Precise Range Analysis on Large Industry Code

Shrawan Kumar, Bharti Chimdyalwar, Ulka Shrotri
Tata Consultancy Services, India
shrawan.kumar@tcs.com, bharti.c@tcs.com, ulka.s@tcs.com

ABSTRACT
Abstract interpretation is widely used to perform static code analysis with non-relational (interval) as well as relational (difference-bound matrices, polyhedral) domains. Analysis using non-relational domains is highly scalable but delivers imprecise results, whereas, use of relational domains produces precise results but does not scale up. We have developed a tool that implements K-limited path sensitive interval domain analysis to get precise results without losing on scalability. The tool was able to successfully analyse 10 million lines of embedded code for different properties such as division by zero, array index out of bound (AIOB), overflow-underflow and so on. This paper presents details of the tool and results of our experiments for detecting AIOB property. A comparison with the existing tools in the market demonstrates that our tool is more precise and scales better.

Categories and Subject Descriptors
F.3.2 [**Semantics of Programming Languages**]: Program analysis; D.2.4 [**Software/ Program Verification**]: Validation.

General Terms
Experimentation, Verification, Scalability, Precision, Soundness

Keywords
Abstract Interpretation, Interval Domain, Range Analysis

1. INTRODUCTION
Abstract interpretation [12] is a method to compute finitely smaller abstract state space by approximating large (possibly infinite) concrete states of programs using an abstract domain. These approximations are sound in the sense that when a property holds with approximations, it is guaranteed to hold in concrete program also but not necessarily the other way round. Static analysis based on abstract interpretation is widely used to detect errors such as array index out of bound (AIOB), arithmetic overflow, zero division and so on [9, 10, 13]. Such instances of analysis using complex relational abstract domain produce precise results but do not scale-up [3]. Analysis using non-relational domain scales to Millions of Lines of Code (MLOC) but at the cost of precision.
In simple terms, non-relational domain maintains merged range of a variable along all paths in a program, whereas relational domain additionally maintains the range of relations among variables. Since non-relational domain maintains less information, it is scalable but lacks precision compared to relational domain. Maintaining path-wise information of variables along different but limited paths will combine benefits of both the domains and should help to match the precision of complex relational domains while retaining scalability. This idea, termed as trace partitioning, is proposed in [7].
We have adapted trace partitioning and have developed a static analysis tool-chain which was used to analyze 10 MLOC industry code for AIOB property. We have extended standard interval domain [1] and power set domain concept [15]. We maintain variables' value ranges along subsets of paths, for each subset. Keeping path-wise value ranges implicitly provides inter-variable relationships as a correlation. We keep a configurable limit, *K*, on the number of subsets of paths for which we maintain this information. When number of paths at a program point exceeds *K*, we partition the set of paths into *K* subsets and information is computed with respect to each of these K subsets. Paths are partitioned arbitrarily. The space complexity of using proposed domain is $O(Kpn)$ for a program of size *p*, with *n* variables. In practice, since *n* is quite large compared to *K* ($n >> K$), this complexity is linear in *n* and scalability is comparable to simple interval domain analysis. Further, we have observed that when K is configured based on the program size and computing/storage size, it gives better results than a fixed *K*.
We conducted three experiments. First one showed that our tool-chain can analyze 10 million lines of industry code. Program analysis tools such as Polyspace (version 7) [8] and Astrée (version 11.08) [9] could not run on this code because they do not scale beyond 100KLOC [3] and 1 MLOC [14] respectively. We used the ratio of number of warnings to the number of review points, for a given property as an indicator of precision of the tool. In the second experiment we observed effect of various values of *K*. The last experiment was to compare precision with other tools. This comparison was done on an industry module of 40KLOC for an AIOB property. Astrée reported 166 warnings while our tool and Polyspace (at highest precision) reported only 13 warnings.
In this paper, we

- explain precise range analysis with an example
- present a tool-chain that was used to analyze 10 MLOC
- demonstrate higher precision achieved without loss of scalability on large industry code
- show effects of configurable limit, K, on industry code
- compare precision of our tool with Polyspace and Astrée

We believe this will help practitioners to understand the benefits of K-limited path sensitive interval domain.

2. EXAMPLE
To illustrate the idea, we present an example in Figure 1 extracted from a real life battery controller application.
To reach the point after line 12 there are three paths namely *P1*: (3, 6, 9, 10, 11, 12), *P2*: (3, 6, 7, 8, 11, 12) and *P3*: (3,4,5,12). Numbers in parenthesis denote line numbers of statements on the path.
In Table 1, we show the results computed using three different domains - simple interval domain, our domain (with K=3), and

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
Copyright is held by the author/owner(s). Publication rights licensed to ACM.

ESEC/FSE'13, August 18–26, 2013, Saint Petersburg, Russia
ACM 978-1-4503-2237-9/13/08
http://dx.doi.org/10.1145/2491411.2494569

```
1. int arr[100];
2. void func ( unsigned int v) {
3.   unsigned int f =0;
4.   if (v >0 && v <25) {
5.      ...; // some code not modifying v and f
6.   } else {
7.      if ( 0 == v){
8.          ...; // some code  not modifying v and  f
9.       } else {
10.          f =1 ;
11.      }
12.     }
13.   if (0==f)
14.      if ( 0 != v) {
15.         if ( arr[ (v-1)]  >= 255)   ...; // some code } }
```

Figure 1. Example

Table 1. Different Domain Analysis of Example Program

Line	Interval domain	Our domain	DBM domain
4	(<f,[0,0]>, <v,[1,24]>)	{P3:(<f,[0,0]>,<v,[1,24]>)}	1≤v-f≤24
6	(<f,[0,0]>, <v,[0,M]>)	{P1:(<f, 0,0]>,<v,[0,M]>) }	0≤v-f≤M
7	(<f,[0,0]>, <v,[0,0]>)	{P2:(<f,[0,0]>,<v,[0,0]>)}	v-f=0
9	(<f,[0,0]>, <v,[25,M]>)	{P1:(<f,[0,0]>, <v,[25,M]>)}	25≤v-f≤M
10	(<f,[1,1]>, <v,[25,M]>)	{P1:(<f,[1,1]>, <v,[25,M]>)}	24≤v-f≤M-1
11	(<f,[0,1]>, <v,[0,M]>)	{P2:(<f,[0,0]>,<v,[0,0]>), P1:(<f,[1,1]>,<v,[25,M]>)}	0≤v-f≤M-1
12	(<f,[0,1]>, <v,[0,M]>)	{P2:(<f,[0,0]>,<v,[0,0]>), P3:(<f,[0,0]>,<v,[1,24]>), P1:(<f,[1,1]>,<v,[25,M]>)}	0≤v-f≤M-1
13	(<f,[0,0]>, <v,[0,M]>)	{P2:(<f,[0,0]>,<v,[0,0]>), P3:(<f,[0,0]>,<v,[1,24]>)}	0≤v-f≤M-1
14	(<f,[0,0]>, <v,[1,M]>)	{P3:(<f,[0,0]>,<v,[1,24]>)}	1≤v-f≤M-1

Difference-Bound Matrices (DBM) [4], a relational domain. DBM, along with ranges of individual variables, tracks differences between every pair of variables in a program**.** So, for example, in Figure 1 DBM domain maintains ranges of values of v, f and v-f while simple interval domain only maintains ranges of values of v and f. In this example, ranges for v and f are identical in DBM and in simple interval domain. We have shown values of *v* and *f* in the Interval domain column and values of *v-f* in the DBM domain column in Table 1. In addition to v and f, DBM will compute ranges of v-f at different program points (although, in this example, there is no relationship between v and f). In our approach, at the end of line 11, we keep information for *P1* and *P3* and at the end of line 12, we keep it for *P1, P2* and *P3.* As a result, at line 14, we carry information only along *P3*, because *P1* and *P2* cease to extend from line 13 and line 14 respectively. Thus at line 14, we safely conclude that *v* can only be in the interval [1, 24] whereas the other two approaches conclude that *v* can be in the interval [1, M], where M denotes the maximum value of an unsigned integer variable. Hence, we can precisely infer that array access at line 15 is safe but other approaches will be imprecise and report it as may-be-unsafe.

3. APPROACH

3.1 Background

1) PRISM*:* our in-house data flow analysis framework which takes specification of a data flow problem to be solved as input and generates a data flow analyzer for the specified problem. Specification consists of transfer functions for different constructs, meet operation and optional widening operation.

2) TECA*:* a static analysis tool [11] which is based on PRISM framework and verifies different properties such as zero division, AIOB, underflow-overflow etc.

3) Clustering*:* a process to divide a large code base into small clusters using the algorithm presented in [6]. Each cluster consists of one top-level function that is not invoked within the code base and all the functions that are directly or transitively invoked from the top-level function. The result of analyzing the complete code base is the same as merging the analysis results of individual clusters.

3.2 The Tool-Chain

We have developed a tool-chain that implements clustering, K-limited path analysis and property verifier. The tool-chain is implemented in JAVA and its architecture is depicted in Figure 2.

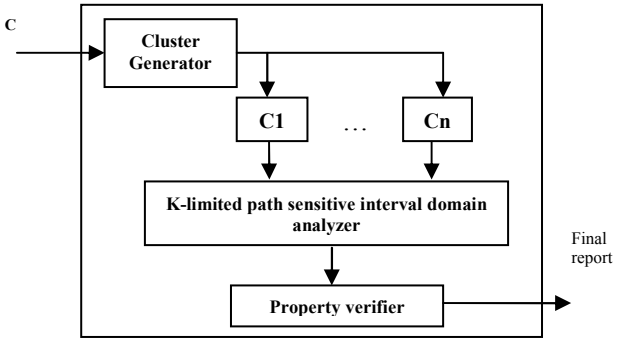

Figure 2. Tool Chain Architecture

1) Cluster Generator*:* The code is divided into several clusters using the clustering algorithm.

2) K-limited path sensitive interval domain analyzer*:* K-limited path sensitive interval domain, as explained earlier, is specified as a data flow problem for PRISM. For specifying this we have used some ideas from [1] for transfer functions of assignments involving complex arithmetic expressions and relational operators. In order to get more precise intervals in cases such as, bit-wise operations and linear inequalities, we have developed our own strategies.

The proposed domain is an extension of interval domain [1]. We extend the abstraction to enable path sensitivity. Instead of keeping a single interval for each variable in isolation, we maintain a set of variables to range (interval) mappings. Number of such mappings is limited to *K*, which is chosen based on the program under analysis. Each mapping represents abstraction over a subset of paths and every path is included in some mapping. Choice of a particular partitioning of set of paths, when there are more than *K* paths, is arbitrary.

This analysis guarantees that range of a variable in a map represents the abstraction of values of that variable when control reaches through any of the paths of corresponding path subset. This way, analysis performed is conservative and interval for a variable will have all possible values, therefore it is sound. Formal proof of soundness is out of the scope of this paper.

To resolve pointers, we compute and use flow insensitive points-to information [2]. Abstract interpretation needs widening when underlying abstract domain is not finite. Widening strategy with K-limited path sensitive interval domain used in our tool is described in the next sub-section. PRISM generated data flow analyzer is run on each cluster as a single unit and results are passed on to the property verifier.

3) Property verifier*:* TECA verifies the code for AIOB property using the analysis results of K-limited path sensitive analyzer. At property checking point, a check is made against each mapping of variables to ranges computed at that point. As explained earlier, there could be at most K such mappings. When the property is satisfied against all mappings, it is considered safe. As the underlying value range analysis is sound, this property verifier doesn't miss any property violation. It verifies each cluster individually and creates a final combined analysis report.

3.3 Widening and Loop Unrolling

In abstract interpretation, presence of loops in a program requires widening [1] to terminate the analysis. In our tool, we use the concept of widening operator as defined in [1] to perform widening. We compute information at each program point as a set S of maps M_1, M_2, ..., M_t where $t \leq K$. Each M_i is a map from a variable to its range value. Let S_1= {M_{11}, M_{12}, .., M_{1h}} and S_2 ={ M_{21}, M_{22},., M_{2r}} , where $h \leq K$ and $r \leq K$, be the information computed at a loop head in two successive iterations of the loop. To compute widening of S_1 with S_2, first we compute a map M_2 as merge of maps M_{2j} where $1 \leq j \leq r$. Elements of widening result are computed by widening individual elements of S_1 with M_2. Widening so defined, satisfies the property of stabilization to guarantee the termination. Proof of stabilization is out of scope of this paper.

Since widening results in imprecision, we unroll the loop for a fixed number of iterations before applying widening. In many cases, the unrolling itself leads to fix point results and therefore imprecision due to widening is avoided. The number of times a loop is to be unrolled, called widening parameter, is also configurable in our tool. Increase in value of this parameter increases precision.

Consider the code snippet in Figure 3. Here division expression is reported as safe when K is 7 and widening parameter is 3. This is because information at the loop head will reach a fixed point in just three loops unrolling and widening will not be needed. Had we applied widening without unrolling the loop, the range of c would have been [0, 255] and division expression would have been reported as possibly unsafe.

```
unsigned char c; unsigned int a,b;
scanf("%d", &b); a = 0;  c = 0;
while(a<3) {
  if (b<=4) c++;
   a++;
}
if (b <5) z = t/c  // division expression
```

Figure 3. Example

Similar to standard interval domain [1], analysis using K-limited path sensitive interval domain becomes imprecise after applying widening. However, experiments shared show that with this domain we could get better precision than Astrée and it remained at par with Polyspace which implements more complex domains.

4. EXPERIMENTAL SETUP AND RESULTS

The experiments were conducted on two real-life automotive applications using dual core 2.26 GHz processor with 2.0 GB of RAM. Both the applications were in C language and were tested completely. First application (A1) of 10 MLOC implements the navigation functionality of an automotive. It was first divided into 94 clusters (C1, .., C94); smallest cluster of 11 KLOC and largest cluster of 2.1 MLOC. The second application (A2) implements battery controller functionality. It was not divided into clusters.

4.1 Scalability

A1 was used to demonstrate the scalability of our tool-chain. We successfully ran our tool on all 94 clusters with the value of *K* as 3. Total end-to-end execution time of our tool-chain was 7 hours. A1 is a client application and the client is very happy with the precision and the scalability of the tool-chain.

Polyspace and Astrée couldn't run on this code as these tools cannot scale up beyond 100 KLOC and 1 MLOC respectively. As an indicator of the precision of our tool, we measured the ratio of number of warnings to number of review points for a given property. We computed this ratio on 4 clusters of varying sizes of A1 for AIOB property. Table 2 describes these results and we can see that the analysis precision is maintained even with increase in size of clusters.

Table 2. Precision Results

Cluster	LOC	Precision ratio
C1	110K	1:11
C2	224K	1:9
C3	311K	1:10
C4	2.1M	1:12

4.2 Configuration of K

To assess the effect of increasing K, that is the number of path sets, the tool was run with different values of *K* on second application, A2, and on four different clusters of A1. In each run, we measured the number of array accesses which were reported as safe, unsafe or may-be-unsafe. As expected, we observed that some may-be-unsafe array accesses were converted into safe or unsafe with increase in value of *K*. We show our results in Table 3. In column 1, A2 represents second application of 40 KLOC and C1-C4 represent different clusters of first application A1. Results show that, the precision increases with larger value of *K*. Table 3 shows the percentage improvement in precision(percentage reduction in may-be-unsafe array access) with respect to *K*=1 (all paths merged together). For example, if the warnings reported by *K*=1 and *K*= 3 be W1 and W3 respectively, precision improvement cell for *K* =3 is computed as (W1-W3 / W1) * 100. Similarly, it is computed for other values of K. For different values of *K*, it also shows the analysis time (in minutes).

Table 3. Experimental Results

App	LOC	Precision improvement (%)			Analysis time (min)		
		K=3	K=10	K=15	K=3	K=10	K=15
A2	40K	0.75	4.01	24.9	1.8	1.9	2
C1	110K	2.86	3.1	3.1	2.1	2.3	2.5
C2	224K	2.07	2.3	6.5	7	7.2	7.8
C3	311K	1.5	4.2	4.2	7.5	8	8.3
C4	2.1M	2.9	4.2	4.4	56.7	58.3	61.7

An important point to note is that there is no significant increase in the analysis time when value of *K* is increased. This clearly highlights the practicality of our approach.

4.3 Comparison with Astrée and Polyspace

A2 (of 40 KLOC) was selected for comparative study of precision as both tools scaled on this application. We analyzed this application's result for AIOB property using Astrée, Polyspace and our tool with K=15. Results are summarized in Table 4.

Results show that our approach produced more precise results than Astrée. It demonstrates that while Polyspace uses complex polyhedral and some other complex relationship inferring

algorithms, in practice, our approach is as precise as Polyspace. It also shows our tool runs much faster than the other two.

Table 4. Comparison Report

Tool	May-be-unsafe warnings	Time (min)
Astrée	166	21
Polyspace	13	360
Our approach	13	10

5. LIMITATIONS

1. Precision of our domain is less than DBM domain in certain cases. Consider this example:

for (i=0, j=0; i< 10; i++) { sum = sum + arr [j]; j++; }

Assume that an array *arr* is of size 10. According to our domain, with *K*=3 and widening parameter as 3, we will infer that at array access point the value of *i* is in the interval [0, 9] and that of *j* is in the interval [0, MAX]. On the other hand, using DBM domain, which is relational domain, relation *i-j* will be tracked and value of *j* will correctly be inferred to be in the interval [0, 9]. Thus, in such cases, relational domain will be more precise than K-limited path sensitive interval domain.

2. Due to widening, the effect of *K* path sets tracked before any loop gets diluted after the loop. Consider an example shown in Figure 4. Let *K* be 2 and the default widening parameter be 3. Here, division expression is reported as potential division by zero error because after the loop, g's range will be considered as [0...20] at division point. However, K paths are tracked after the loop as well, concluding the last array access to be safe.

```
unsigned int a,x,g;  int arr[100];  scanf ("%d", &a); x = 255;
if (a < 10)  g =20; else g =0 ;
while(x != 0) {   x--; }
if (a < 8)  {
     z = t/g;  // division expression – range of g is [0..20]
     g = 100; }
if (a >= 50) arr[g] = 40;   // Array access  - range of g is [0..20]
```

Figure 4. Example

6. RELATED WORK

Tools Coverity and UNO perform standard interval analysis and scale to million lines of code but with unsound analysis [13]. These tools do not track any branching correlation. Frama-C's value analysis plug in [5] computes sets of possible values for the variables in a program; however, it does not track branching correlations among variables.

Polyspace [8], a commercial tool, performs sound analysis at different levels of precision which is configurable. At lower precision level, it performs standard interval domain analysis. And at higher precision level configuration it uses some complex algorithms along with complex relational domain and is able to track some of branching correlation, as done by our approach. However, its scalability is limited to 100 KLOC [3, 13] while our approach could run on million lines of code.

Astrée [9, 10] performs sound, precise and scalable analysis for the family of programs it was designed for. It is scalable because it does not use complex polyhedral domain (uses only octagon domain). But as shown through experiments, it becomes less precise for a significant set of practical scenarios.

Our extension is based on the trace partitioning idea [7] which is also implemented in Astrée. However, in Astrée its effect to track branching correlation can be seen by specifying directives in user code. Our implementation does not require any such modification.

7. CONCLUSION

Using trace partitioning idea from [7], we extended standard interval domain by introducing path sensitivity and limited the number of paths to be maintained to K. We have demonstrated better precision along with scalability by successfully analyzing a system of 10 MLOC. Through experiments, we observed that the relationship we are tracking is not tracked by other tools that use relational domain analysis. We have shown that a configurable *K* helps produce precise results with acceptable scalability so as to make the technique useful for analyzing large industry applications. Therefore, even in presence of commercial tools like Polyspace and Astrée, due to better scalability with precision, our tool chain is successfully used by one of the leading automotive OEM (Original Equipment Manufacturer) on their large applications.

8. REFERENCES

[1] P. Cousot, R. Cousot. Static Determination of Dynamic Properties of Programs. *In proceedings of the 2nd international symposium on programming*, pages 106-130, 13-15 April 1976, Dunod, Paris.

[2] M. Das. 2000. Unification-based pointer analysis with directional assignments. *In Proceedings of the ACM SIGPLAN PLDI '00*. ACM, 35-46.

[3] P. Emanuelsson, U. Nilsson. 2008. A Comparative study of Industrial static analysis tool. *ENTCS*. 217, 5-21.

[4] A. Mine. 2001. A New Numerical Abstract Domain Based on Difference-Bound Matrices. *In Proceedings of the 2nd Symposium PADO'01*.Springer-Verlag, London, UK, 155-172.

[5] http://frama-c.com

[6] S. Khare, S. Saraswat and S. Kumar. 2011. Static program analysis of large embedded code base: an experience. *In Proceedings of the 4th ISEC '11*. ACM, 99-102.

[7] L.Mauborgne, X. Rival. 2005. Trace partitioning in abstract interpretation based static analyzers. *In proceedings of 14th ESOP'05*. Springer-Verlag, Berlin, Heidelberg, 5-20.

[8] Polyspace. http:// www.mathworks.in/products/polyspace/.

[9] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Mine, D. Monniaux, X. Rival. 2005.The Astrée Analyzer. *In proceedings of 14th ESOP'05*. Springer-Verlag, Berlin, Heidelberg, 21-30.

[10] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Mine, D. Monniaux, X. Rival, D. Monniaux. 2007. Varieties of static analyzers: A comparison with Astrée. *In proceedings of 1st Symposium TASE*'07. Pages 3-20.

[11] TCS Embedded Code Analyzer. http://www.tcs.com/resources /brochures/Pages/TCS_Embedded_Code_Analyzer.aspx

[12] P. Cousot, R. Cousot. Abstract Interpretation: A unified lattice model for static analysis of programs by construction or approximation of fix-points. 1977. *In proceedings of 4th ACM* SIGACT-SIGPLAN *symposium POPL'77*. ACM, 238-252.

[13] C. Bharti. 2012. Survey of array out of bound access checkers for C code. *In Proceedings of the 5th ISEC'12*. ACM, 99-102.

[14] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Mine, and X. Rival. 2009. Why does Astrée scale up? *In Journal, Formal Methods in System Design, 35, 3, 229-264.*

[15] B. Gulavani, S. Rajamani. 2006. Counterexample Driven Refinement for Abstract Interpretation. I*n proceedings of TACAS'06, LNCS 3920, 474-488.*