*2012 IACSIT Hong Kong Conferences*
***IPCSIT vol. 30 (2012) © (2012) IACSIT Press, Singapore***

# Research on Theorem Proving of Pointer Logic

Zhang Yuping [+] and Xu Xiaoyu

National Lab of Software Development Environment, Department of Computer Science,
Beihang University, Beijing, China

**Abstract.** We propose logic rules for pointers in C language to avoid dangling pointer and array index out of bound. A certain kind of dangling pointers will be detected by our algorithm. We firstly build pointer trees for dynamically allocated memory, and then update pointer information on these trees according to pointer operations. Dangling pointers will be detected with information on pointer trees. Array index out of bound will be checked by logic rules, which are proved by Coq. We test several typical programs. The result shows that our method can check dangling pointers and array index out of bound, which can be a supplement to Visual C++ compiler.

**Keywords:** pointer logic, dangling pointer, array index reference, Coq.

## 1. Introduction

Pointer is the treasure of C language which makes C programming flexible. Meanwhile, pointer is the disaster of C programming because of memory leak and other exceptions caused by pointer. 2009, C. A. C. Hoare spoke in QCON that the invention of null reference in 1965 was a billion-dollar mistake. Null reference, which was designed by him for ALGOL W, caused a billion dollars of pain and damage in the last forty years [1]. Microsoft uses Prefix and Prefast to check references. Hoare logic, separation logic, shape analysis are studied for pointer safety verification. However, these theories have different problems in verification. In this paper, we firstly give formal definitions of different pointers, which would make pointer problems clear. Secondly, we propose an algorithm to check a certain kind of dangling pointers.

## 2. Formal Definitions of Pointers

Pointer can be defined by function, whose domain is its address, and range is its content, which is another address.

**Definition1** Assume that $M$is the memory set, if $f$fulfills $\phi_1 \wedge \phi_2 \wedge \phi_3$, which

$\phi_1$: $\forall x(x \in f \rightarrow \exists uv(x = \langle u, v\rangle))$

$\phi_2$: $\forall uvw(\langle u, v\rangle \in f \wedge \langle u, w\rangle \in f \rightarrow v = w)$

$\phi_3$: $\forall x(x = \langle u, v\rangle \in f \rightarrow u \in M \wedge v \in M)$

then$f$is called pointer function.

**Definition2** Given a pointer function$f: A \rightarrow B$, if $x = \langle u, v\rangle \in f$ and $u \in A$, $v \in B$ , then $x$ is a pointer object.

**Definition3** A pointer variable is a name to a pointer object in programming language. Assume that there is a pointer object $x = \langle u, v\rangle$, if $p$ is the name of $x$, then $p$ is the pointer variable of $x$, that is $p: x$. The l-value of $p$ is $u$, the r-value is $v$.

[+] Corresponding author. Tel.: +0086-010-82327912; fax: +0086-010-82316736.
*E-mail address*: *zhangyp@nlsde.buaa.edu.cn.*

Table. 1: Declarations of pointers

| Declaration | Name | Address | Content |
|---|---|---|---|
| int i=39; | i | 1001 | 39 |
| int *p=&i; | p | 1003 | 1001 |
| int **q=&p; | q | 1005 | 1003 |

Operation $\&$ can get l-value of $i$ or $p$. The r-value of $p$ is $\&i$, r-value of $q$ is $\&p$.

Table. 2: Expressions of pointer function

| Exp. | Value | Function Expression |
|---|---|---|
| i | 39 | i=f(1001)=39 |
| p | 1001 | p=f(1003)=1001 |
| q | 1003 | q=f(1005)=1003 |
| *p | 39 | *p=$f^2$(1003)=f(1001)=f(p)=39 |
| *q | 1001 | *q=$f^2$(1005)=f(1003)=f(q)=1001 |
| **q | 39 | **q=$f^3$(1005)=$f^2$(1003)=f(1001)=$f^2$(q)=39 |

**Definition4** A null pointer is a pointer which points to NULL. Assume that there is a pointer function $f: A \to B$, if $x = \langle u, NULL \rangle$, that is $f(u) =$ NULL, then $x$ is a null pointer.

**Definition5** A valid pointer is a pointer points to valid memory. Given a pointer function$f: A \to B$, pointer $x = \langle u, v \rangle$, valid memory $B_v \subseteq B$, if $u \in A$ and $v \in B_v$, then $x$ is a valid pointer.

Pointer function is a function defined on computer memory *M*. The two-tuples, of which the function value is memory address, is pointer. Otherwise, the two-tuples is nomal variable or the object the pointer points. Whether the pointer is valid depends on whether the second element of the two-tuples is valid. Valid memory is the memory allocated to the program, which can be used by programmer. Invalid memory is the memory which can not be used by program or programmer, such as the dynamically allocated memory which has been freed.

**Definition6** A dangling pointer is a pointer points to invalid memory. Assume that there is a pointer function$f: A \to B$, pointer $x = \langle u, v \rangle$, valid memory $B_v \subseteq B$, if $u \in A$ and $v \in B - B_v$, then $x$ is a dangling pointer.

**Definition7** Equal pointers are pointers(at least two) point to the same address. Given a pointer function$f: A \to B$, pointer $x_1 = \langle u_1, v \rangle$, $x_2 = \langle u_2, v \rangle$, if $u_1 \neq u_2$, then $x_1$and $x_2$ are equal pointers.

**Definition8** An alias or reference is a substitute of a pointer. Given a pointer function$f: A \to B$, pointer variable $p$ and its pointer object $x$, that is , if $p'$ is an alias or reference of $p$, then $p'$ is a substitute of pointer $p$, that is $v[p/p']: x = \langle u, v \rangle$.

Alias or reference are not real pointers. They are substitutes of pointer variable name, which have to be initialized and can not be reinitialied. $p'$ and $p$ are two names of pointer object $x$, both of which have the same l-value and r-value. However, equal pointers are different pointer objects, who have different l-values and same r-value.

**Definition 9** Array is an ordered group of data objects, all of which have the same data type. The name of array can be regarded as an implicit pointer or reference.

# 3. Pointer Logic

In [2], there is a simple pointer logic which is decidable. We add multiplication into the operation. Adding the multiplication with constants, Presburger algorithm is still decidable.

**Definition 10 (pointer logic)** The sysntax of pointer logic is defined in [2]. The definition of *formula, atom, pointer, term* is the same, while *op* is different.

op: $+ \mid - \mid *$

**Definition 11 (semantics of pointer logic)** Let *L* denote a memory layout and let *M* denote a valuation of the memory. Let $\mathcal{L}_P$ denote the set of pointer logic expressions, and let $\mathcal{L}_D$ denote the set of expressions

permitted by the logic for the data words. We define a meaning for $e \in \mathcal{L}_P$ using the function $[\![\cdot]\!]: \mathcal{L}_P \to \mathcal{L}_D$. The function is defined recursively in [2]. The expression is valid if and only if  is valid. Let $R[L[v], [\![t]\!]]$ denote the range $\{L[v], \cdots, L[v] + [\![t]\!]\}$. We define the following rules.

- Array declaration  Type $a[t_c]$

$$a \to R[L[a], [\![t_c]\!]]$$

  Type is data type, $t_c$ is constant expression, the rang of array index is $\{0, \cdots, [\![t_c]\!] - 1\}$, starting from $L[a]$.

- Array index  $a[t]$

$$0 \le t < [\![t_c]\!]$$

  Array index should be in the range of the definition.

- Dynamically allocated memory

$$p_{mal} = (\text{Type} *)\text{malloc}(t_p * \text{sizeof(Type)})$$

$$p_{mal} \to R[L[p_{mal}], [\![t_p]\!]]$$

  The range of $p_{mal}$ is $\{0, \cdots, [\![t_p]\!] - 1\}$, starting from $L[p_{mal}]$.

- Index  $p_{mal}[t]$  $*(p_{mal} + t)$

$$0 \le t < [\![t_p]\!]$$

- Pointer assignment  $q = \&p_{mal}[t]$

$$q \hookrightarrow R[L[p_{mal}], [\![t_p]\!]]$$

  Pointer $q$ points to the memory block $p_{mal}$.

- Memory freed  $\text{free}(p_{mal})$

$$\left(\forall s\left((s \hookrightarrow R[L[p_{mal}], [\![t_p]\!]]) \to \text{s} = [\![NULL]\!]\right)\right) \wedge p_{mal} = [\![NULL]\!]$$

  If $p_{mal}$ is freed, then all the pointers point to this memory block should be assigned to null, and so does $p_{mal}$.

- for loop  $\text{for}(\text{for}_{con})\{\text{for}_{body}\}$

  Firstly, get the bound of expression in $\text{for}_{con}$

$$\forall \text{identifier } i \in \text{for}_{con}, i_{min} \le i < i_{max}$$

  Secondly, all the index expression about *i* should be valid

$$\forall p[t_i] \in \text{for}_{body}, i_{min} \le i < i_{max} \to 0 \le t_i < [\![t_p]\!]$$

We can not always have the min and max value of expression in $\text{for}_{con}$, due to flexibility of program. However, our *for loop* rule can check a part of index reference.

# 4. Dangling Pointer Algorithm

The essence of dangling pointer is different live-time of its l-value and r-value. That is, the object of its r-value is freed or recycled earlier than the pointer itself, which makes the pointer points to invalid memory.

According to *pointer assignment* and *memory freed* rules, we can check dangling pointers after dynamically allocated memory is freed.

Let *Value(p)* expresses the value of pointer *p*.

**Algorithm**

1. Assume that there are *n* objects $p_1, p_2, \ldots, p_n$ dynamically allocated on the heap. The corresponding memory blocks are $M_1, M_2, \ldots, M_n$ . Then the used memory on the heap can be expressed as $M_H = M_1 \cup M_2 \cup \ldots \cup M_n$.
2. Build pointer tree $T_i$ for each $M_i$, and record all the pointers point to $M_i$. Then the pointer forest of the heap is $T_H = T_1 \cup T_2 \cup \ldots \cup T_n$.
3. Add pointers to or delete pointers from the corresponding pointer tree.
4. All the pointers on the pointer tree $T_j$ are dangling pointers after *free($M_j$)*.

**Adding property** For all $i = 1, \ldots, n$,

$$\forall p(\text{Value}(p) \in M_i \leftrightarrow p \in T_i)$$

That is, pointer $p$ is on the pointer tree $T_i$ of memory block $M_i$ if and only if it points to $M_i$.

**Deleting property** For all $i = 1, \ldots, n$,

$$\forall p(\text{Value}(p) \notin M_i \leftrightarrow p \notin T_i)$$

That is, if pointer $p$ does not point to $M_i$, then it should be deleted from $T_i$.

**Freeing property** For all $i = 1, \ldots, n$,

$$\forall i(\text{free}(p_i) \rightarrow \forall p(p \in T_i \rightarrow p = \text{NULL}))$$

That is, if memory $M_i$ which $p_i$ points is freed, all the pointers on the pointer tree $T_i$ should be assigned to null.

## 5. Theorem Proving Module

We add "pointer checker" into C compiler UCC to check dangling pointers and array index references. "Pointer checker" locates after "Syntax Parser".

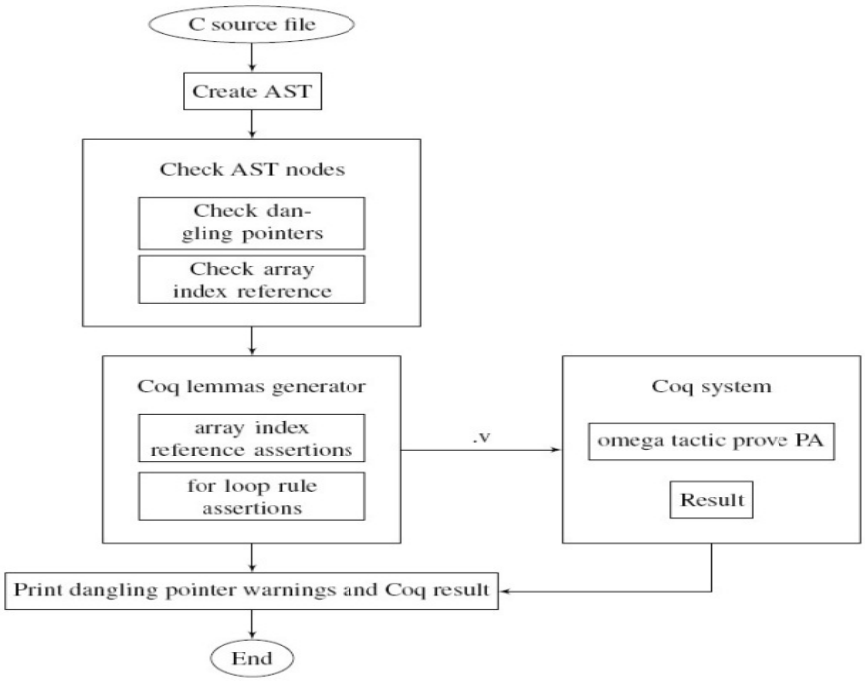


Fig. 1: Overall design of the module

*Array index* and *for loop* rules can be proved by Coq system. Our algorithm generates lemmas which would be proved by Coq. In 1929, M. Presburger proved that $\mathfrak{N}_A = \{\mathbb{N}; 0, S, <, +\}$ was decidable[3]. Pierre Crégut in CNET-Lannion developed *omega* tactic for *Presburger Algorithm* in Coq[4].

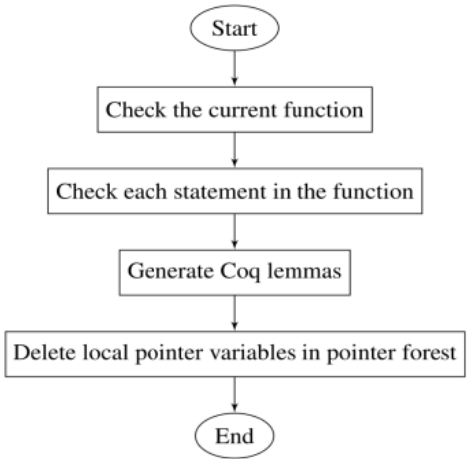


Fig. 2: MyCheckFunction flow chat

## 6. Experiments and Summary

We test several programs by our pointer checker. The result shows that we can find dangling pointers and array index out of bound.

Table. 3: Experiments

| Program | Lines | Lemmas | Dangling pointer | Time(s) |
|---|---|---|---|---|
| bubblesort.c | 21 | 7 | 0 | 6.382 |
| insertsort.c | 20 | 6 | 0 | 5.532 |
| selectsort.c | 22 | 7 | 0 | 5.312 |
| shellsort.c | 21 | 8 | 0 | 5.912 |
| malloc.c | 13 | 0 | 1 | 0.449 |
| malloc-two.c | 19 | 0 | 2 | 0.450 |
| malloc-multi.c | 20 | 0 | 3 | 0.451 |
| malloc-change.c | 21 | 0 | 1 | 0.450 |
| malloc-null.c | 24 | 0 | 0 | 0.430 |

The compiler with pointer checker would spend much time to prove lemmas in Coq system. However, our compiler can check dangling pointers and array index out of bound, which Visual C++ compiler would ignore. Other checkers, such as TVLA [5] and Cling [6], avoid reference errors by memory management.

We focus on the problems of dangling pointer and array index out of bound in C language. We analyse the reasons for these errors, and give formal definitions of pointers, and then design an algorithm to detect dangling pointers on heap. We add static analysis module in compiler UCC. We use automated theorem prover Coq to prove properties of program.

Our research is a simple application of automated theorem prover. In fact, automated theorem prover is much more powerful, which is very useful in the proving of program properties. We check a part of dangling pointers and array out of bound. We should increase the range of our checking and generate more kinds of logic rules from programs. How to use automated theorem prover to verify programs should be studied further.

## 7. References


[1] F. Marinescu; A. Corry. Historically bad ideas, Presentation: “Null References: The Billion Dollar Mistake”. http://qconlondon.com/london-2009/presentation/ Null+References: +The+Billion+Dollar+Mistake.

[2] D. Kroening and Ofer Strichmann. *Decision Procedures, an Algorithm Point of View*. Springer, Berling Heidelberg, 2008.

[3] M. Presburger. Über die vollständigkeit eines gewissen systems der arimethik ganzer zahlen, in welchem die addition als einzige operation hervortritt. Warsawa: *In Comptes Rendus du premier Congrés des Mathématiciens des Pays slaves*. 1929, PP: 92-101.

[4] Y. Bertot, P. Casteran. *Interactive Theorem Proving and Program Development, Coq’Art: the Calculu of Inductive Constructions*. Springer-Verlag, 2004.

[5] Tal Lev-Ami and Mooly Sagiv. TVLA: A system for implementing static analyses. *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2000, 1824: 105-110.

[6] Periklis Akritidis. Cling: A memory allocator to mitigate dangling pointers. *In Proceedings of the 19th USENIX conference on Security (USENIX Security’10)*, USENIX Association, Berkeley, CA, USA, 2010: 12-12.