基于位运算的固件代码变量区间分析法

谢晓东 李清宝 王 炜 牛小鹏 赵 远
(信息工程大学信息工程学院计算机系 郑州 450002)

摘 要 变量区间分析是程序代码数据流分析的重要部分。固件反汇编代码存在字节运算和位运算两类运算,当变量取值范围较大时,依次迭代法不能高效地分析经位运算后的结果的取值区间。提出一种快速位运算方法,即根据变量取值区间范围,先将变量转化为不确定位形式再进行位运算;提出一种区间生成算法,即把不确定位形式转化为区间来实现字节运算。实验结果表明,该基于位运算的固件代码变量区间分析方法在程序变量取值范围较大时效率远远高于依次迭代法,而且在各种变量取值区间范围内所需时间稳定,并随着区间范围的增大,其有略微下降趋势。

关键词 固件代码,区间分析,不确定位,位运算
中图法分类号 TP309.5 **文献标识码** A

Variable Intervals Analysis of Firmware Code Based on Binary-bit Operation

XIE Xiao-dong LI Qing-bao WANG Wei NIU Xiao-peng ZHAO Yuan
(Department of Computer Science and Technology, Institute of Information Engineering, PLA Information Engineering University, Zhengzhou 450002, China)

Abstract The variable intervals analysis plays an important role in program data-flow analysis. There are two different operations, word-level and bit-level. For the traditional iterative algorithm is inefficient to analyze the result intervals of bitwise operations if the variable has a large range, a quick bitwise operation method was proposed, which turns variables into the uncertainty bit form, and then makes the bitwise operators. When the uncertainty bit form of a variable needs to do word-level operation, the interval generated algorithm proposed can convert the form to the intervals. The experimental results show that the proposed method is time-consuming stability and more efficient than iterative algorithm with variable range large, and has a downward trend when the variable range expands.

Keywords Firmware code, Interval analysis, Uncertainty bit, Bit operation

1 引言

固件往往是整个电子系统中的控制核心,协调和控制系统的各个部分正常有序地运行,故固件的安全问题至关重要。固件代码安全分析时,程序变量可能取值的分析是程序数据流分析的重要目标之一,对发现程序中潜在的问题,如程序输出是否正确、代码逻辑是否完备等都有重要的指导意义。

对固件反汇编代码进行数据流分析,必须实现代码中的各种运算,包括字节级的标量运算(如加、减、乘、除等基本数学运算)和位级别的向量运算(如按位与、按位或、按位异或、移位等位逻辑运算)这两种不同性质的运算。在固件代码,按位的逻辑运算使用得非常频繁,但除了异或运算,其他运算都不存在逆运算,这就使得许多数据流方程不可解。若程序变量的可能取值使用区间表示,如何分析变量参与位运算后的结果的可能取值范围,成为亟待解决的问题。

使用布尔公式来抽象变量区间法[1]、代码切片法[2]及文献[3-5]中所提的几种抽象方法都可对变量范围进行分析,但上述文献所提方法没有考虑固件中字节级与位级两种性质的运算,因此在固件反汇编代码中无法直接应用。文献[6]中使用了另一种抽象解释技术将这两种运算结合在一起,在抽象空间中对指令语义进行解释,进而达到区间分析的目的。这种方法虽然可靠,但比较复杂,区间精度也有所损失。另外一种精确的方法就是依次迭代法,即对变量的每个可能取值依次做相应运算来得到结果,但这种方法在变量取值区间较大时,效率会明显下降。

本文针对固件中存在按字节和按位两种不同性质运算的特点,提出了一种精确地对变量区间进行分析的快速位运算方法。该方法通过变量可能取值区间的上下限计算出变量的不确定位形式,再进行位运算,之后若要进行字节级的运算,可通过区间生成算法把结果的不确定位形式转化为结果的可能取值区间。该方法实现简单,而且在变量取值范围较大时,其分析效率明显优于依次迭代法。

2 快速位运算方法

在固件代码中,系统的绝大多数变量的初始化信息都会存放在代码中某个位置。在对程序进行静态分析时,这些变

到稿日期:2012-03-02 返修日期:2012-06-09 本文受国家高技术研究发展计划(863,2009AA01Z434)资助。
谢晓东(1987-),男,硕士生,主要研究方向为信息安全,E-mail:xdicac226@163.com;李清宝(1967-),男,教授,博士生导师,主要研究方向为信息安全、可信计算;王 炜(1975-),男,博士,讲师,主要研究方向为信息安全、计算机系统结构;牛小鹏(1983-),男,博士生,主要研究方向为信息安全;赵 远(1984-),男,硕士生,主要研究方向为信息安全。

量的值就可直接由这些数据计算得到，即这些变量的值是确定的。但程序的输入变量不确定，分析人员最多只能确定这些输入变量的一个大致范围，或者可能的取值集合。当取值确定的变量同取值不确定的变量进行运算时，就有可能转化为取值不确定的变量。即程序变量的取值存在两种状态：获得定值和未获得定值。

定义 1 一个程序的控制流图(Control Flow Graph, CFG)是个三元组$\langle L,E,l_0\rangle$，其中 x,L 为顶点集，E 为连接了两个顶点的边组成的集合，$l_0\in L$ 为起始顶点。

在 CFG 中，顶点 $l\in L$ 可以是一条指令的标识，如指令地址，也可以是由指令组成的指令序列(如基本块、函数等)的标识，如指令序列中第一条指令的地址。总之，l 是它们在程序中唯一的标识。每个顶点与其直接后继相连，形成边。

程序中的每个变量都有一个地址，是变量的唯一标识。本文使用不同的符号来表示这些地址，那么这些符号就可以唯一标识一个变量，该符号称为变量名。所有变量名构成一个变量表，也称为符号表，记为 V。在这些变量中，把标识引脚的变量称为输入/输出变量。用符号 $val(x)$ 表示程序变量 x 的取值。

定义 2 设一个程序的 CFG 为$\langle L,E,l_0\rangle$，变量集为 V，如果存在一个 $l\in L$，使得某个变量 $x\in V$ 在此时可以通过程序上下文和变量来计算或者确定出其唯一的具体的数值，那么称变量 x 在 l 处有定值(或取得定值)。否则，称变量 x 在 l 处没有定值(或没有取得定值)

如果变量 x 在 l 处获得定值，那么符号 $val_l(x)$ 表示其值，否则，$val_l(x)$ 表示变量 x 的可能取值集合中的元素。在不引起歧义的情况下，l 可以省略。

图 1 所示的是某 8051 代码片段。

```
1        mov a,#50H
2        add a,r0
3        mov r0,a
4 target:mov b,#10H
5        nop
6        mov p0,#FFH
7        cjmp a,b,target
8        add a,b
```

图 1 示例代码

在第 1 行处，a 获得定值 50H，假设 r0 是不确定的，那么在第 2 行，由于和 r0 做加法运算，a 没有获得定值。在第 7 行，根据 CJMP 指令的语义，不论程序跳到哪个分支，标志位 CY 都会获得定值：若跳到 target 处，a 和 b 不等，CY=1；否则，a 和 b 相等，CY=0。对一个没有取得定值的变量，一般使用区间表示其可能的取值范围。当然对取值不连续的变量，可使用列举法表示该变量的可能取值的集合，这种情况本文不作讨论。使用区间表示的优点在于，可以利用不等式的性质，很方便地计算参与加减乘除等数学运算后的结果。前文提到，固件代码中存在字节级的标量运算和位级别的向量运算。区间是适合于字节级别的运算，而对于按位逻辑运算，其优势也就不存在了。因为按位运算是向量运算，不分大小，所以也就没有不等式了。此时依次迭代方法仍然适用，但如果区间所含数值个数很多，这种方法就会很耗时。

设程序变量 x 的取值范围为$[a,b]$。在位运算中，每个变量都是一个位向量，这个位向量每个分量只能取 0 或者 1。设存储器使用 n 位(bit)存储一个程序变量，使用 n 维位向量 $bit(x)$ 表示程序变量 x 对应的位向量，$\mathbf{E}=(2^n,2^{n-1},\cdots,2,1)$，那么就有 $val(x)=bit(x)\ \mathbf{E}^{\mathrm{T}}$。使用 $bit(x)_i$ 表示 x 变量的第 i 位。使用 $pre(x,k)$ 表示 $bit(x)$ 的前 k 位组成的位向量。$pre(x,0)$ 为空位向量。所有空位向量都相等。

例如，程序变量 x 在 $val(x)=43$(十进制)时，用一个字节(8 位)存储。把 x 写成二进制形式 $val(x)=101011$，那么，$bit(x)=(0,0,1,0,1,0,1,1)$，$bit(x)_1=0$，$bit(x)_5=1$，$pre(x,1)=(0)$，$pre(x,5)=(0,0,1,0,1)$。

定义 3 如果当 x 依次取遍$[a,b]$中所有值时，$bit(x)$ 能够确定 $bit(x)_i$ 只能取 1 或者 0，那么就称 $bit(x)_i$ 是可确定的，第 i 位为可确定位；否则称 $bit(x)_i$ 是不可确定的，第 i 位为不可确定位。

定义 4 称位向量 $bit^*(x)$ 为变量 x 的不确定位形式，其中 $bit^*(x)$ 满足如下条件：

(1) $|bit^*(x)|=|bit(x)|$；

(2) $bit^*(x)_i=bit(x)_i$，如果 $bit(x)_i$ 是确定位；

(3) $bit^*(x)_i=2$，如果 $bit(x)_i$ 是不确定位。

其中，$bit^*(x)_i$ 表示 $bit^*(x)$ 的第 i 位，$|bit^*(x)|$ 和 $|bit(x)|$ 分别表示这两个向量的长度，$1\leqslant i\leqslant|bit^*(x)|$。

作为向量，变量的不确定位形式可以参与位逻辑运算，只是位逻辑运算的计算方法要做相应的修改。

有了这种变量的不确定位形式，当使用区间表示的变量需要参与位级别逻辑运算时，可将其转化为不确定位形式，然后进行位运算。之后若需进行字节级别的运算，则将该形式转化为区间。

3 快速位运算方法实现

3.1 区间表示的变量不确定位形式的计算

首先，证明如下定理。

定理 1 对任意两个整数 a 和 b，如果满足 $0\leqslant a<b$，那么必然存在一个整数 $j\geqslant 0$，使得 $pre(a,j)=pre(b,j)$，并且 $bit(a)_{j+1}=0$，$bit(b)_{j+1}=1$。

证明：因为 a 和 b 不相等，所以必然存在一个 $j\geqslant 0$，使得 $pre(a,j)=pre(b,j)$ 并且 $pre(a,j+1)\neq pre(b,j+1)$。而 $a<b$，所以 $bit(a)_{j+1}<bit(b)_{j+1}$。由于 $bit(a)_{j+1}$ 和 $bit(b)_{j+1}$ 只能取 1 和 0，因此 $bit(a)_{j+1}=0$，$bit(b)_{j+1}=1$。证毕。

本文称上面定理中的 j 为整数 a 和 b 的分界位，记为 $j=sep(a,b)$。其中 $j=sep(a,b)$ 就蕴含了约束 $0\leqslant a<b$。

对于区间$[a,b]$，设 $val(x)\in[a,b]$，如果 $k=sep(a,b)$，那么就有 $pre(x,k)=pre(a,k)=pre(b,k)$，即 x 的前 k 位由常数 a 或 b 确定，是确定位。而 k 位之后的位是由 x 的具体取值决定的，是不可确定位。例如，当 $val(x)=a$ 时，$bit(x)_j+1$ 可以取 0，当 $val(x)=b$ 时，$bit(x)_j+1$ 可以取 1。

定理 2 给定一个区间$[a,b]$，$0\leqslant a<b$，a 和 b 为整数。如果 $k=sep(a,b)$，那么对于程序变量 $x\in V$，$val(x)\in[a,b]$，其对应的位向量的 k 位之前(包括 k 位)的位都是确定位，之后的位全是不可确定位。

证明：先证明 $bit(x)$ 的 k 位之前(包括 k 位)的位全部可确定。

因为 $k=sep(a,b)$，由定理 1 可知，$pre(a,k)=pre(b,k)$。

又因为 $x\in[a,b]$，即 $a\leqslant val(x)\leqslant b$，则 $pre(x,k)=pre(a,k)=pre(b,k)$。显然，$pre(a,k)$是可确定的，所以，$bit(x)$的 k 位及其之前的位全是可确定的。

再证明 $bit(x)$的 k 位之后全是不确定位。

假设存在一个 $i>k$，$bit(x)_i$ 为确定位。首先必有 $i\neq k+1$。因为根据定理 1，当 $val(x)=a$ 时，$bit(x)_{k+1}=0$，当 $val(x)=b$ 时，$bit(x)_{k+1}=1$，即 $bit(x)$的第 $k+1$ 位是不确定的，所以 $i>k+1$。

在 $i>k+1$ 的情况下：

若 $bit(a)_i\neq bit(b)_i$，即 x 分别取 a 和 b 时，$bit(x)_i$ 不同，这与假设 $bit(x)_i$ 为确定位矛盾。

若 $bit(a)_i=bit(b)_i$，当 $bit(a)_i=bit(b)_i=0$ 时，$bit(x)_i=0$。构造整数 c，使得 $pre(c,k+1)=pre(a,k+1)$，其他位全部为 1。此时 $bit(c)_i=1$。显然 $c\geqslant a$，又由 $pre(c,k+1)=pre(a,k+1)$可知 $bit(c)_{k+1}=bit(a)_{k+1}=0<bit(b)_{k+1}=1$，故 $c<b$。即 $c\in[a,b]$，所以 x 可以取到 c，此时 $bit(x)_i=bit(c)_i=1$，这与之前 $bit(x)_i=0$ 矛盾。

当 $bit(a)_i=bit(b)_i=1$ 时，$bit(x)_i=1$。构造整数 c，$pre(c,k+1)=pre(b,k+1)$，其他位全部为 0。同理可证，$c\in[a,b]$。那么 x 可以取到 c，此时 $bit(x)_i=bit(c)_i=0$，与之前 $bit(x)_i=1$ 矛盾。

综上所述，假设不成立，原命题成立。证毕。

此时，把 k 称为变量 x 的分界位。进一步还可以得出如下结论。

推论 对于程序变量 $x\in V$，若 $val(x)\in[a,b]$，其中 $0\leqslant a<b$，a 和 b 为整数，那么当 x 取遍$[a,b]$中每个数时，其每个不确定位必然会取到 0 和 1。

证明略。

依据定理 2，可以把使用区间表示其可能取值范围的变量转化为适用于按位运算的不确定位形式。此步转化的重点是把数据转化为二进制表示时的位向量，即计算 $bit(x)$。计算 $bit(x)$的算法很多，本文对这步转化的算法不做规定，在实现时根据所使用的编程语言的特点，高效计算出结果即可。本文实验部分的不确定位形式计算算法是基于不同进制之间的转化关系的思想实现的。

3.2 位运算算法

含有区间表示的变量间的按位运算会有两种情况：取得定值的变量和没有取得定值的变量之间的运算，以及没有取得定值的变量之间的运算。

使用区间表示的变量的位运算计算方法的基本思想是首先计算使用区间表示的变量的分界位，然后分别对确定位和不确定位进行运算。

(1)未获得定值的变量和获得定值的变量进行位运算

设 $x\in V_l^*$，取值范围为$[a,b]$，与常整数 $d\geqslant 0$ 做按位逻辑运算，结果为 x'。

首先，计算 $k=sep(a,b)$，得到 x 的确定位和不确定位；

接着，如果 $bit(x)_i$ 是可确定的，$bit(x')_i=bit(x)_i\cdot bit(d)_i$，其中 $\cdot$ 表示所有的二元位运算；

如果 $bit(x)_i$ 是不可确定的，有如下几种和位操作相关的情况：

1)按位与(bitand)运算：当 $bit(d)_i$ 为 0 时，$bit(x')_i=0$，当 $bit(d)_i$ 为 1 时，$bit(x')_i$ 是不可确定的；

2)按位或(bitor)运算：当 $bit(d)_i$ 为 1 时，$bit(x')_i=1$，当 $bit(d)_i$ 为 0 时，$bit(x')_i$ 是不可确定的；

3)按位异或(bitxor)运算：$bit(x')_i$ 也是不可确定的。

4)对于移位和高低字节交换操作：设 $bit(x)_i$ 经过运算后对应 $bit(x')_{i'}$，$bit(x')_{i'}$ 也是不可确定的。

(2)两个未获得定值的变量之间的位运算

设 $x\in[a_1,a_2]$，$y\in[b_1,b_2]$，x 和 y 做按位逻辑运算，其结果为 z。

第 1 步 计算 $k_1=sep(a_1,a_2)$和 $k_2=sep(b_1,b_2)$；

第 2 步 取 $k=\max(k_1,k_2)$，那么 z 的 k 位之后的位全为不可确定位；

第 3 步 对于 z 的前 k 位(包括 k 位)，如果 $k=k_1$，那么把 $pre(x,k)$作为常数，$pre(y,k)$作为变量，按前面变量和常量的计算方法计算 z 的前 k 位；如果 $k=k_2$，那么把 $pre(y,k)$作为常数，$pre(x,k)$作为变量，按前面变量和常量的计算方法计算 z 的前 k 位。

对一元位运算，可使用上面的思想进行计算，此处不再赘述。至此，面向变量不确定位形式的位运算算法描述完毕。

3.3 区间生成算法

依据推论，当变量 x 取遍区间每个值时，其不确定位会遍取 1 和 0，在经上面位运算算法之后的结果中，还是不确定位的也必然会取到 1 和 0。如果对计算结果中的不确定位分别取 1 和 0，就会产生计算结果的可能取值集合。

设 $\boldsymbol{x}$ 是一个位向量，使用 $vari(\boldsymbol{x})$表示 $\boldsymbol{x}$ 的不确定位的下标区间的集合，$|\boldsymbol{x}|$为位向量的维数。例如，一个 8 维位向量 $\boldsymbol{x}=(0,1,0,0,0,0,0,0)$，其不确定位为 3、6、7 和 8，那么 $vari(\boldsymbol{x})=\{(3,3),(6,8)\}$，$|\boldsymbol{x}|=8$。

对于区间 g，使用 $low(g)$表示 g 的下限，$up(g)$表示 g 的上限，$|g|$表示区间 g 中元素的个数。

定义 5 若区间集 $vari(\boldsymbol{x})$中存在一个区间 g，使得 $up(g)=|\boldsymbol{x}|$，那么把属于 g 中的不确定位称为区间不确定位，简称区间位，其他不确定位称为非区间不确定位，简称非区间位，如图 2 所示。

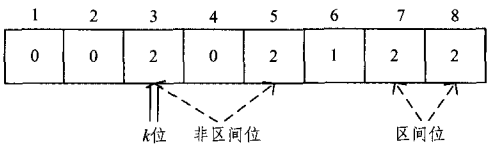


图 2 区间位和非区间位示意图

区间生成算法如图 3 所示。区间生成算法的思想是对每个非区间位取 0 和 1，生成新的值，再对新值中的非区间分别取 0 和 1，产生新值，如此循环，直到所有非区间位都变为确定位；对于区间位，全取 0 得到区间下限，全取 1 得到区间上限，即可得到该变量的可能取值范围。

区间生成算法

输入：x(x 的每个分量只能取 0,1,2,其中 2 表示此位是不确定位)
输出：集合 Rset(包含数字或区间)

```
begin
  queue. add(x)
  g= intervalbit (x)
  st =0
  if g is empty then
      en=len(x)
  else
      en=low(g)-1
```

```
endif
while st <= ke do //非区间位
    x= queue. remove()
    k=find (x,2,st,en)
    if k =-1 then
        queue. add(x)
        break
    endif
    x1=replace (x,k,0)
    queue. add(x1)
    x1=replace (x,k,1)
    queue. add(x1)
    st=k
endwhile
if g is empty then
    foreach t in queue do
        a=tE^T
        Rset. add(a)
    endforeach
else //区间位
    foreach t in queue do
        x1=replace(t,g,0)
        x2=replace(t,g,1)
        a=x1E^T
        b=x2E^T
        Rset. add((a,b))
    endforeach
endif
end
```

图 3　区间生成算法

其中，函数 *intervalbit*(***x***)用来计算区间位所属区间。*replace* (***x***,*g*,*t*)将向量 ***x*** 中 *g* 范围内的位或者某一位换成 *t*。*len*(*g*)返回区间 *g* 的长度。*queue* 是一个队列，有 *add* 和 *remove* 两种操作。函数 *find*(***x***, *v*, *start*, *end*)是在向量 ***x*** 的 *start* 到 *end* 范围中查找第一个值为 *v* 的元素下标。

从算法中可以看出，算法的输出是一个个独立的取值，即采用列举法表示的变量可能的取值。根据区间生成算法，非区间位的位数决定了区间生成算法生成区间和数值的个数。区间位位数则决定了每个区间的长度。

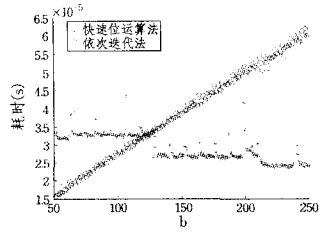
由前面讨论可知，变量的区间决定了变量的不确定位，区间长度决定了变量的不确定位的位数。经过位运算后，结果的不确定位的位数必然小于等于原变量的不确定位位数。区间生成算法对每个不确定位分别取 0 和 1，这样做必须有个前提，即当结果取遍其可能取值之后，它的所有不确定位能够取得 0 和 1 的所有排列组合。而要保证这一点，就要求参与运算的变量取值区间足够长。

具体地，当计算结果满足 $2^{(n-k+1)} \leqslant b-a+1$，$k=low(LOW(vari(bit(x))))$，$x \in [a,b]$时，区间生成算法成立。$n$ 为变量存储位数。k 的位置如图 2 所示。

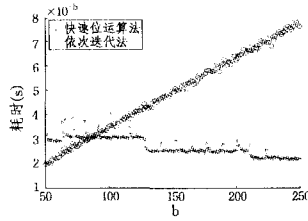
4　实验

本文所述算法已使用 python3. 2 实现。实验主要是对快速位运算算法进行测试，并与依次迭代法的效率进行对比。由快速位运算算法可知，该算法包含 3 个步骤，步骤 1 根据区间计算变量的不确定位形式，步骤 2 对不确定位形式进行位运算，步骤 3 将不确定位形式转化为区间。影响该算法效率的因素主要有：变量存储位数 n、程序变量的可能取值区间 $[a,b]$的长度，以及参与位运算的整数 p。

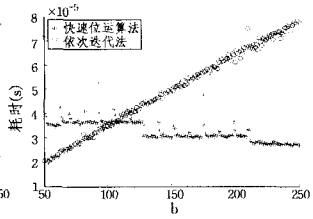
为了比较快速位运算算法和依次迭代算法的效率，实验取 $n=8$，对程序变量区间 $[a,b]$，$a=1$，b 从 50 增长到 250，p 分别取 6(结果只含有非区间不确定位)、7(结果只含有区间不确定位)、11(结果同时含有区间不确定位和非区间不确定位)进行按位与运算。图 4 显示不同情况下，快速位运算算法和依次迭代算法随 b 增长的耗时情况。



(*a*) p=6



(*b*) p=7



(*c*) p=11

图 4　不同情况快速位运算算法与依次迭代法的分析耗时 (n=8,a=1)

实验结果表明，当程序变量取值范围较小时，依次迭代法耗时小于快速位运算算法，但随着 b 的增长，快速位运算算法耗时都会小于依次迭代法。而且 p 取不同值，两个算法的交点也不一样。快速位运算法的步骤 2 位运算结果中只含有区间位($p=7$)时，步骤 3 就不需要对非区间位进行处理，只将不确定位分别设置为 0 和 1，因此使得整个算法耗时最小，与依次迭代法耗时曲线的交点自然靠前。当结果中同时含有区间位和非区间位($p=11$)时，由于对非区间位要将所有非区间位转化为确定位，依次算法耗时会大些。当结果中仅含有非区间位($p=6$)时，步骤 3 就只对所有非区间位进行处理。由于需要对每个非区间位依次进行 0 和 1 的赋值，因此对非区间位的处理是最耗时的。对于后两种情况，会因为非区间位的位数不同而导致消耗不同。由于 $p=11$ 时结果的非区间位位数小于 $p=6$ 时的位数，故与依次迭代法的耗时曲线的交点要小于 $p=6$ 时的交点，如图 4(a)和 4(c)所示。

实验结果还表明，随着区间长度的增长，每个图中快速位运算算法的耗时基本平稳，并有下降的趋势。影响该算法耗时的因素有 n、p 和区间长度，对于每个图而言，n 和 p 都已经确定。区间长度则通过影响变量的不确定位个数来影响算法的耗时，但不确定位的个数存在上界 n，所以快速位运算算法耗时始终保持在某种固定水平之下，不会增长。

对耗时有下降趋势的原因，需要从算法的 3 个步骤来分析。步骤 1 不确定位形式转化算法从位向量的低位开始，遇到不确定位立即停止。步骤 2 位运算也从位向量的低位开始，遇到确定位进行位运算，遇到不确定位则直接赋值，而赋值的耗时要小于位运算的耗时，所以影响前两步耗时与确定位的位数正相关。步骤 3 区间生成算法则是从位向量的高位

开始查找不确定位，其耗时与不确定位的位数正相关。当区间上限 b 增长到 64 和 128 时，由区间计算而得的不确定位位数增加，确定位个数减少，进而导致前两个步骤耗时减少，步骤 3 耗时增加，若前两步骤减少的时间大于步骤 3 增加的时间时，快速位运算算法耗时就会下降。

与之对应的是依次迭代法呈线性增长趋势。这是由依次迭代算法本身机制决定的。若设 python 中一次位运算所需时间为 t，那么区间增长 Δx 个数值，依次迭代法的耗时就增长 $t \cdot \Delta x$。

以上是 $n=8$ 时的测试结果，对 n 取别的数值如 16、32 等时进行测试，也有类似的结论，只是快速位运算算法的优势会更明显。图 5 即为 $n=16, a=1, b$ 从 2000 增长到 5000，$p=1895$(同时存在非区间位和区间位)时，两个算法的耗时对比。可以看出，快速位运算算法耗时要远远小于依次迭代法。

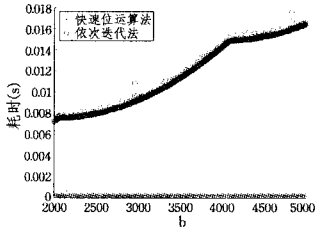
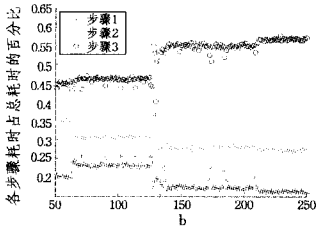


图 5　$n=16, p=1895$ 时快速位运算算法与依次迭代法的耗时

上述算法各步骤所耗时间各不相同，图 6 给出了 $n=8$，$p=6$时算法各步骤耗时占总耗时的百分比情况。



注：步骤 1 指变量不确定位形式的计算；步骤 2 指位运算；步骤 3 指区间生成算法。

图 6　$n=8, p=6$ 时快速位运算算法各步骤耗时百分比

分析结果表明，区间生成算法(步骤 3)占据整个快速位运算算法耗时中绝大部分。由算法可知，此步骤中时耗主要集中在对非区间位的处理上。对于连续多次位运算的情况，由于快速位运算算法中只有在变量使用不确定位形式并需要进行字节级运算时，才使用区间生成算法进行转化，因此快速位运算算法优势更加明显。

结束语　固件代码数据流分析时，正确高效地计算代码中字节级别与位级别的运算结果，是提高固件代码分析效率的重要方法之一。本文提出的针对使用区间表示的变量进行快速位运算的算法，在变量需要位运算的时候，将其转化为变量的不确定位形式后参与位运算，不确定位形式遇字节级运算时，使用区间生成算法将不确定位形式转化为区间。该算法在区间较大时的执行效率要高于依次迭代法。

参 考 文 献

[1] Brauer J, King A. Automatic Abstraction for Intervals using Boolean Formulae[C]//SAS, LNCS. Springer, 2010

[2] Bergeron J, Debbabi M, Erhioui M M, et al. Static Analysis of Binary Code to Isolate Malicious Behaviors[C]//IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. Palo Alto, California, IEEE Press, June 1999

[3] Sankaranarayanan S, Ivancic F, Gupta A. Program Analysis Using Symbolic Ranges[C]//SAS. 2007

[4] Cifuentes C, Fraboulet A. Interprocedural Data Flow Recovery of High-level Language Code from Assembly[R]. Technical Report 421. Department of Computer Science and Electrical Engineering, The University of Queensland, Dec. 1997

[5] Kinder J, Zuleger F, Veith H. An Interpretation-Based Framework for Control Flow Reconstruction from Binaries[C]//VMCAI 2009. 2009: 214-228

[6] Brauer J, Noll T, Schlich B. Interval Analysis of Microcontroller Code using Interpretation of Hardware and Software[C]//Proc. Proceedings of the 13th International Workshop on Software and Compilers for Embedded Systems (SCOPES 2010). ACM, 2010

(上接第 93 页)

线放置角度、收发天线间距、巷道长度、莱斯因子、角度扩展和平均达到角等对空间相关函数的具体影响。并且对仿真结果进行了具体分析。仿真结果表明，巷道环境中天线放置角度对相关系数的影响较大；天线间距取较大值时，才能得到较小的相关性；相关系数还要受到天线间距和巷道长度以及角度扩展等的影响，比地面的相关性更大。因此，在将 MIMO 系统应用于煤矿井巷中时，链路之间的空时相关特性不可忽略，但是在考虑相关性影响的同时，可以尽量减小系统的相关性。

参 考 文 献

[1] Chuang J, Sollenberger N. Beyond 3G wideband wireless data access based on OFDM and dynamic packet assignment [J]. IEEE Commun. Mag., 2000, 8(1): 735-749

[2] Bölcskei H, Paulraj A J, et al. Fixed Broadband Wireless Access: State of the Art, Challenges, and Future Directions [J]. IEEE Commun. Mag., 2001, 6(2): 905-917

[3] 杨正光，等. SC-FDE/mimo 技术在宽带无线接入系统中的应用研究[J]. 南京邮电学院学报，2004，9(1)：11-13

[4] 朱近康. 无线信道的应用模型和估计[J]. 中兴通信技术期刊，2003(s1)：33-35

[5] Raleigh G G, Cioffi J M. Spatio-temporal coding for wireless communi- cations [J]. Proc. IEEE 1996 Global Communications Conf., 1996, 6(1): 1809-1814

[6] 刘刚，郭漪，葛建华. MIMO-OFDM 系统中的信道估计[J]. 华中科技大学学报：自然科学版，2005，9(1)：26-29

[7] Hochwald B H B M. How Much Training is Needed in Multiple-Antenna Wireless Links[J]. 电子学报，2001，7(1)：16-22

[8] 陈可，李野. EEMD 分解在电力系统故障信号检测中的应用[J]. 计算机仿真，2010(03)

[9] 孙增友，车成华，赵涛. 基于认知无线电的超宽带信号频谱检测的研究[J]. 重庆邮电大学学报：自然科学版，2010，22(4)：426-430