SOFTWARE: PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 2016; **46**:1317–1328
Published online 4 December 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.2381

Widening with thresholds via binary search

Sol Kim[1], Kihong Heo[1], Hakjoo Oh[2,*,†] and Kwangkeun Yi[1]

[1]*Seoul National University, Seoul, Korea*
[2]*Korea University, Seoul, Korea*

SUMMARY

In this paper, we present a useful technique for implementing practical static program analyzers that use widening. Our technique aims to improve the efficiency of the conventional widening-with-thresholds technique at a small precision compromise. In static analysis, widening is used to accelerate (or converge) fixed point iterations. Unfortunately, this acceleration often comes with a significant loss in analysis precision. A standard method to improve the precision is to apply the widening with a set of thresholds. However, this technique may significantly slow down the analysis, because in practice it is commonplace to use a large set of thresholds. In worst case, the technique increases the analysis cost by the size N of the threshold set. In this paper, we propose a technique to reduce the worst case by $\log N$, by employing a binary search in the process of applying threshold values. We formalize the technique in the abstract interpretation framework and show that, by experiments with a realistic static analyzer for C, our technique considerably improves the efficiency (by 81.5%) of the existing method with a small compromise (20.9%) on the analysis precision. Copyright © 2015 John Wiley & Sons, Ltd.

Received 11 August 2015; Revised 4 November 2015; Accepted 6 November 2015

KEY WORDS: static program analyzers; abstract interpretation; widening

1. OVERVIEW

In this paper, we present a technique that can be used for implementing practical static analyzers that use widening. Our technique provides a way of achieving a better precision/cost balance than the conventional widening-with-thresholds technique. Our technique is best illustrated with an example. Consider the following example code:

```
int a[10];
int i = 0;
while (1) {
  i++;
  a[i] = 0; // safe
  if (i >= 8) break;
}
```

Suppose that we analyze the program using the interval abstract domain [1] (our technique is not limited to the interval domain; it is generally applicable regardless of abstract domains). The aim of the analysis is to prove the safety of the buffer access (`a[i]`) at line 5; that is, we would like to show that no buffer-overrun errors occur at that program point.

*Correspondence to: Hakjoo Oh, Programming Research Laboratory, College of Informatics, Korea University, Anam-dong 5-ga, Seongbuk-gu Seoul 136-713, Korea.
†E-mail: hakjoo_oh@korea.ac.kr

Widening without thresholds. If we analyze the program using the standard widening operator,[‡] we cannot prove the safety. With this widening operator, the loop is analyzed as follows:

Lines	Iterations		
	1	2	3
3	$[0,0]$	$[0,+\infty]$	$[0,+\infty]$
4, 5	$[1,1]$	$[1,+\infty]$	$[1,+\infty]$
6	$[1,1]$	$[1,7]$	$[1,7]$

At the first iteration of the loop, variable i has interval $[0,0]$ at the entry of the loop (line 3). At lines 4 and 5, it increments to $[1,1]$. At the second iteration of the loop, the value of `i` at line 3 becomes $[0,+\infty]$, because the value $[1,1]$ from line 6 is combined with the initial value $[0,0]$ with the widening operator: $[0,0]\nabla[1,1] = [0,+\infty]$. Thus, `i` has $[1,+\infty]$ and $[1,7]$ at lines 4 (and 5) and 6, respectively. At the third iteration, we combine the values $[0,0]$ and $[1,7]$ with the widening: $[0,0]\nabla[1,7] = [0,+\infty]$, reaching a fixed point. Thus, the analysis concludes that `i` has interval $[1,+\infty]$ at line 5 and fails to prove the buffer-overrun safety.

(Note: In this simplified example program, the precision loss can be recovered by applying narrowing after widening. However, in reality, things become much complicated: for instance, each of the statements in the aforementioned program may appear in distant program points (across procedures), and the loop can be made by calling a procedure multiple times. In such a case, narrowing hardly recovers the precision loss caused by widening, and using thresholds in the widening phase is essential for analysis precision [2, 3].)

Widening with thresholds. On the other hand, if we analyze the program with the widening-with-thresholds technique [2, 3], we can prove the safety. This method improves the analysis precision by bounding the extrapolation performed by the widening operator. In this approach, we are given a set $T \subseteq \mathbb{N}$ of thresholds, and these thresholds are successively used as a candidate of a fixed point. The benefits of the technique crucially depend on the choice of T, but choosing a good T is orthogonal to our technique, and in this overview, we assume that $T = \{1, 2, \ldots, 12\}$ is given. With this threshold set, the analysis proceeds as follows:

Lines	Iterations								
	1	2	3	4	5	6	7	8	9
3	$[0,0]$	$[0,1]$	$[0,2]$	$[0,3]$	$[0,4]$	$[0,5]$	$[0,6]$	$[0,7]$	$[0,7]$
4, 5	$[1,1]$	$[1,2]$	$[1,3]$	$[1,4]$	$[1,5]$	$[1,6]$	$[1,7]$	$[1,8]$	$[1,8]$
6	$[1,1]$	$[1,2]$	$[1,3]$	$[1,4]$	$[1,5]$	$[1,6]$	$[1,7]$	$[1,7]$	$[1,7]$

At the second iteration, the widening $[0,0]\nabla[1,1]$ is bounded by the best possible upper bound in the threshold set (i.e., 1), giving the result $[0,1]$ at line 3. At the third iteration, we apply widening $[0,0]\nabla[1,2]$ with threshold 2, obtaining $[0,2]$. In this way, the analysis progressively uses threshold values $1, 2, 3, \ldots, 7$ until the analysis reaches a fixed point. At line 3, interval $[0,7]$ is a fixed point, and the analysis terminates at the ninth iteration. With this result, we can prove the safety at line 5, because the value of `i` is $[1,8]$, which is less than the size of array `a`.

Problem. However, this technique requires significantly longer iterations to converge than the original widening approach does. The main problem is that the technique searches for the effective threshold value in a linear fashion. For instance, in our example program, the bound 7, which

[‡]The widening operator for intervals is defined as follows:

$$\begin{array}{rcll} [a,b] & \nabla & \bot & = [a,b] \\ \bot & \nabla & [c,d] & = [c,d] \\ [a,b] & \nabla & [c,d] & = [(c < a ? -\infty : a), (b < d ? +\infty : b)] \end{array}$$

actually improves the precision, is found only after considering all of the smaller values $1, 2, \ldots, 6$ in T. The ideal solution to this problem is to use a smaller threshold set. For instance, if we use $T = \{7\}$ for our example program, the analysis reaches the fixed point ($[0, 7]$ at line 3) with just one extra iteration. However, choosing such a minimal yet effective threshold set in practice is very challenging, and sometimes it is inevitable to use a large threshold set. For instance, we found that a common heuristic that collects all constant integers involved in conditional expressions in the program [2] leads to fairly large threshold sets in our case (Section 4).

Our approach. Our aim is to improve the efficiency of the widening-with-thresholds technique itself, no matter what threshold sets are used. Thus, with our technique, widening with thresholds can be efficiently applied even when the threshold set is inevitably large. The key idea is to replace the linear search involved in the existing technique by a binary search, in order to reduce ineffective trials of threshold values. Consider the example program with the same threshold set $T = \{1, 2, \ldots, 12\}$. With our technique, the analysis proceeds as follows:

Lines	Iterations				
	1	2	3	4	5
3	$[0, 0]$	$[0, 6]$	$[0, 9]$	$[0, 7]$	$[0, 7]$
4, 5	$[1, 1]$	$[1, 7]$	$[1, 10]$	$[1, 8]$	$[1, 8]$
6	$[1, 1]$	$[1, 7]$	$[1, 7]$	$[1, 7]$	$[1, 7]$

At the second iteration, we apply widening $[0, 0]\nabla[1, 1]$ with threshold 6, not with the best possible upper bound 2. That is, we use the threshold value at the middle position of T, that is, $6 = (1 + 12)/2$. At the third iteration, we first check if the analysis has reached a fixed point. Because the result is not yet a fixed point, that is, $[0, 7] \not\sqsubseteq [0, 6]$, we apply widening $[0, 0]\nabla[1, 7]$, now with the threshold 9, which is determined by applying the binary search with the range from 6 to 12, that is, $9 = (6 + 12)/2$. At the fourth iteration, we find that the result is a fixed point, that is, $[0, 8] \sqsubseteq [0, 9]$. In this case, we roll back the analysis result to that of the second iteration ($[0, 6]$) and restart the binary search with a reduced range (6 to 9); we use 7 as the threshold, that is, $7 = (6 + 9)/2$. At the fifth iteration, the analysis terminates because it has reached a fixed point and there is no more range to perform the binary search. (We tried both thresholds 6 and 7, where the analysis result is a fixed point with 7 but not with 6.) Note that, for this example program, the analysis has the same precision as the ordinary technique but requires fewer iterations to converge.

Tradeoffs between precision and cost in practice. In some cases, our binary-search-based approach produces less precise results than the conventional linear-search-based approach. The following program shows a typical case where our technique misses opportunities for precision improvement:

```
int a[10];
int i = 0;
while (1) {
  i++;
  a[i] = 0; // safe
  if (i == 8) break;
}
```

The difference from the previous example is that we use `==` at line 6 rather than `>=`. The conventional technique with $T = \{1, 2, \ldots, 12\}$ produces the same result as before:

Lines	Iterations								
	1	2	3	4	5	6	7	8	9
3	$[0,0]$	$[0,1]$	$[0,2]$	$[0,3]$	$[0,4]$	$[0,5]$	$[0,6]$	$[0,7]$	$[0,7]$
4, 5	$[1,1]$	$[1,2]$	$[1,3]$	$[1,4]$	$[1,5]$	$[1,6]$	$[1,7]$	$[1,8]$	$[1,8]$
6	$[1,1]$	$[1,2]$	$[1,3]$	$[1,4]$	$[1,5]$	$[1,6]$	$[1,7]$	$[1,7]$	$[1,7]$

but our technique gives this result:

Lines	Iterations						
	1	2	3	4	5	6	7
3	$[0,0]$	$[0,6]$	$[0,9]$	$[0,11]$	$[0,12]$	$[0,+\infty]$	$[0,+\infty]$
4, 5	$[1,1]$	$[1,7]$	$[1,10]$	$[1,12]$	$[1,13]$	$[1,+\infty]$	$[1,+\infty]$
6	$[1,1]$	$[1,7]$	$[1,10]$	$[1,12]$	$[1,13]$	$[1,+\infty]$	$[1,+\infty]$

That is, because the condition `i == 8` has no effect on interval $[1,10]$ at the third iteration, our technique misses to prescribe 7 as a threshold. On the other hand, the conventional method tries all the values in T and does not miss the chance.

Experimental results. In practice, our technique is shown to have a good precision/cost balance compared with the conventional method. We have implemented the existing and our widening-with-threshold techniques in a realistic C static analyzer, Sparrow [4], and compared their performance in terms of analysis precision and time. The results show that, for five real C programs, our technique achieved on average 79.1% of the conventional technique's precision (i.e., 20.9% loss in precision) with only 18.5% of cost overhead (i.e., 81.5% improvement in efficiency).

Contributions. To summarize, we make the following contributions:

- We present a new technique for performing widening with thresholds via binary search.
- We formalize the technique in a general setting and prove its correctness and termination.
- We show the effectiveness of the technique by experiments with a realistic static analyzer.

2. PRELIMINARIES

In this section, we review static analysis with widening and the standard technique of widening with thresholds.

2.1. *Static analysis with widening*

We consider a static analysis designed by abstract interpretation [5, 6]. In abstract interpretation, a static analysis is specified with an abstract domain $\mathbb{D}$ and abstract semantic function:

$$F : \mathbb{D} \to \mathbb{D}$$

where $\mathbb{D}$ is a complete partial order and F is a monotone function, that is, $d_1 \sqsubseteq d_2 \implies F(d_1) \sqsubseteq F(d_2)$. Then, the job of the analysis is to compute, in finite steps, an upper bound $\mathcal{A} \in \mathbb{D}$ of

$$\bigsqcup_{i \geqslant 0} F^i(\bot) = F^0(\bot) \sqsubseteq F^1(\bot) \sqsubseteq F^2(\bot) \sqsubseteq \dots \tag{1}$$

However, when the height of abstract domain $\mathbb{D}$ is infinite or large, the fixed point computation (1) may not terminate or takes too much time to complete. In this case, we can compute an upper bound, that is, $\mathcal{A} \sqsupseteq \bigsqcup_{i \geqslant 0} F^i(\bot)$, with a widening operator to guarantee or accelerate the termination [5]. A widening operator

$$\nabla : \mathbb{D} \to \mathbb{D} \to \mathbb{D}$$

is a binary operator that has the following two properties:

- It is an upper bound operator, that is,

$$\forall a, b \in \mathbb{D} : (a \sqsubseteq a \nabla b) \wedge (b \sqsubseteq a \nabla b). \tag{2}$$

- For all increasing chains $x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq \ldots$ in $\mathbb{D}$, the chain $(y_i)_i$ defined as

$$\begin{aligned} y_0 &= x_0 \\ y_{i+1} &= y_i \nabla x_{i+1} \end{aligned} \tag{3}$$

  is finite (eventually stabilizes after finite steps).

With a widening operator $\nabla : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$, we compute an increasing chain $(X_i)_i$ as follows:

$$\begin{aligned} X_0 &= \bot \\ X_{i+1} &= X_i & F(X_i) \sqsubseteq X_i \\ &= X_i \nabla F(X_i) & \text{otherwise.} \end{aligned} \tag{4}$$

Then the abstract interpretation framework ensures that the chain $(X_i)_i$ is finite and its limit (X such that $F(X) \sqsubseteq X$) is an upper bound of $\bigsqcup_{i \geqslant 0} F^i(\bot)$.

Example 1
Throughout the paper, we use a simple running example. Consider an abstract domain $\mathbb{D} = \mathbb{N} \cup \{\infty\}$, where $\mathbb{N} = \{\overline{1}, \overline{2}, \overline{3}, \ldots\}$ is the set of natural numbers (we put the bar on top of the domain elements to distinguish them from numbers used in subscripts). Note that the domain $\mathbb{D}$ is a complete partial order, where the order between elements is defined as follows:

$$i \sqsubseteq j \text{ iff } \begin{cases} i \leqslant j & i, j \in \mathbb{N} \\ j = \infty & \text{otherwise.} \end{cases}$$

Consider the following semantic function $F \in \mathbb{D} \to \mathbb{D}$:

$$F(n) = \text{if } n < \overline{9} \text{ then } n + \overline{1} \text{ else } n.$$

Intuitively, F models a conditional statement `if (n < 9) n++;`. Note that, in this case, computing the sequence in (1) involves nine iterations:

$$\begin{aligned} F(\overline{1}) &= \overline{2} \\ F(\overline{2}) &= \overline{3} \\ &\vdots \\ F(\overline{8}) &= \overline{9} \\ F(\overline{9}) &= \overline{9} \end{aligned}$$

A widening operator is used to accelerate the iteration. Suppose that we use a widening operator $\nabla : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$, defined as follows:

$$x \nabla y = \begin{cases} x & \text{if } x \sqsupseteq y \\ \infty & \text{otherwise.} \end{cases} \tag{5}$$

Then, the chain (4) proceeds as follows:

$$\begin{aligned} X_0 &= \overline{1} \\ X_1 &= \overline{1} \nabla F(\overline{1}) = \overline{1} \nabla \overline{2} = \infty \\ X_2 &= X_1 = \infty \ \ (\text{since } F(X_1) \sqsubseteq X_1). \end{aligned}$$

Thus, with the widening operator, the fixed-point computation terminates in three steps, but the result, $\lim_{i \in \mathbb{N}} X_i = \infty$, is less precise than the least fixed point, $\overline{9}$. The goal of widening with threshold is to prevent this precision loss. ■

2.2. Widening with thresholds

The idea of widening with thresholds is to bound the extrapolation of the standard widening iteration.

Suppose that a set $T \subseteq \mathbb{D}$ of thresholds is given, where the elements in T are totally ordered. When T has n elements, we write $t_k (1 \leqslant k \leqslant n)$ for the kth smallest element in T. That is,

$$T = \{t_1, t_2, \ldots, t_n\}$$

where $t_i \in \mathbb{D}$ and $t_1 \sqsubseteq t_2 \sqsubseteq \cdots \sqsubseteq t_n$. Without loss of generality, we assume that $n \geqslant 2, t_1$ is the bottom element in $\mathbb{D}$, that is, $t_1 = \bot$, and t_n is the top element, that is, $t_n = \top$.

We define two auxiliary functions regarding the thresholds set:

$$lub(X) = \min\{k \mid 1 \leqslant k \leqslant |T| \wedge X \sqsubseteq t_k\}$$

$$glb(X) = \max\{k \mid 1 \leqslant k \leqslant |T| \wedge t_k \sqsubseteq X\}.$$

For a given $X \in \mathbb{D}, lub(X)$ denotes the smallest index k of threshold values greater than or equal to X. That is, $T_{lub(X)}$ is the best possible (i.e., least) upper bound of X in the threshold set. On the other hand, $T_{glb(X)}$ means the largest threshold that is less than or equal to X, that is, $T_{glb(X)}$ is the best possible (i.e., greatest) lower bound of X in the threshold set.

Example 2

In our running example (Example 1), suppose that we use the threshold set $T = \{\overline{1}, \overline{2}, \ldots, \overline{9}, \infty\}$. In this case, $t_1 = \overline{1}, t_2 = \overline{2}, \ldots, t_9 = \overline{9}$, and $t_{10} = \infty$. Then, for instance,

$$lub(\overline{2}) = \min\{k \mid 1 \leqslant k \leqslant |T| \wedge \overline{2} \sqsubseteq t_k\} = \min\{2, 3, \ldots, 9, 10\} = 2$$

$$glb(\overline{6}) = \max\{k \mid 1 \leqslant k \leqslant |T| \wedge t_k \sqsubseteq \overline{6}\} = \min\{1, 2, \ldots, 6\} = 6.$$

■

The widening operator $\nabla^T : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$ with threshold set T is defined as follows:

$$X \nabla^T Y = (X \nabla Y) \sqcap t_{lub(X \sqcup Y)}. \tag{6}$$

That is, given X and Y, we first apply the ordinary widening operator, that is, $X \nabla Y$, and then prune the result with the best possible threshold that over-approximates both X and Y, that is, $lub(X \sqcup Y)$. This widening operator is used in the widening sequence (4) in the usual way.

Example 3

In the running example with $T = \{\overline{1}, \overline{2}, \ldots, \overline{9}, \infty\}$, the increasing chain (4) with the widening operator ∇^T proceeds as follows:

$$\begin{aligned}
X_0 &= \overline{1} \\
X_1 &= \overline{1} \nabla^T F(\overline{1}) = \overline{1} \nabla^T \overline{2} = (\overline{1} \nabla \overline{2}) \sqcap t_{lub(\overline{2})} = (\overline{1} \nabla \overline{2}) \sqcap t_2 = \infty \sqcap \overline{2} = \overline{2} \\
&\vdots \\
X_8 &= \overline{8} \nabla^T F(\overline{8}) = \overline{8} \nabla^T \overline{9} = (\overline{8} \nabla \overline{9}) \sqcap t_{lub(\overline{9})} = (\overline{8} \nabla \overline{9}) \sqcap t_9 = \infty \sqcap \overline{9} = \overline{9} \\
X_9 &= X_8 = \overline{9}.
\end{aligned}$$

Thus, $\lim_{i \in \mathbb{N}} X_i = \overline{9}$. Note that the fixed point iteration with the threshold set prevents the precision loss of the simple widening operator in (5). However, the sequence of widening with thresholds takes longer iterations to converge than the simple widening method. Our goal is to accelerate the widening iteration with thresholds without compromising the analysis precision too much. ■

3. OUR TECHNIQUE

In this section, we formalize our technique.

3.1. Widening with thresholds via binary search

Our technique differs from the standard widening-with-thresholds technique in two ways.

Widening operator. First, we use a different widening operator. Our widening operator $\nabla_s^T : \mathbb{D} \times \mathbb{D} \rightarrow \mathbb{D}$ is not only parameterized with thresholds set T but also with an index s ($1 \leqslant s \leqslant |T|$) and is defined as follows:

$$X \nabla_s^T Y = (X \nabla Y) \sqcap t_{\left\lfloor \frac{lub(X \sqcup Y)+s}{2} \right\rfloor}. \tag{7}$$

Note that, given X and Y, our widening operator uses the threshold $t_{\left\lfloor \frac{lub(X \sqcup Y)+s}{2} \right\rfloor}$, that is, the threshold value at the middle position of $lub(X \sqcup Y)$ and s, whereas the standard method uses the best possible one, that is, $t_{lub(X \sqcup Y)}$.

Example 4
Consider the running example with $T = \{\overline{1}, \overline{2}, \ldots, \overline{9}, \infty\}$. Then, for example,

$$\overline{1} \nabla_{10}^T \overline{2} = (\overline{1} \nabla \overline{2}) \sqcap t_{\left\lfloor \frac{lub(\overline{1} \sqcup \overline{2})+10}{2} \right\rfloor} = (\overline{1} \nabla \overline{2}) \sqcap t_{\left\lfloor \frac{2+10}{2} \right\rfloor} = \infty \sqcap t_6 = \infty \sqcap \overline{6} = \overline{6},$$

while the standard widening with thresholds results in $\overline{1} \nabla^T \overline{2} = \overline{2}$. ■

We assume that the widening operation $X \nabla_s^T Y$ is always performed under the condition $s \geqslant lub(X \sqcup Y)$. With this condition, it is easy to show that ∇_s^T is a valid widening operator that satisfies conditions (2) and (3).

Fixed-point computation. The other difference is that we use our own fixed-point computation strategy. Instead of computing the chain (4), we compute sequence $Y_0, Y_1, Y_2, \ldots$ (until $Y_{i+1} = Y_i$) defined as follows. Each $Y_i = (X_i, l_i, r_i)$ is a triple of an abstract state $X_i \in \mathbb{D}$ and indices l_i and r_i that denote the current status of binary search. The initial configuration Y_0 has the bottom state with the first and last indices of the thresholds set T:

$$Y_0 = (\bot, 1, |T|).$$

Throughout the computation, our method assumes that l_i is the largest unstable threshold index and r_i is the smallest stable threshold index among the ones considered so far. For instance, t_0 satisfies this invariant because F is unstable at $t_1 (= \bot)$, that is, $F(\bot) \not\sqsubseteq \bot$, and F is stable at $t_{|T|} = \top$, that is, $F(\top) \sqsubseteq \top$.

Given the ith configuration $Y_i = (X_i, l_i, r_i)$, Y_{i+1} is defined as follows:

$$Y_{i+1} = \begin{cases} (t_{r_i}, r_i - 1, r_i) & \text{if } F(X_i) \not\sqsubseteq X_i \wedge lub(X_i \sqcup F(X_i)) \geqslant r_i \\ (X_i \nabla_{r_i}^T F(X_i), glb(X_i), r_i) & \text{if } F(X_i) \not\sqsubseteq X_i \wedge lub(X_i \sqcup F(X_i)) < r_i \\ (t_{l_i}, l_i, lub(X_i)) & \text{if } F(X_i) \sqsubseteq X_i \wedge l_i + 1 < r_i \\ (X_i, l_i, r_i) & \text{if } F(X_i) \sqsubseteq X_i \wedge l_i + 1 = r_i. \end{cases} \tag{8}$$

Suppose that X_i is not yet a fixed point of F (i.e., $F(X_i) \not\sqsubseteq X_i$). We consider two cases: (1) when the binary search is finished (i.e., $lub(X_i \sqcup F(X_i)) \geqslant r_i$, here if F is monotone $lub(X_i \sqcup F(X_i))$ should not exceed r_i), we terminate the sequence and return the smallest stable result t_{r_i} searched so far and set $l_{i+1} = r_i - 1$ and $r_{i+1} = r_i$ (which indicates the completion of the search) and (2) otherwise, we apply the widening operator with index r_i:

$$X_i \nabla_{r_i}^T F(X_i) = (X_i \nabla F(X_i)) \sqcap t_{\left\lfloor \frac{lub(X_i \sqcup F(X_i)) + r_i}{2} \right\rfloor}$$

and set l_{i+1} to $glb(X_i)$ and r_{i+1} to r_i because now the largest unstable threshold index is $glb(X_i)$ and the smallest stable index is unchanged.

Next, suppose that X_i is a fixed point of F (i.e., $F(X_i) \sqsubseteq X_i$). We consider two cases. The first case is when we have not yet finished the search (i.e., $l_i + 1 < r_i$). In this case, we roll back the most recently unstable state and continue to search for more precise fixed points. Because we do not keep track of the previously unstable point, we instead go back to t_{l_i}, the largest unstable threshold. And, because X_i is a fixed point, we set r_{i+1} to $lub(X_i)$, which is also a fixed point of F. The second case is when we have finished the search (i.e., $l_i + 1 = r_i$). In this case, Y_{i+1} is defined as Y_i and we terminate the computation. In summary, we compute the sequence

$$(X_0, l_0, r_0),\ (X_1, l_1, r_1),\ (X_2, l_2, r_2),\ \ldots,\ (X_i, l_i, r_i),\ \ldots$$

until $F(X_i) \sqsubseteq X_i$ and $l_i + 1 = r_i$.

Example 5
Consider Example 1, and let $T = \{\overline{1}, \overline{2}, \overline{3}, \ldots, \overline{9}, \infty\}$. Our fixed-point computation strategy (8) proceeds as follows:

$$\begin{aligned}
Y_0 &= (\overline{1}, 1, 10) \\
Y_1 &= (\overline{1} \nabla_{10}^T \overline{2}, glb(\overline{1}), 10) = \left(\infty \sqcap t_{\left\lfloor \frac{lub(\overline{2})+10}{2} \right\rfloor}, 1, 10\right) = (\overline{6}, 1, 10) \\
Y_2 &= (\overline{6} \nabla_{10}^T \overline{7}, glb(\overline{6}), 10) = \left(\infty \sqcap t_{\left\lfloor \frac{lub(\overline{7})+10}{2} \right\rfloor}, 6, 10\right) = (\overline{8}, 6, 10) \\
Y_3 &= (\overline{8} \nabla_{10}^T \overline{9}, glb(\overline{8}), 10) = \left(\infty \sqcap t_{\left\lfloor \frac{lub(\overline{9})+10}{2} \right\rfloor}, 8, 10\right) = (\overline{9}, 8, 10) \\
Y_4 &= (t_8, 8, lub(\overline{9})) = (\overline{8}, 8, 9) \\
Y_5 &= \left(\overline{8} \nabla_{10}^T \overline{9}, glb(\overline{8}), 9\right) = \left(\infty \sqcap t_{\left\lfloor \frac{lub(\overline{9})+10}{2} \right\rfloor}, 8, 9\right) = (\overline{9}, 8, 9) \\
Y_6 &= Y_5 = (\overline{9}, 8, 9).
\end{aligned}$$

Note that this result is the same as the result with standard widening with threshold (Example 3) but needs fewer iterations to terminate. ■

Correctness and termination. It is easy to show that the result of our method is safe: if the sequence (8) terminates, the final result X is guaranteed to be an upper bound of $\bigsqcup_{i \geqslant \mathbb{N}} F^i(\bot)$, because the result X is a fixed point of F (i.e., $F(X) \sqsubseteq X$). However, termination of (8) is not obvious, which we prove in the following lemma.

Theorem 1 (Termination)
Let $\mathbb{D}$ be an abstract domain and $F : \mathbb{D} \to \mathbb{D}$ be a monotone function on $\mathbb{D}$. Let $T = \{t_1, t_2, \ldots, t_n\}$ be the set of thresholds and $Y_0 = (\bot, 1, |T|)$. Define $Y_i = (X_i, l_i, r_i)$ according to (8). Then, there always exists N such that $(X_{N+1}, l_{N+1}, r_{N+1}) = (X_N, l_N, r_N)$.

Proof
Consider $Y_i = (X_i, l_i, r_i)$. We consider the four cases in (8). For the first and last cases, it is easy to show that the sequence immediately terminates. For the second and third cases, we show that the search bound (l_i, r_i) will be eventually narrowed down, subsequently leading to the immediate (the first and last) cases.

- When $F(X_i) \sqsubseteq X_i$ and $l_i + 1 < r_i$: In this case,

  $$Y_{i+1} = (t_{l_i}, l_i, lub(X_i)).$$

  Consider the four possible cases for Y_i:

  1. $Y_i = (X_{i-1} \nabla^T_{r_{i-1}} F(X_{i-1}), glb(X_{i-1}), r_{i-1})$ with $F(X_{i-1}) \not\sqsubseteq X_{i-1}$ and $lub(X_{i-1} \sqcup F(X_{i-1})) < r_{i-1}$. In this case, we show that $r_i = r_{i-1} > r_{i+1}$:

  $$\begin{aligned} r_{i+1} &= lub(X_i) && \text{def. of } r_{i+1} \\ &= lub(X_{i-1} \nabla^T_{r_{i-1}} F(X_{i-1})) && \text{def. of } X_i \\ &= lub\left(X_{i-1} \nabla F(X_{i-1}) \sqcap t_{\left\lfloor \frac{lub(X_{i-1} \sqcup F(X_{i-1})) + r_{i-1}}{2} \right\rfloor} \right) && \text{def. of } \nabla^T_{r_{i-1}} \\ &\leqslant lub\left(t_{\left\lfloor \frac{lub(X_{i-1} \sqcup F(X_{i-1})) + r_{i-1}}{2} \right\rfloor} \right) && \\ &< lub(t_{r_{i-1}}) && \text{condition on } Y_i \\ &= r_{i-1} = r_i && \text{def. of } lub \end{aligned}$$

  2. $Y_i = (t_{l_{i-1}}, l_{i-1}, lub(X_{i-1}))$ with $F(X_{i-1}) \sqsubseteq X_{i-1}$ and $l_{i-1} + 1 < r_{i-1}$. We show that $r_{i+1} < r_{i-1}$:

  $$\begin{aligned} r_{i+1} &= lub(X_i) && \text{def. of } r_{i+1} \\ &= lub(t_{l_{i-1}}) && \text{def. of } X_i \\ &= l_{i-1} && \text{def. of } lub \\ &< r_{i-1} && \text{condition on } Y_i \end{aligned}$$

  3. When $Y_i = (t_{r_{i-1}}, r_{i-1} - 1, r_{i-1})$ and $F(X_{i-1}) \not\sqsubseteq X_{i-1} \wedge lub(X_{i-1} \sqcup F(X_{i-1})) \geqslant r_{i-1}$: This case cannot occur because we assume that $l_i + 1 < r_i$.
  4. When $Y_i = (X_{i-1}, l_{i-1}, r_{i-1})$ and $F(X_{i-1}) \sqsubseteq X_{i-1} \wedge l_{i-1} + 1 = r_{i-1}$: This case cannot occur we assume that $l_i + 1 < r_i$.

- When $F(X_i) \not\sqsubseteq X_i$ and $lub(X_i \sqcup F(X_i)) < r_i$: In this case,

  $$Y_{i+1} = (X_i \nabla^T_{r_i} F(X_i), glb(X_i), r_i).$$

  We consider the four possible cases for Y_{i+1}, as we did in the previous case:

  1. For instance, consider the case that $F(X_{i+1}) \not\sqsubseteq X_{i+1}$ and $lub(X_{i+1} \sqcup F(X_{i+1})) < r_{i+1}$: In this case, we show that $l_{i+1} < l_{i+2}$:

  $$\begin{aligned} l_{i+2} &= glb(X_{i+1}) && \text{def. of } l_{i+2} \\ &= glb\left(X_i \nabla^T_{r_i} F(X_i)\right) && \text{def. of } X_{i+1} \\ &= \left\lfloor \frac{lub(X_i \sqcup F(X_i)) + r_i}{2} \right\rfloor && \text{def. of } X_i \nabla^T_{r_i} F(X_i) \\ &> glb(X_i) && F(X_i) \not\sqsubseteq X_i,\ \text{def. of } lub, glb \\ &= l_{i+1} && \text{def. of } l_{i+1} \end{aligned}$$

  2. Other three cases are proved similarly.

□

Discussion. Our widening-with-thresholds technique via binary search can be formulated as a specialization of an abstract domain and widening operator. Given an abstract domain $\mathbb{D}$, suppose we define a new domain $\mathbb{D}' = \mathbb{D} \times \mathbb{N} \times \mathbb{N}$, where domain elements $(d, l, r) \in \mathbb{D}'$ is ordered as follows:

$$(d_1, l_1, r_1) \sqsubseteq (d_2, l_2, r_2) \text{ iff } (l_1 \leqslant l_2 \wedge r_1 \geqslant r_2) \vee (l_1 = l_2 \wedge r_1 = r_2 \wedge d_1 \sqsubseteq d_2).$$

Table I. Experimental results.

		Without		Standard			Ours		Comparison	
Programs	LOC (K)	alms	sec	$\|T\|$	alms	sec	alms	sec	precision (%)	cost (%)
archimedes	7	1273	15	113	1105	18	1113	16	95.2	33.3
bc-10.6	13	555	144	120	547	277	551	154	50.0	7.5
tar-1.13	20	954	78	131	941	237	944	105	76.9	17.0
make-3.76	27	1847	224	122	1819	656	1825	309	78.6	19.7
a2ps-6.14	64	2029	2073	241	1768	2489	1847	2161	69.7	21.2
TOTAL	131	6658	2534		6180	3677	6280	2745	**79.1**	**18.5**

Without, widening without thresholds; Standard, the standard widening-with-thresholds technique (with linear search); Ours, widening with threshold based on binary search. The 'Comparison' show the comparison against the standard technique of widening with thresholds. precision, the relative precision of our technique compared with the standard method; cost, the relative cost (analysis time) of our technique compared with the standard method; $|T|$, the size of the threshold set used in experiments; alms, the number of alarms reported by each analyzer.

The definition of the join operator for $\mathbb{D}'$ immediately follows from the ordering definition. Now, the fixed-point computation in (8) defines a particular widening operator $\nabla' : \mathbb{D}' \times \mathbb{D}' \to \mathbb{D}'$ for the new domain, which performs a binary search. In this formulation, Theorem 1 corresponds to proving that ∇' satisfies the conditions of widening (i.e., termination of widening).

4. EXPERIMENTS

We have implemented our technique in Sparrow [4], a buffer-overrun analysis tool for C programs. Sparrow basically performs a flow-sensitive and context-insensitive analysis using the interval abstract domain. From the baseline analyzer, we have made two analyzers that apply widening with thresholds based on the standard linear search and our binary search, respectively. For those analyses, we used the same threshold set T that contains all the constant integers that appear in conditional statements of the given program. Note that this simple method generates a quite large threshold set in practice (column $|T|$). To implement our technique, we applied the technique in Section 3 separately for variables and program points. That is, we maintain the status information (left/right indices) of binary search for each variable and program point in the program.

Table I shows that our technique has a better cost/precision balance than the standard widening-with-threshold technique. Also, the results show that, with our technique, the widening with threshold can be effectively used even with a large set of thresholds. We have tested five GNU open-source programs. We compared the precision and cost of our technique based on the standard widening-with-threshold technique. In total, the standard method have reduced 478 alarms over the five programs while our technique 378 alarms, achieving 79.1% (378/478) of precision. In doing so, the standard method increased the analysis time by 1142 s and ours 210 s: our technique only has 18.5% (210/1142) overhead compared with the standard technique.

5. RELATED WORK

Our technique can be orthogonally used with existing threshold-inference techniques. Most existing work on widening with thresholds have focused on the problem of determining threshold values [2, 3, 7–11]. For instance, in [2, 7], the set of thresholds is inferred from the program text; integer constants such as those used in conditional expressions are used as thresholds. In our experiments, we also used this method to determine the threshold set and showed that our technique improves its effectiveness. In [3], a more sophisticated method is proposed, where relevant thresholds are inferred by a semantics-based pre-analysis. In [3], the number of thresholds used is not given, but the experiments show that the analysis with the inferred threshold sometimes leads to significant cost blow-up. Our technique can be used with these techniques to improve the efficiency.

Comparing the widening-with-thresholds technique with other advanced extrapolation techniques is beyond the scope of this paper. In the literature, many techniques have been proposed to refine the naive application of widening operators [12–14]. Gopan and Reps [13] proposed the guided static analysis framework that tames the widening by restricting the target program. Given a suitable program transformer (restriction strategy), they iteratively apply static analysis to the sequence of restricted program. They provided two instances: (1) widening in loops with multiple phases, which is a generalization of the lookahead widening [14] and (2) widening in loops with non-deterministically chosen behavior. Widening with landmarks [12] dynamically generates and selects the limit of widening (landmark), which can be considered as a refinement of widening with thresholds. Whenever a polyhedron (current state) is intersected with another inequality that is unsatisfiable in the current iterate, the inequality is selected as a landmark. These techniques provide means for improving widening, which is basically orthogonal to the widening-with-threshold technique. However, it would be an interesting direction to compare the effectiveness of these different techniques in practice.

6. CONCLUSIONS

In this paper, we have presented a general and practical technique for efficiently performing widening with thresholds. Our technique employs a binary search in the process of applying the thresholds, thereby reducing the number of ineffective trials. We formalize the technique in the abstract interpretation framework, prove its correctness, and experimentally show that our technique has a better cost/accuracy balance than the conventional one.

Our technique provides a new point in the design space of static analyzers that use widening. The ideal solution to the performance problem of widening with thresholds would be to use a small yet precision-effective set of thresholds. However, automatically finding such a set for a given program is non-trivial. With our technique, the method of widening with thresholds can be effectively used even when a conservative set of thresholds is used.

ACKNOWLEDGEMENTS

This work was supported by the Engineering Research Center of Excellence Program of Korea Ministry of Science, ICT and Future Planning(MSIP)/National Research Foundation of Korea(NRF) (grant NRF-2008-0062609) and by Samsung Electronics Software Center. This work was partly supported by the Institute for Information and Communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (no. B0101-15-0557, Resilient Cyber-Physical Systems Research) and (no. R0190-15-2011, Development of Vulnerability Discovery Technologies for IoT Software Security).

REFERENCES

1. Cousot P, Cousot R. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, Dunod, Paris, France, 1976; 106–130.
2. Halbwachs N, Proy YE, Roumanoff P. Verification of real-time systems using linear relation analysis. In *Formal Methods in System Design*, 1997; 157–185.
3. Lakhdar-Chaouch L, Jeannet B, Girault A. Widening with thresholds for programs with complex control graphs. In *Proceedings of the 9th International Conference on Automated Technology for Verification and Analysis*, ATVA'11. Springer-Verlag: Berlin, Heidelberg, 2011; 492–502.
4. Oh H, Heo K, Lee W, Lee W, Yi K. Sparrow. http://ropas.snu.ac.kr/sparrow [last accessed November 2015].
5. Cousot P, Cousot R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Los Angeles, California, USA, 1977; 238–252.
6. Cousot P, Cousot R. Systematic design of program analysis frameworks. In *Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press: New York, NY, San Antonio, Texas, 1979; 269–282.
7. Blanchet B, Cousot P, Cousot R, Feret J, Mauborgne L, Miné A, Monniaux D, Rival X. The essence of computation. In *Design and Implementation of a Special-purpose Static Program Analyzer for Safety-critical Real-time Embedded Software*. Springer-Verlag New York, Inc.: New York, NY, USA, 2002; 85–108.
8. Bouissou O, Seladji Y, Chapoutot A. Acceleration of the abstract fixpoint computation in numerical program analysis. *Journal of Symbolic Computation* 2012; **47**(12):1479–1511.

9. Mihaila B, Sepp A, Simon A. Widening as abstract domain. In *NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings*, Moffett Field, CA, USA, 2013; 170–184.
10. Cousot P, Cousot R, Feret J, Mauborgne L, Miné A, Rival X. Why does Astrée scale up? *Formal Methods in System Design* 2009; **35**(3):229–264.
11. Cousot P, Cousot R, Feret J, Mauborgne L, Miné A, Monniaux D, Rival X. Combination of abstractions in the ASTREE static analyzer. In *Proceedings of the 11th Asian Computing Science Conference on Advances in Computer Science: Secure Software and Related Issues*, ASIAN'06. Springer-Verlag: Berlin, Heidelberg, 2007; 272–300.
12. Simon A, King A. Widening polyhedra with landmarks. In *Asian Symposium on Programming Languages and Systems*, vol. 4279, LNCS. Springer: Sydney, Australia, 2006; 166–182.
13. Gopan D, Reps TW. Guided static analysis. In *SAS 2007*, LNCS 4634. Springer Verlag: Berlin, Heidelberg, 2007; 349–365.
14. Gopan D, Reps T. Lookahead widening. In *Proceedings of the 18th International Conference on Computer Aided Verification*, CAV'06. Springer-Verlag: Berlin, Heidelberg, 2006; 452–466.