Research on Complicate Array Object via Region-Based Memory Model

Jing Wang, Yukun Dong, Dahai Jin, YunZhan Gong

State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China

wjsmiling@126.com

Abstract – Array is widely used in software development, but in static analysis, array cannot be perfectly processed. There exists a series of problems, such as incompletely representation of the associations of different memory units. In this paper, a memory model Symbolic Region Model (SRM) is proposed. It can describe the shape of array objects, different kinds of memory states of array object and relations of array variables. The relations include alias relations, hierarchical relations inside a array object and logic relations between array and pointer variables. Then to improve the precision of array object analysis, a SRM based analysis method is proposed, which is flow sensitive, field sensitive and context sensitive, to analyze the shape, dataflow and point-to relationship at every procedure point. According to the experimental results of a large number of open source projects, the method proposed in this paper can improve the accuracy to process array in the interval computation of static analysis.

Index Terms - array object, memory model, static analysis, region-based

1. Introduction

Software analysis includes static analysis and dynamic analysis. Using static analysis, we can evaluate the program result without running the program; it predicts the situation of the software in the future through current state and tendency.

Only if the program is in running state, we can accurately get the value of the variables and the expressions, so internal computation is widely used in software defect testing field to calculate the range of the values of variables. In this way, we can effectively improve the accuracy in static analysis. But there are still two challenges: one is how to accurately represent the storing information of the memory object and completely describe the relations between different variables; the other is how to use abstract methods to describe different program semantics.

To detect the defects of code, we usually get the program running state by static analysis, determining the running state satisfying the safe attribute or not to ensure whether there exist defects. But static analysis cannot analyze all attributes of the program, it can only get the approximate result. The accuracy of static analysis determines the accuracy of the array object.

Array object analysis is complicated, especially for C programs mainly because of the alias relations, hierarchical relations inside a compound type variable and logic relations. If we cannot completely represent these relations, it will reduce the analysis accuracy. In order to totally represent the relations between variables, we need to abstract the memory model. One way is based on the purpose of the static analysis and the needing accuracy, just describing the relations between a few array objects in memory blocks. For example, big array model is OK, but it can hardly describe the hierarchical relations, shape diagram. TVLA [1] are regardless of the relations of the store information in the memory units Its or The point-to diagrams only describe the point-to relations. And models based on region use a region which is represented by a continuous memory block of a memory object. STVL [2] considers the logic relations of value of variables, but it cannot describe the hierarchical relations of compound type data structure. Another way is to limit the testing program requirements, such as no memory allocated for the analyzed program, and assumes all types are completely safe.

To solve these problems, in this paper, a method is proposed, Symbolic Region Model (SRM). It is used in static analysis of C programs. SRM can describe different storing information of objects in physical memory, the hierarchical relations between different storing units, the point-to relations, and the logic relations. Based on SRM, we can map the variable operation to the analysis method that is region operated, flow sensitive, field sensitive and context sensitive.

To prove the method effective, we did some experiments on the Defect Testing System and analyzed some open source projects. The result indicates that the method proposed in this paper can obviously improve the accuracy of array analysis and reduce potential false positives.

2. Symbolic Region Model

A program P can be represented by six tuple(V, L, S, τ, $init$, end). Among them, $V \in$ Vars, is the variable set of program P; L is the set of program point; $S \in$ Stmts, is the set of program sentences; $\tau \in L \times S \times L$, represents the transfer relationships; $init \in L$, is the start point of program P; $end \in L$, is the end point of program P.

The main job of static analysis is analyzing the concrete semantics and the program state. The program state sets S can be divided into two parts, control parts C and data parts D, $S := C \times D$. Among them, control parts C includes the next program point that will be executed, data parts D includes the storing state in memory block. The storing state is the state sets of memory object in that program point and includes the address information of memory unit represented by left value and the value information that is represented by right value.

Worklist [3] algorithm is one classical completion of iteration algorithm It starts from the program beginning point

init, based on the transfer relation τ, analyze every storing information of variable $v \in V$ on every produce point $l \in L$. It will end until the program end *end*. If one static analysis method *A* is going to analyze any program *P* that is developed by language *L*, and the memory unit value that is computed in the any program point *l* of *P* contains every probable value in running state, we can tell that *A* is reliability.

Reliable static analysis will analyze the program state in every program point, not only conservatively analyze all the storing information of memory object. But array object is widely used, and it includes the alias relation, hierarchical relation, logic relation. Besides, there are pointer operations, dynamic memory allocation with array object. These increase the difficulty in analyze the array structure object in C program.

2.1 Motivation

There is an example bellow:

```
L1: struct s1 { int d; } data;
L2: struct s2 { struct s1 *p; } *sp;
L3: void f2(int i) {
L4:    int a[2] = {0, 0}
L5:    sp = malloc(sizeof(struct s2));
L6:    sp-> = &data;
L7:    sp->p->d = 3;
L8:    if (i != 0)
L9:       i = 1;
L10:   a[i] = data.d;
L11:}
```

When the program executes after L6 sentence, *sp->p->d* on sentence L7 has alias relation with *data.d*. After execute L7 assign operation. The value of *data.d* will also be 3. But after L8, L9 branch sentences, the value of *i* on L10 will be 1 or 2, if we do not use sensitive path analysis, the one that a[i] is assigned may be a[1] or a[2]. In this way, we can only process weak update operation.

From this code segment above, we can see that, there are more complicated expressions in C program, and there are many different relations between these expressions. Besides, when these expressions are with complicated array object, the analysis will be very difficult. To accomplish accurate analysis in array object in C program, static analysis needs to accurately describe different expressions that are with array structure object, and it can also describe the memory state of the running program, then, it can get the vary relations between different variables. When it involves with function, we need to consider the context environment on the call point.

2.2 Addressable Expression

An expression is a sequence composed by operators and operands, defines computation to a value. The C programming language standard classifies expressions into l-value and r-value. An expression's l-value is the memory location of its associated memory object, and an expression's r-value is the value associated with the memory object. Some expressions have both l-value and r-value, for example integer variable *a*, and array element *arr[0]*; some expressions only have r-value, for example *a*b*, and *c||d*; only array variable have l-value but no r-value.

Definition. Addressable Expression. Array or an expression that has l-value and which can be assigned.

Some expressions have l-values, but can't be assigned, for example *p+2*, where *p* is a pointer.

For all types of expressions defined by C99, we describe a C addressable expression *aexp* by the following grammar:

aexp::= *id* | *aexp.f* | *aexp->f* | *aexp*[*exp*] | (*aexp*) | **aexp* | *id*(*exp*)

aexp* can be defined as: **aexp*::=aexp*' | *(++*aexp*') | *(--*aexp*') | *(*aexp*'++) | *(*aexp*' --) | *(*aexp*' *op exp*'), the type of *aexp*' is pointer, *op*= + | -, the type of *exp'* is integer.

For *id*(*exp*), where *id* is a method and return type is pointer, *exp* means parameters.

The syntax of C can be described by BNF; expressions closely related to C addressable expression are UnaryExpression, PostfixExpression, and PrimaryExpression. These three expressions are defined as follows:

```
UnaryExpression::=
    PostfixExpression
    | "++" UnaryExpression
    | "--" UnaryExpression
    | UnaryOperator CastExpression
    | <SIZEOF> ( UnaryExpression | "("TypeName") ")
PostfixExpression ::=
    PrimaryExpression (
    "["Expression"] "
    | "("( ArgumentExpressionList )? ") "
    | ". " <IDENTIFIER>
    | "->"<IDENTIFIER>
    | "++" | "--")*
PrimaryExpression ::=
    <IDENTIFIER> | Constant |"("Expression")"
```

And UnaryOperator is defined as:

```
UnaryOperator::= "&" | "*" | "+" | "-" | "~" | "! ")
```

As an intermediate representation of source code, AST is built by syntax analysis, the node corresponding to code block of program, and every expression mapped to a node of AST, so we can recognize expressions from AST nodes. If we can recognize all nodes that express addressable expressions, then we can recognize all addressable expressions.

2.3 Symbolic Region Model

If we want to completely accurately analyze all array objects storing information in memory block, we should consider all the relationships between addressable expressions. Region-based three-tuple <*Var*, *Region*, *Value*> only considers the alias relation and hierarchical relation, *Value* can only represents the concrete value or region number, it does not consider the value logic relation, and this model only use in one path analysis, STVL [2] is a three-tuple model<*Var*, S_{Exp}, *Domain*>, it considers the value logic relation, but it does not consider the hierarchical relation and the alias relation. In this paper, we propose a advanced approach, Region-based Symbolic Four-Valued (SRM), a model based on region.

Symbolic Region Model is four tuples, SRM = <*Var*, *Region*, S_{Exp}, *Domain*>, Among them, *Var* represents memory objects, *Region* represents region, S_{Exp} represents symbolic expression, *Domain* represents the range of the value.

Four tuples Symbolic Region Model (SRM) is used to describe array type structure object in memory block. Complicated type memory object can separate into simple type memory objectWe can use three tuple <*Var*, *Region*, χ> to describe this. *Var* is array type structure object, χ is {<*i*, *Region*>}, $i \in N$ which is the index of the array structure object. It can also be used in more complicated type structure.

For different type objects in memory, SRM use different type region to describe their storing state. *PrimitiveRegion* is used to describe the basic type memory object, *PointerRegion* is used to describe pointer, *ArrayRegion* is used to describe array structure, *StructRegion* is used to describe structural structure. Every region has only one number, the number of *PrimitiveRegion* is like *bm_i*($i \in N$), the number of *PointerRegion* is like *pm_i*, the number of *ArrayRegion* is like *am_i*, the number of *StructRegion* is *sm_i*, for dynamic allocated anonymous memory, we can use *mxm_i_n* (*x* represents the type of region, value is *b*, *p*, *a* or *s*) to describe this region, *n* is the number of byte that is allocated for this anonymous memory. The number of null address region is *null*, the number of wild region is *wild*.

We call the region that is allocated for memory object is safe region, and the region that is allocated dynamically is dynamic region. We call these two regions operated regions. We call the region that is flagged by *null* and *wild* not operated region.

Symbolic expression S_{Exp} is construed by symbol that is operated by mathematical operation and relation operation, the definition of recursion is:

$S_{Exp} \rightarrow Rel_{Exp} \mid \neg Rel_{Exp} \mid Rel_{Exp}\ £\ Rel_{Exp}\ £ \in \{\&\&, \|\}$

$S_{Exp} \rightarrow Exp \mid Exp\ \xi\ Rel_{Exp}\ , \xi \in \{<, \leq >, \geq ==, \neq\}$

$Exp \rightarrow Term \mid Term \pm Exp$

$Term \rightarrow Power \mid Power \times Term \mid Power \div Term$

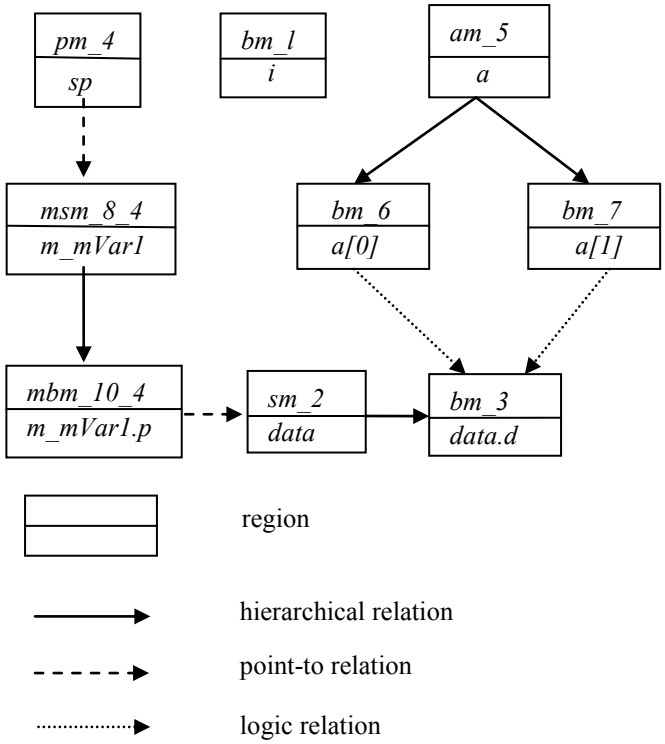
$Power \rightarrow Factor \mid Factor^{Power}$

$Factor \rightarrow Constant \mid Symbol \mid (S_{Exp})$

Among them; the symbolic expression S_{Exp} is constructed by logic expression Rel_{Exp} through relation operation; Rel_{Exp} is constructed by mathematical operation *Exp* through logic operation; *Exp* is constructed by *Term* item through add and subtract operation; Term is constructed by many factor Power through multiply and divide operation; Each Power is constructed by one or more atomistic Factor through power operation; Factor is the most basic element of symbolic expression, it can be an constant value Constant, symbolic variable Symbol or symbolic expression; every Symbol maps to a addressable expression.

For the intervals of value, we can use interval abstract domain method. Every *Symbol* is represented by interval. The text interval is abstract by interval set; it divides into two big parts, value type interval and pointer type interval. Although interval abstract domain is not relation type, we can use symbolic expression to describe logic relation.

In the code segment before, after execute L10 sentence, the region relation will be like



SRM can divide the memory into discrete region; it can describe the point-to relation, hierarchical relation and the logic relation.

3. Experimental Results

We have implemented the approach presented in this paper in DTSC, a defect testing system for C source codes written in Java developed in our laboratory. DTSC mainly has seven steps: creating Abstract Syntax Tree(AST), generating scope table, function calling analysis, generating control flow graph, calculating def-use chain, data flow analysis, detecting defects. In the second step, all C addressable expressions can be recognized and stored in scope table.

In the experiment, we use two different analysis methods. One is the ordinary method, the other is the method proposed by this paper

As shown in Table 1, we analyze three projects, with a total source code line of 62476, intermediate code line of 269368. Method 1 cost 831 seconds, and it reported 1364 inspection point, The method proposed by this paper cost 1000 seconds, and the inspection point was reduced to 1219. The number of false positives reduced by 10.63%.

Because the analysis complexity is improved, the analysis time costs a little longer, which will reduce the efficiency.

But we can see that if we use the method proposed in this paper, we can obviously improve the accuracy of complicated array object.

TABLE I experimental result

project name	num-ber of source files	num-ber of sourcecode lines	Number of interm-diate source code file lines	ordinarymethod		Method in this paper		accuracy
				time	point	time	point	
combine	74	16701	23892	270	401	344	288	113
readline	74	21460	109775	172	498	191	490	8
antiword	80	24315	126701	389	465	459	441	24

4. Related Work

There are so many different static analysis and different applications. For them, there are already thousands of models to describe memory state of running program. The simplest is name-value. It does not consider the relationship between memory units. Then, the big array model, it cannot represent the hierarchical relation between memory unit. alias can be useful to point-to relation set, we usually use it in pointer analysis. Shape diagram and TVAL is useful in shape analysis, we usually use it in determine the data structure shape information in memory on running state. Region concept is usually used in memory management. In static analysis field, region based model uses a region that represents a continuous memory block of a memory object allocated in memory.

To improve the accuracy of static analysis, we usually use sensitive analysis method. It includes flow-sensitive, domain-sensitive, context-sensitive, path-sensitive. Flow-sensitive is used to analyse every program point and save the program state. Worklist algorithm is the most classical iteration algorithm. Domain-sensitive can guarantee the analysis accuracy for compound type variable.

5. Conclusions and Future Work

In this paper, we proposed a method based on Symbolic Region Model (SRM). It describes the shape of array structures, different kinds of array structure memory states and relations of array variables including alias relations, hierarchical relations inside a array and logic relations between array structure variables and pointers. It is flow-sensitive, field-sensitive and context-sensitive, to analyze the shape, dataflow and point-to relationships at every procedure point. Experiments show that our static analysis can guarantee higher precision on the array data structure object of no efficiency loss.

In the future, we plan to improve the naming rule of add-erssable expressions, and apply this approach in sound data flow analysis and fully null pointer dereferences detection.

Acknowledgment

This work is supported by the High Technology Research and Development project in China (National 863 Project in China) under Grant 2012AA011201 and The National Natural Science Fund (91318301, 61202080).

References

[1] T. Lev-Ami, M. Sagiv. TVLA: A system for implementing static analyses. In Proceedings of the 7th International Static Analysis Symposium. LNCS 1824, Berlin: Springer-Verlag. 2000: 280-301

[2] Zhao YS, Wang YW, Gong YZ, Chen HH, Xiao Q, Yang ZH. STVL: Improve the Precision of Static Defect Detection with Symbolic Three-Valued Logic. The 8th Asia-Pacific software Engineering Conference. Washington: IEEE Computer Society, 2011.

[3] Keith D. Cooper, Timothy J. Harvey, Ken Kennedy. Iterative Data-flow Analysis, Revisited. Proceedings of the ACM SIGPLAN 2002 conference on Programming language design and implementation. NewYork: ACM Press, 2002

[4] S. Muchnick, Advanced compiler design and implementation: Morgan Kaufmann, 1997

[5] Zhaohong, Gong YunZhan, Xiao Qing, et at. Variable Range Analysis Based on Interval Computation. Journal of Beijing University of Posts and Telecommunications, 2009(03)

[6] J. King, "Symbolic execution and program testing," Communications of the ACM, vol 19, pp, 385-394, 1976

[7] P.Godeforid, "Compositional dynamic test generation," 2007, pp.45-54

[8] Cousot P, Cousot R.Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints[C]//Proc of the 4th POPL, January 17-19, 1977.Los Angeles: ACM Press, 1977: 238-252

[9] Yang Zhaohong, Gong Yunzhan, Xiao Qing, et al. A defect model based testing system. Journal of Beijing University of Posts and Telecommunications, 2008, 3l(5): 1-4

[7] B. Blanchet, P. Cousot, R, Cousot, J. Feret, L, Mauborgne, A. Mine, D. Monniaux, X. Rival. A static analyzer for large safety – critical software. In Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation. San Diego, California, USA: ACM Press, 2003. 196-207. [doi: 10.1145/780822. 781153]