \le i \le 32}\}$	$\left\{\begin{array}{l}0 \le i \le 31,\\ 0 \le c \le 255,\\ -i - 10c \le -10\end{array}\right\}$	$\left\{\begin{array}{l}10 \le i,\\ i \le 31,\\ 0 \le c \le 0\end{array}\right\}$	$\left\{\begin{array}{l}0 \le i \le 31,\\ 1 \le c \le 255\end{array}\right\}$	$\left\{\begin{array}{l}1 \le i \le 31,\\ 1 \le c \le 255\end{array}\right\}$

Fig. 1. Fixpoint calculation of the string loop. A polyhedron $[\![S]\!]$ is abbreviated to S and $\bot$ denotes an unsatisfiable set of inequalities. The column P_j is omitted since $P_j = [\![\{0 \le i \le 0\}]\!]$ for all iterations j. Further we omit $10 \le n \le 10$ from all polyhedra.



j	Q_j	R_j	S_j	T_j	U_j
10	$\{\mathbf{0 \le i \le 10}\}$	$\left\{\begin{array}{l} 0 \le i \le 1, \\ 1 \le c \le 255 \end{array}\right\}$	$\bot$	$\left\{\begin{array}{l} 0 \le i \le 1, \\ 1 \le c \le 255 \end{array}\right\}$	$\left\{\begin{array}{l} 1 \le i \le 2, \\ 1 \le c \le 255 \end{array}\right\}$
11	$\{0 \le i \le 10\}$	$\left\{\begin{array}{l} \mathbf{0 \le i}, \\ \mathbf{c \le 255}, \\ \mathbf{255i + c \le 2550}, \\ \mathbf{-i - 10c \le -10} \end{array}\right\}$	$\bot$	$\left\{\begin{array}{l} 0 \le i \le 1, \\ 1 \le c \le 255 \end{array}\right\}$	$\left\{\begin{array}{l} 1 \le i \le 2, \\ 1 \le c \le 255 \end{array}\right\}$
12	$\{0 \le i \le 10\}$	$\left\{\begin{array}{l} 0 \le i, \\ c \le 255, \\ 255i + c \le 2550, \\ -i - 10c \le -10 \end{array}\right\}$	$\left\{\begin{array}{l} \mathbf{10 \le i}, \\ \mathbf{i \le 10}, \\ \mathbf{0 \le c}, \\ \mathbf{c \le 0} \end{array}\right\}$	$\left\{\begin{array}{l} \mathbf{0 \le i \le 9}, \\ \mathbf{1 \le c \le 255} \end{array}\right\}$	$\left\{\begin{array}{l} 1 \le i \le 2, \\ 1 \le c \le 255 \end{array}\right\}$
13	$\{0 \le i \le 10\}$	$\left\{\begin{array}{l} 0 \le i, \\ c \le 255, \\ 255i + c \le 2550, \\ -i - 10c \le -10 \end{array}\right\}$	$\left\{\begin{array}{l} 10 \le i, \\ i \le 10, \\ 0 \le c, \\ c \le 0 \end{array}\right\}$	$\left\{\begin{array}{l} 0 \le i \le 9, \\ 1 \le c \le 255 \end{array}\right\}$	$\left\{\begin{array}{l} \mathbf{1 \le i \le 10}, \\ \mathbf{1 \le c \le 255} \end{array}\right\}$
14	$\{0 \le i \le 10\}$	$\left\{\begin{array}{l} 0 \le i, \\ c \le 255, \\ 255i + c \le 2550, \\ -i - 10c \le -10 \end{array}\right\}$	$\left\{\begin{array}{l} 10 \le i, \\ i \le 10, \\ 0 \le c, \\ c \le 0 \end{array}\right\}$	$\left\{\begin{array}{l} 0 \le i \le 9, \\ 1 \le c \le 255 \end{array}\right\}$	$\left\{\begin{array}{l} 1 \le i \le 10, \\ 1 \le c \le 255 \end{array}\right\}$

Fig. 2. Fixpoint calculation using widening with landmarks.

As the fixpoint calculation progresses, polyhedra grow and new behaviours are incrementally enabled. A behaviour can only change from being disabled to being enabled when one of its constituent inequalities makes the transition from unsatisfiable to satisfiable. A fixpoint may exist in which not all behaviours of a program are enabled, that is, there are behaviours that contain unsatisfiable inequalities. The rationale for widening with landmarks is to find these fixpoints by systematically considering the inequalities that prevent a behaviour from being enabled. These inequalities are exactly those inequalities in the semantic equations that are unsatisfiable in the context of the current iterate. In the example, the last two behaviours contain the inequalities $n \le i$ (arising from $i = n$) and $n + 1 \le i$ that are responsible for enabling the second and third behaviour. These inequalities are unsatisfiable in Q'_3 and are therefore stored as landmarks. The leftmost graph in Figure 2 indicates the position of the two inequalities $n \le i$ and $n + 1 \le i$ which define the landmarks we record for R_3.

3.3 Creating Landmarks for Widening

A landmark is a triple comprised of an inequality and two distances. On creation, the first distance is set to the shortest straight-line distance the inequality must be translated so as to touch the current iterate. In this example, translations by 10 and 11 units are required for $n \leq i$ and $n + 1 \leq i$, respectively, to touch R_3.

In the 7th iteration, when R_j is updated again, a second measurement is taken between the inequality and the new iterate. This distance is recorded as the second distance in the existing landmark. In the example, the second distance for the landmarks for $n \leq i$ and $n + 1 \leq i$ is set to 9 and 10 units, respectively.

By iteration 8 both landmarks have acquired a second measurement, however, it is not until widening is applied in iteration 10 that the landmarks are actually used. The difference between the two measurements of a particular landmark indicates how fast the iterates R_j are approaching the as-of-yet unsatisfiable inequality of that landmark. From this difference we estimate how many times R_j must be updated until the inequality becomes satisfiable. In the example, the difference in distance between the two updates R_3 and R_7 is one unit for each landmark. Thus, at this rate, R_j would be updated 9 more times until the closer inequality, namely $n \leq i$, becomes satisfied. Rather than calculating all these intermediate iterates, we use this information to perform an extrapolation step when the widening point Q is revisited.

3.4 Using Landmarks in Widening

From the perspective of the widening operator, the task is, firstly, to gather all landmarks that have been generated in the traversal of the cycle in which the widening operator resides. Secondly, the widening operator ranks the landmarks by the number of iterations needed for the corresponding inequality to become satisfied. Thirdly, the landmark with the smallest rank determines the amount of extrapolation the widening operator applies. In the example, recall that the unsatisfiable inequality $n \leq i$ in R_7 would become satisfiable after 9 more updates of R whereas the other unsatisfiable inequality $n+1 \leq i$ becomes satisfiable after 10 updates. Hence, $n \leq i$ constitutes the nearest inequality and, rather than applying widening when calculating $Q_{10} = Q_9 \nabla (P_9 \sqcup U_9)$, extrapolation is performed. Specifically, the changes between $Q_9 = [\![\{0 \leq i \leq 1\}]\!]$ and $P_9 \sqcup U_9 = [\![\{0 \leq i \leq 2\}]\!]$ are extrapolated 9 times to yield $Q_{10} = [\![\{0 \leq i \leq 10\}]\!]$. The new value of Q_{10} forces a re-evaluation of R, yielding R_{11}, as shown in Figure 2. In the next iteration, the semantic equation for T yields $[\![\{0 \leq i, 1 \leq c \leq 255, 255i + c \leq 2550\}]\!]$. Since i and c are known to be integral, this polyhedron can be refined [16] to the entry T_{12} as shown in the table. A final iteration leads to a fixpoint.

Note that it is possible to apply extrapolation as soon as a single landmark acquires its second measurement. However, to ensure that the state is extrapolated only to the point where the first additional behaviour becomes enabled, the extrapolation step should be deferred until all landmarks have acquired their second value. In practise, this means that no extrapolation is performed if a new landmark was created in the last iteration. Note that new landmarks cannot

Listing 1 Adding or tightening a landmark: $updateLandmark(P, \iota, L)$

Require: $P \in Poly, \iota \in Ineq, L \subseteq Lin \times \mathbb{Z} \times (\mathbb{Z} \cup \{\infty\})$
1: $e \leq c \leftarrow \iota$
2: $c' \leftarrow \min(P, e)$
3: **if** $c < c'$ **then** /* $P \sqcap [\![\{\iota\}]\!]$ is empty */
4: $dist \leftarrow c' - c$ /* calculate the distance between P and $[\![\{\iota\}]\!]$ */
5: **if** $\exists dist_c, dist_p \,.\, \langle e, dist_c, dist_p\rangle \in L$ **then**
6: **return** $(L \setminus \{\langle e, dist_c, dist_p\rangle\}) \cup \{\langle e, \min(dist, dist_c), dist_p\rangle\}$
7: **else**
8: **return** $L \cup \{\langle e, dist, \infty\rangle\}$
9: **end if**
10: **end if**
11: **return** L

be added indefinitely as there is at most one landmark for each inequality that occurs in the semantic equations which are, in turn, finite.

The following sections formalise these ideas by presenting algorithms for gathering landmarks and performing extrapolation using landmarks.

4 Acquiring Landmarks

This section formalises the intuition behind widening with landmarks by giving a more algorithmic description on how landmarks are acquired. Listing 1 presents the algorithm *updateLandmark* which is invoked whenever a polyhedron P is intersected with an inequality ι that arises from a semantic equation. In line 2, the distance between ι and P is measured by calculating $c' = \min(P, e)$. Intuitively, $e \leq c'$ is a parallel translation of ι that has a minimal intersection with P. Line 3 compares the relative location of ι and its translation, thereby ensuring that lines 4 to 9 are only executed if ι is unsatisfiable and, thus, can yield a landmark. If ι is indeed unsatisfiable, line 4 calculates its distance to P.

Given this distance, line 5 determines if a landmark is to be updated or created. An update occurs whenever different semantic equations contain the same unsatisfiable inequality. In this case line 6 ensures that the smaller distance is stored in the landmark. The rationale for storing the distance to the closer inequality is that a landmark for the inequality that is further away can be gathered later. In particular, if extrapolation to the nearer inequality does not lead to a fixpoint, the nearer inequality is satisfiable in the extrapolated space and cannot induce a new landmark. At this point the inequality that is further away can become a landmark. Hence, tracking distances to closer inequalities ensures that all landmarks are considered in turn.

When creating a new landmark, line 8 sets the second distance to infinity which indicates that this new landmark is not yet ready to be used in extrapolation. The next section details how the acquired landmarks are manipulated.

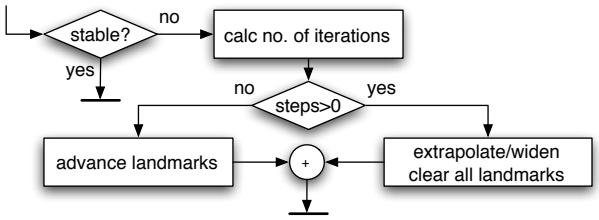


Fig. 3. Operations performed at a widening point.

Listing 2 Advance a landmark: *advanceLandmarks*(L)

Require: $L \subseteq Lin \times \mathbb{Z} \times (\mathbb{Z} \cup \{\infty\})$
1: $L' \leftarrow \{\langle e, dist_c, dist_c\rangle \mid \langle e, dist_c, dist_p\rangle \in L\}$
2: **return** L'

5 Using Landmarks at a Widening Point

The semantic equations of the program induce cyclic dependencies between the states at each program point. A widening point must be inserted into each cycle to ensure that the fixpoint computation eventually stabilises. In case of nested cycles, a fixpoint is calculated on each inner cycle before moving on to the containing cycle [6]. Figure 3 schematically shows the actions taken when a semantic equation at a widening point is evaluated. If stability has not yet been achieved, all landmarks gathered in the current cycle (excluding those in inner cycles) are passed to the algorithm *calcIterations* which estimates the number of times the cycle needs to be traversed until a state is reached at which the first as-of-yet unsatisfiable inequality becomes satisfiable. This count is denoted as *steps* in Figure 3. Two special values are distinguished: 0 and ∞. A value of zero indicates that new landmarks were created during the last traversal of the cycle. In this case, the left branch of Figure 3 is taken and the algorithm *advanceLandmarks*, which is presented in Listing 2, is called. Normal fixpoint computation is then resumed, allowing landmarks to acquire a second measurement. The call to *advanceLandmarks* stores the calculated distance in the third element of each landmark, thereby ensuring that this value is not lost when *updateLandmark* updates the second element of the landmark tuple during the next iteration.

The right branch of Figure 3 is selected whenever *calcIterations* returns a non-zero value for *steps* which indicates that all landmarks have acquired two measurements. This is the propitious moment for extrapolation as only now can all landmarks participate in predicting the number of cycles until the first as-of-yet unsatisfiable inequality is reached. In order to show how this number is derived, consider Listing 3. The algorithm *calcIterations* calculates an estimate of the number of iterations necessary to satisfy the nearest landmark stored in *steps*. This variable is initially set to ∞ which is the value returned if no landmarks have been gathered. An infinite value in *steps* indicates that

Listing 3 Calculate distance $calcIterations(L)$

Require: $L \subseteq Lin \times \mathbb{Z} \times (\mathbb{Z} \cup \{\infty\})$
1: $steps \leftarrow \infty$ /* indicate that normal widening should be applied */
2: **for** $\langle e, dist_c, dist_p \rangle \in L$ **do**
3: **if** $dist_p = \infty$ **then**
4: $steps \leftarrow 0$
5: **else if** $dist_p > dist_c$ **then**
6: $steps \leftarrow \min(steps, \lceil dist_c/(dist_p - dist_c) \rceil)$ /* assume $\min(\infty, n) = n$ */
7: **end if**
8: **end for**
9: **return** $steps$

widening, rather than extrapolation, has to be applied. Otherwise, the loop in lines 2–8 examines each landmark in turn. For any landmark with two measurements, i.e. those for which $dist_p \neq \infty$, line 6 calculates after how many steps the unsatisfiable inequality that gave rise to the landmark $\langle e, dist_c, dist_p \rangle$ becomes satisfiable. Specifically, $dist_p - dist_c$ represents the distance traversed during one iteration. Given that $dist_c$ is the distance between the boundary of the unsatisfiable inequality and the polyhedron in that iteration, the algorithm computes $\lceil dist_c/(dist_p - dist_c) \rceil$ as an estimate of the number of iterations required to make the inequality satisfiable. This number is stored in *steps* unless another landmark has already been encountered that can be reached in fewer iterations.

The next section presents an algorithm that extrapolates the change between two iterates by a given number of steps. It thereby completes the suite of algorithms necessary to realise widening with landmarks.

6 Extrapolation Operator for Polyhedra

In contrast to standard widening which removes inequalities that are unstable, extrapolation by a finite number of steps merely relaxes inequalities until the next landmark is reached. Listing 4 presents a simple extrapolation algorithm that performs this relaxation based on two iterates, namely P_1 and P_2. This extrapolation is applied by replacing any semantic equation of the form $Q_{i+1} = Q_i \nabla R_i$ with $Q_{i+1} = extrapolate(Q_i, R_i, steps)$ where $steps = calcIterations(L)$ and L is the set of landmarks relevant to this widening point. Thus the first argument to *extrapolate*, namely P_1, corresponds to the previous iterate Q_i, while P_2 corresponds to R_i. Line 2 calculates the join P of both, P_1 and P_2, which forms the basis for extrapolating the polyhedron P_1. Specifically, bounds of P_1 that are not preserved in the join are extrapolated. The loop in lines 7–15 implements this strategy which resembles the original widening on polyhedra [10] which can be defined as $E_{res} = \{\iota_i \mid P \sqsubseteq [\![\{\iota_i\}]\!]\}$ where $\iota_1, \ldots \iota_n$ is a non-redundant set of inequalities such that $[\![\{\iota_1, \ldots \iota_n\}]\!] = P_1$, c.f. [1]. Note that this widening can be inaccurate if the dimensionality of P_1 is smaller than that of $P = P_1 \sqcup P_2$; other inequalities from P can be added to E_{res} to remedy this [1, 14] but we omit this additional step for brevity. The entailment check $P \sqsubseteq [\![\{\iota_i\}]\!]$ for $\iota_i \equiv e \leq c$ is

Listing 4 Extrapolate changes *extrapolate*$(P_1, P_2, steps)$

Require: $P_1, P_2 \in Poly, steps \in \mathbb{N} \cup \{\infty\}$
1: $[\![\iota_1, \ldots \iota_n]\!] \leftarrow P_1$ /* $\iota_1, \ldots \iota_n$ is a non-redundant description of P_1 */
2: $P \leftarrow P_1 \sqcup P_2$
3: **if** $steps = 0$ **then**
4: **return** P
5: **else**
6: $Eres \leftarrow \emptyset$
7: **for** $i = 1, \ldots n$ **do**
8: $e \leq c \leftarrow \iota_i$
9: $c' \leftarrow \min(P, e)$
10: **if** $c' \leq c$ **then**
11: $Eres \leftarrow Eres \cup \{e \leq c\}$ /* since $P \sqsubseteq [\![\iota_i]\!]$ */
12: **else if** $steps \neq \infty$ **then**
13: $Eres \leftarrow Eres \cup \{e \leq (c + (c' - c)steps)\}$
14: **end if**
15: **end for**
16: **return** $[\![Eres]\!]$
17: **end if**

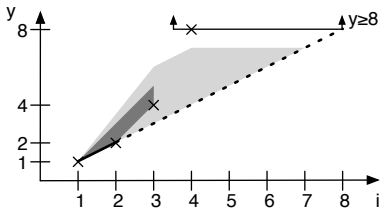


Fig. 4. Illustrating non-linear growth.

implemented in line 9 by calculating the smallest c' such that $P \sqsubseteq [\![\{e \leq c'\}]\!]$. In the case that $c' \leq c$, the entailment holds and line 11 adds the inequality to the result set. In the case that the entailment does not hold, the inequality is discarded whenever $steps = \infty$. In this case *extrapolate* reduces to a simple widening. If *steps* is finite, line 13 translates the inequality, thereby anticipating the change that is likely to occur during the next *steps* loop iterations.

The presented algorithm performs a linear translation of inequalities. Since array accesses are typically linear, this approach is well suited for verifying that indices fall within bounds. However, a non-linear relationship such as that arising in the C loop `int i=1; for(int y=1; y<8; y=y*2) i++;` is not amenable to linear extrapolation and thus leads to a loss of precision. The loop creates successive values for `i`, `y` that correspond to the points $\langle 1, 1\rangle$, $\langle 2, 2\rangle$, $\langle 3, 4\rangle$ and, finally, at the exit of the loop, the point $\langle 4, 8\rangle$. These are indicated as crosses in Figure 4. The best polyhedral approximation of these points restricted by the loop invariant $y < 8$ is shown in dark grey. However, extrapolating the first two iterates, namely the polyhedron $\{\langle 1, 1\rangle\}$ and the polyhedron that additionally

contains $\langle 2, 2\rangle$, predicts that the shown landmark $y \geq 8$ becomes satisfiable after 7 additional loop iterations. The extrapolation results in the state depicted as a dashed line; continuing the fixpoint calculation leads to the light grey area as loop invariant which is a coarser approximation than the optimal polyhedron.

7 Implementation

We have implemented widening with landmarks in a verifier for C programs that combines numeric analysis with points-to analysis. The verifier is geared towards string buffer analysis in that it implements the tracking of NUL positions [20]. One obstacle in a polyhedral analysis is the complexity of the polyhedral operations. To support a large number of variables, we chose the two-variable-per-inequality (TVPI) domain [22] which can only represent inequalities with at most two non-zero variables per inequality. The underlying idea in this domain is to calculate a closure of the TVPI inequalities. A closure step eliminates a from any two (appropriately scaled) inequalities $ax_i + bx_j \leq c$ and $-ax_i + dx_k \leq e$ to obtain $bx_j + dx_k \leq c + e$ which is then added to the closure. A closed system makes it possible to implement all polyhedral operations efficiently on sets of planar polyhedra. For example, the convex hull operation on planar polyhedra runs in $O(n \log n)$ where n is the number of inequalities in the planar polyhedron [22]. Another advantage is the availability of algorithms to shrink each planar polyhedron around the integral grid [16]. This is not only useful to improve precision when analysing integer variables (as necessary in T_{12} of Figure 2) but also limits the size of coefficients of inequalities. Otherwise, inequalities with excessively large coefficients have to be removed to ensure progress [21], a step that is impossible in the TVPI domain since closure could re-introduce these inequalities.

Implementing widening with landmarks requires two modifications to an existing analysis, namely modifying the intersection operation to gather landmarks and replacing widening operators by extrapolation operations that evaluate the acquired landmarks. When it comes to gathering landmarks, note that the TVPI domain implements intersection of a polyhedron P and a set of inequalities $\{\iota_1, \ldots \iota_n\}$ by computing $(\ldots((P \sqcap \{\iota_1\}) \sqcap \{\iota_2\}) \ldots) \sqcap \{\iota_n\}$ since a cheap incremental closure can be applied after adding a single inequality. Adding inequalities one-by-one makes it possible to intersperse calls to *updateLandmark* for landmark acquisition. While it may seem that calculating $\min(e, P)$ in line 2 of Listing 1 incurs a performance penalty, it turns out that running this linear program can actually improve memory performance of a domain. Consider the semantic equation of R from Section 3 whose calculation requires three copies of the input polyhedron $\exists_c(Q')$ which are then intersected with inequalities expressing three different behaviours. During the fixpoint calculation, many behaviours are disabled. A consequence of this is that a polyhedron will be copied, only for the copy to become unsatisfiable when intersected with an inequality that expresses such a disabled behaviour. Observe that in this case the test on line 3 of *updateLandmark* succeeds and a landmark is added. A better strategy is to call *updateLandmark* without copying the original input polyhedron and, only if no

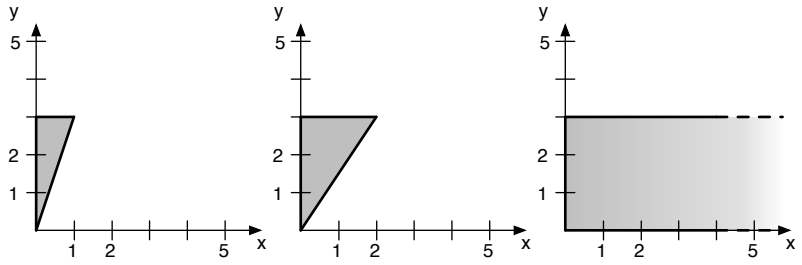


Fig. 5. Improved widening from polytopes to polyhedra.

new landmarks arise, an actual copy of the input polyhedron is needed further processing. This strategy avoids copying polyhedra that are shortly after abandoned when they turn unsatisfiable. Note that this refinement can be applied to many semantic equations, in particular, to those that model conditionals which, in our application, make up the majority of all intersection operations.

8 Related Work

Although the foundations of widening and narrowing were laid three decades ago [7], the value of widening was largely unappreciated until comparatively recently [9]. In the last decade there has been a resurgence of interest in applying polyhedral analysis and, specifically, polyhedral widenings [1, 3, 4]. The original widening operator in [10] discards linear relationships that result from joining the state of the previous loop iteration with the current loop iteration. This causes a loss of precision, especially when widening is applied in each loop iteration. The so-called revised widening [14] remedies this by adding additional inequalities from the join. Benoy [3] showed that the two widenings coincide whenever widening is postponed until the dimensionality of the iterates has stabilised.

Besson et al. [4] present widenings that are especially precise when widening polytopes into polyhedra. For instance, the iterates shown in Figure 5 feature an inequality with changing coefficients that standard widening would remove. Instead, this inequality is widened to $y \geq 0$, thereby retaining a lower bound on y. Extending our extrapolation function to include inequalities with changing coefficients is an interesting research question. Bagnara et al. [1] combine the techniques of Besson et al. and other widenings with extrapolation strategies that delay widening. More closely related is work on extrapolation using information from the analysed equation system. For instance, widening with thresholds [5] uses a sequence of user-specified values (thresholds) on individual variables up to which the state space is extrapolated in sequence. Halbwachs et al. [15] deduce thresholds automatically from guards in the semantic equations. However, they observe redundant inequalities rather than unsatisfiable inequalities, thereby possibly extrapolating to thresholds where no fixpoint can exist, such as redundant inequalities that express verification conditions. The restriction of inferring thresholds on single variables is lifted by lookahead widening which uses standard widening and narrowing operators and thereby is able to

find bounds that are expressed with more than one variable [12]. It uses a pilot polyhedron on which widening and narrowing is performed alongside a main polyhedron. Once the pilot value has stabilised after narrowing, it is promoted to become the main value. By using the main value to evaluate effects in other domains, the problems of domain interaction as discussed in Section 1 do not occur. Furthermore, by discarding behaviours that are enabled after widening but are disabled with respect to the main value, their approach is able to find fixpoints in which not all behaviours are enabled, such as the one in the example on string buffers. While their approach solves essentially the same problem as widening with landmarks, the analysis operates on two polyhedra instead of one.

Further afield is the technique of counterexample-driven refinement that has recently been adapted to polyhedral analysis [13]. This approach is in some sense orthogonal to narrowing that refines a single fixpoint. In counterexample-driven refinement, the fixpoint computation is repeatedly restarted, guided by a backwards analysis from the point of a false warning to some widening point. Finally, it has been shown that widening and narrowing can be avoided altogether in a relational analysis if the semantic equations are affine [24]. Incredibly, for this restricted class of equations, least fixpoints can be found in polynomial time.

9 Conclusion

Motivated by shortcomings encountered in narrowing polyhedra, this paper proposes an extrapolation technique called widening with landmarks. The idea is to reason about unsatisfiable inequalities to guide the extrapolation process. This tactic is sensitive to invariants that are not obvious from the loop condition.

Acknowledgements We would like to thank Denis Gopan for useful discussions on lookahead widening. This work was supported by EPSRC project EP/C015517.

References

1. R. Bagnara, P. Hill, E. Ricci, and E. Zaffanella. Precise Widening Operators for Convex Polyhedra. *Science of Computer Programming*, 58(1–2):28–56, 2005.
2. R. Bagnara, P. M. Hill, and E. Zaffanella. Not necessarily closed convex polyhedra and the double desciption method. *Formal Asp. Comput.*, 17(2):222–257, 2005.
3. P. M. Benoy. *Polyhedral Domains for Abstract Interpretation in Logic Programming*. PhD thesis, Computing Lab., Univ. of Kent, Canterbury, UK, January 2002.
4. F. Besson, T. P. Jensen, and J.-P. Talpin. Polyhedral Analysis for Synchronous Languages. In *SAS*, number 1694 in LNCS, pages 51–68, Venice, Italy, 1999.
5. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Mine, D. Monniaux, and X. Rival. Design and Implementation of a Special-Purpose Static Program Analyzer for Safety-Critical Real-Time Embedded Software. In *The Essence of Computation: Complexity, Analysis, Transformation*, number 2566 in LNCS, pages 85–108. Springer Verlag, 2002.
6. F. Bourdoncle. Efficient Chaotic Iteration Strategies with Widenings. In *International Conference on Formal Methods in Programming and their Applications*, volume 735 of *LNCS*, pages 128–141. Springer Verlag, 1993.

7. P. Cousot and R. Cousot. Static Determination of Dynamic Properties of Programs. In *Second International Symposium on Programming*, pages 106–130. Dunod, Paris, France, 1976.
8. P. Cousot and R. Cousot. Abstract Interpretation and Application to Logic Programs. *Journal of Logic Programming*, 13(2–3):103–179, 1992.
9. P. Cousot and R. Cousot. Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation. In M. Bruynooghe and M. Wirsing, editors, *International Symposium on Programming Language Implementation and Logic Programming*, volume 631 of *LNCS*, pages 269–295. Springer-Verlag, 1992.
10. P. Cousot and N. Halbwachs. Automatic Discovery of Linear Constraints among Variables of a Program. In *Symposium on Principles of Programming Languages*, pages 84–97, Tucson, Arizona, 1978. ACM Press.
11. N. Dor, M. Rodeh, and M. Sagiv. Cleanness Checking of String Manipulations in C Programs via Integer Analysis. In P. Cousot, editor, *Static Analysis Symposium*, number 2126 in LNCS, pages 194–212, Paris, France, 2001. Springer-Verlag.
12. D. Gopan and T. Reps. Lookahead Widening. In *CAV*, volume 4144. Springer, 2006. To appear.
13. B. S. Gulavani and S. K. Rajamani. Counterexample Driven Refinement for Abstract Interpretation. In *TACAS*, volume 3920 of *LNCS*, pages 474–488. Springer, April 2006.
14. N. Halbwachs. *Détermination Automatique de Relations Linéaires Vérifiées par les Variables d'un Programme*. Thèse de 3ème cicle d'informatique, Université scientifique et médicale de Grenoble, Grenoble, France, March 1979.
15. N. Halbwachs, Y.-E. Proy, and P. Roumanoff. Verification of Real-Time Systems using Linear Relation Analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.
16. W. Harvey. Computing Two-Dimensional Integer Hulls. *SIAM Journal on Computing*, 28(6):2285–2299, 1999.
17. N. Heintze and O. Tardieu. Ultra-fast Aliasing Analysis using CLA: A Million Lines of C Code in a Second. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 254–263, 2001.
18. A. Miné. The Octagon Abstract Domain. In *Eighth Working Conference on Reverse Engineering*, pages 310–319. IEEE Computer Society, 2001.
19. S. Sankaranarayanan, M. Colón, H. B. Sipma, and Z. Manna. Efficient Strongly Relational Polyhedral Analysis. In *VMCAI*, pages 111–125, 2006.
20. A. Simon and A. King. Analyzing String Buffers in C. In *Algebraic Methodology and Software Technology*, number 2422 in LNCS, pages 365–379. Springer, 2002.
21. A. Simon and A. King. Exploiting Sparsity in Polyhedral Analysis. In C. Hankin and I. Siveroni, editors, *Static Analysis Symposium*, number 3672 in LNCS, pages 336–351. Springer Verlag, September 2005.
22. A. Simon, A. King, and J. M. Howe. Two Variables per Linear Inequality as an Abstract Domain. In M. Leuschel, editor, *Logic Based Program Development and Transformation*, number 2664 in LNCS, pages 71–89. Springer, September 2002.
23. B. Steensgaard. Points-to Analysis in Almost Linear Time. In *Symposium on the Principles of Progamming Languages*, pages 32–41, 1996.
24. Z. Su and D. Wagner. A Class of Polynomially Solvable Range Constraints for Interval Analysis without Widenings. *Theor. Comput. Sci.*, 345(1):122–138, 2005.
25. D. Wagner. *Static analysis and computer security: New techniques for software assurance*. PhD thesis, University of California at Berkeley, December 2000.