计算机科学2001Vol. 28№. 10

按位异或及其在求解游戏策略问题中的应用

The Bitwise Exclusive OR Operator and its Application to Strategy for Playing

李学武

(天津师范大学计算机科学系　天津300074)

Abstract　The paper presents some properties on bitwise exclusive OR operator, especially, theorem 1 and theorem 2 are two important results. And also it presents its application to strategy for playing about balance status

Keywords　Exclusive OR, Bitwise exclusive OR, Strategy for playing, Balance status

1　按位异或的若干性质

约定　在下文中，N^*均表示非负十进制数的集合

定义1(a⊙b)　设a,b∈{0,1}，两个一位二进制数a,b的异或a⊙b的真值表如下：

a	b	a⊙b
1	1	0
1	0	1
0	1	1
0	0	0

定义2(x xor y)　x,y∈N^*，两个非负十进制数x,y的按位异或x xor y是对相应的二进制数按位进行⊙运算，其结果仍转换为十进制数。如果相应的二进制位数不等，较小的数前面的空白按零处理。

一些高级计算机语言，如C，(扩展)PASCAL等都支持对十进制整数的xor运算[1]。

⊙运算的性质：

性质1(交换律)　若a,b∈{0,1}，则a⊙b=b⊙a。

性质2(结合律)　若a,b,c∈{0,1}，则(a⊙b)⊙c=a⊙(b⊙c)。

性质1、2容易利用定义1中的真值表验证。

⊙xor运算的性质

在下文中，x,y,z∈N^*，$x_1,x_2,\cdots,x_n\in N^*$。

性质3　x xor 0=x。

性质4　x xor y=0的充分必要条件是x=y。

利用定义2容易证明性质3、4。

性质5(交换律)　x xor y=y xoy x

证明：设$x=(b_t\cdots b_1b_0)_2, b_i\in\{0,1\}, i=0,1,\cdots,t, b_t=1$；$y=(c_r\cdots c_1c_0)_2, c_i\in\{0,1\}, i=0,1,\cdots,r, c_r=1$，

不妨设$t\geqslant r$，可在c_r之前补$r-t$个零，即$c_t=\cdots=c_{r+1}=0$，由性质1，$b_i\odot c_i=c_i\odot b_i, i=1,\cdots,t$，所以性质5成立。

性质6(结合律)　(x xor y)xor z=x xor(y xor z)

证明与性质5类似(利用性质2)

性质7(增广律)　若x=y，则x xor z=y xor z。

性质7可利用定义1、2直接导出。

性质8　若$x<2^t$，则x xor $2^t=x+2^t$

性质9　设$x=(b_t\cdots b_1b_0)_2, b_i\in\{0,1\}, i=0,1,\cdots,t, b_t=1$；则x xor $2^t=x-2^t$

性质8、9可利用定义1、2直接导出。

性质10　若$x<2^t$，且$y<2^t$，则(x xor y)$+2^t$=x xor (y-2^t)。

由已知及定义，易知x xor y$<2^t$，由性质8，(x xor y)-2^t=x xor y xor 2^t=x xor (y xor 2^t)=x xor (y$+2^t$)，故性质10成立。

性质11　x与偶数个y的异或仍为x，即：x xor y xor y xor$\cdots$xor y=x(共偶数个y)。

这是性质3、4的直接推论。

定义3(T)　设$x_1,x_2,\cdots,x_n\in N^*$，则：$T=x_1$ xor x_2 xor $\cdots$ xor x_n。

定义4(T_i)　设$x_1,x_2,\cdots,x_n\in N^*$，则$T_i=T$ xor $x_i, i=1,2,\cdots,n$。

由性质3.4，T_i实质上就是对$\{x_1,x_2,\cdots,x_{i-1},x_{i+1},\cdots,x_n\}$中的n-1个数求异或。

定义5(取数操作)　在n个非负整数序列$x_1,x_2,\cdots,x_n$中，选取一个x_i及正整数b，满足$x_i\geqslant b>0$，再用$y_i=x_i-b$取代序列中的x_i，称为对序列$x_1,x_2,\cdots,x_n$的一个取数操作。

定理1　给定N^*中的序列$x_1,x_2,\cdots,x_n(n\geqslant 2)$，若$T\neq 0$，必存在一个取数操作，使$y_i$取代$x_i$后，$T'=x_1$ xor$\cdots x_{i-1}$ xor y_i xor x_{i+1} xor$\cdots x_n=0$。

证明：取$T_i=T$ xor x_i，如果存在某个$x_i>T_i$令$b=x_i-T_i>0, y_i=x_i-b=T_i$，由性质3、4，知，用$y_i$代替

序列 $x_1\cdots x_n$ 中的 x_i，即 $T'=(T \text{ xor } x_i)\text{xor } y_i=T_i \text{ xor } T_i=0$。

下面证明必存在某个 $i\in\{1,2,\cdots,n\}$，使 $x_i>T_i$。

设序列 $x_1\cdots x_n$ 对应二进制数序列中，最大数的最高非零位为 t+1位，最高位的值对应于 2^t，设这样的数有 r 个($n\geq r\geq 1$)，记相应的集合为 A^t，分两种情况讨论：

(1)r 是奇数。不妨设 $x_1\in A^t$，即 $x_1\geq 2^t$。A^t 中还有偶数个具有 t+1位的数，其中任何两个数的异或均 $<2^t$，利用性质3、4可知 $T_1=T \text{ xor } x_1<2^t$，$\therefore x_1>T_1$，结论正确。

(2)r 是偶数。不妨设 $x_1\cdots x_r\in A^t$，其余 x_i 均小于 2^t，在 T 中添加 r(偶数)个 2^t：

$$T=x_1 \text{ xor } x_2\cdots\text{xor } x_r \text{ xor}\cdots\text{xor } x_n$$
$$=(x_1 \text{ xor } 2^t) \text{ xor } \cdots \text{ xor } (x_r \text{ xor } 2^t) \text{ xor } \cdots \text{ xor } x_n$$

(根据性质11)

$$=y_1 \text{ xor } y_2\cdots \text{ xor } y_r \text{ xor } x_{r+1}\text{xor } \cdots \text{ xor } x_n$$

根据性质9，$y_i=x_i-2^t$，$i=1,\cdots,r$，序列 $y_1,y_2,\cdots,y_r,x_{r+1}\cdots x_n$ 对应的二进制数序列中的最大数最高非零位为 s+1位，显然 $s<t$，设最高位对应于 2^s 的数有 p 个，若 p 是奇数，则已；若是偶数，则重复上述操作，直到某一步，具有相同最高位的数的个数为奇数个为止，我们说这一步必然能达到，否则，若具有相同最高位的数的个数永远为偶数，其结果必导致 T=0，与题设矛盾。

将上述操作的最后的结果记为 $y_1\cdots y_p,y_{p+1}\cdots y_n$，由于每次只增加了偶数个 2^t 的异或，由性质11，T 的值不变，其中 $y_1\cdots y_p$ 各数对应的二进制数的最高位对应于 2^t，其余 y_i 均小于 2^t，且 p 是奇数。$T=y_1 \text{ xor}\cdots\text{xor } y_p \text{ xor } y_{p+1}\cdots\text{xor } y_n$

下面证明 $x_1>T_1$

易知：$x_1=y_1+2^{t1}+\cdots+2^{tu}$，其中，$t1>t2>\cdots>tu$，即 y_1 是由 x_1 依次经过对 $2^{t1},\cdots,2^{tu}$ 的异或得到的(参看性质8、9)。

利用前面(1)的分析，易知 $y_1>T \text{ xor } y_1$，显然 $T \text{ xor } y_i<2^n$，$i=1,\cdots,u$；$\therefore y_1-2^{t1}+\cdots+2^{tu}>(T \text{ xor } y_1)+2^{t1}+\cdots+2^{tu}$，左边即为 x_1，右边多次利用性质10，可得：

$(T \text{ xor } y_1)+2^{t1}+\cdots-2^{t1}=T \text{ xor } (y1+2^{t1}+\cdots+2^{tu})=T \text{ xor } x_1=T_1$，即 $x_1>T_1$，证毕。

定理2 设 T=0，对序列 $x_1\cdots x_n$ 作任一次取数操作，即取 x_i 及正数 b，满足 $x_i\geq b>0$，然后用 $y_i=x_i-b$ 代替 $x_i(x_i>y_i\geq 0)$ 后，记 $T'=x_1 \text{ xor}\cdots x_{i-1} \text{ xor } y_i \text{ xor } x_{i+1}\cdots\text{xor } x_n$，则必有 $T'\neq 0$。

证明：令 $y_i=x_i-b$，$x_i\geq b>0$，$y_i\geq 0$，$T'=T \text{ xor } y_i \text{ xor } x_i$(即从 T 中去掉 x_i，加上 y_i)$=y_i \text{ xor } x_i=y_i \text{ xor } (y_i+b)$(注意 T=0)，如果 T'=0，由性质4，必有 $y_i=x_i$ 从而导致 b=0，与 b>0矛盾，证毕。

如果将 T=0的情形定义为某系统处于平衡态，$T\neq 0$的情况定义为处于非平衡态，定理1、定理2表明：当系统处于非平衡态时，至少存在一种取数操作，将其变为平衡态；当系统处于平衡态时，任何一个取数操作都将导致非平衡态。

2 一类游戏策略问题及解法

2.1 取石子游戏1

任给 N 堆石子，堆数 N 及每堆石子数由游戏者给出，两人(游戏者与计算机)轮流从任一堆中任取(每次只能取自一堆)，计算机先取，取最后一颗石子的一方获胜。试设计一种方法，使计算机有较多的获胜的机会。

分析：记第 i 堆石子数为 x_i，$T=x_1 \text{ xor } x_2\cdots\text{xor } x_r \text{ xor}\cdots\text{xor } x_n$，显然，有必胜策略的一方应在取数操作之前面临非平衡态，操作后留给对方平衡态。根据定理1与定理2，如果初始状态时 T=0，计算机一方没有必胜策略，可在最多的一堆中取一颗，寄获胜希望于对方的失误。如果初始状态时 $T\neq 0$，由定理1，必存在某个 $x_i>T_i$，可在第 i 堆中取 x_i-T_i 颗，使对方永远处于 T=0的平衡状态，计算机必胜。

C 语言程序如下：

```
#include(stdio.h)
unsigned int a[11],n;
void init()
 {int i;
  printf("input n(2—10) ");scanf("%u",&n);
  for(i=1;i<=n;i++)
    {printf("input No.%d Number of stone:\n",i);
     scanf("%d",&a[i]);}
 }
void status()
 { int i;
   printf("Now remainder:\n");
   for (i=1;i<=n;i++) printf(" No %d rem. %u \n",i,a
      [i]);
 }
unsigned int suml()
 { unsigned int s;int i;
   for(s=0,i=1;i<=n;i++) s+=a[i];
   return s;
 }
unsigned int xorall()
 { unsigned int s;int i;
   for(s=0,i=1;i<=n;i++) s^=a[i];
   return s;
 }
main()
  { unsigned int f;int i,s,e;
    init();
    while (suml())
     {if (xorall()==0)
       {for (i=1;i<=;i++)
        if (a[i]>0)
          {printf("computer take 1 from No. %d /n",i);
          a[i]--;break;
          }
       }
```

(下转第125页)

概念类的依赖度关系　分为两个方面，一个是依赖度，另一个是被依赖度。其中概念类之间的依赖度指的是某个类所依赖的类的数目与类的总数的比值，其计算公式如下，令类的总数为 C，类 i 所依赖类的数目为 D，依赖度为：

依赖度＝D/C

而被依赖度指的是依赖于某个类的类的个数与类的总数的比值。令类的总数为 C，依赖于类 i 的类的个数为 D'，被依赖度为：

被依赖度＝D'/C

依赖度反映的是该类的稳定性，一个类对其它的类的依赖程度越大，这个类越不稳定，这个类就需要更认真的规划设计；而被依赖度反映的是一个类的重要性，一个类越是被依赖的程度越深，则这个类设计的优劣对系统的影响越大。依赖度和被依赖度可以为设计阶段类的设计提供参考依据。

3.4　状态图度量

状态图是对状态变化较复杂的类或对象进行补充说明的一种图，对于它主要是度量出该类的复杂性，以使开发人员对这个类事先有充分的估计。状态图中主要是状态和转移两种结点，故对于状态图的度量，只需简单地计算状态的个数。

结束语　一个好的度量方案不仅要能度量出一些关键的指标，而且要尽可能再现一个系统所有可能有用的特征，这好比我们认识事物，只有把事物的尽可能多的特性了解清楚，才能更真实地把握事物的运动规律，才能以不变应万变。针对需求分析模型的度量，是软件度量的一个重要部分，它可以使得在项目还没有设计之前对项目情况有一个较好的了解。由于目前 UML 在市场上的主导地位以及其本身信息的完善性、规范性等优点，使得基于 UML 的软件度量研究成为当务之急，本文正是从这点出发，提出了自己的度量方案。

参 考 文 献

1　潘飚. C++语言面向对象的软件度量学研究.[硕士学位论文] 合肥工业大学，1999.5

2　Albrecht A J. Measuring Application Development Productivity In. Proc. IBM Application Development Symposium，Monterey，CA，Oct 1979. 83～92

3　DeMarco T. Controlling Software Projects Yourdon Press，1982

4　Jones C Applied Software Measurement. McGraw-Hill，1986

5　Whitmre S A. An Introduction to 3D Function Points Software Development，April 1995. 43～53

6　李心科，刘宗田，等. 一个面向对象软件度量工具的实现和度量实验研究. 计算机学报，2000(11)

7　Chidamber R，Kemerer F. A Metrics Suite for Object-Oriented Design. IEEE Trans，Softw.，Eng ，1994，20(6)

(上接第127页)

```
else
 for (i=1;i<=n;i++)
   {s=a[i]-(xorall() ^ a[i]);
    if (s>0)
      {printf("computer take %u from No. %d \n",s,i);
      a[i] ^ =xorall();
      break;
      }
 }
  if(sum()==0)
     {printf("computer win!");break;}
  status();
  while(1)
     {printf("Input your selection\n(exp 1 2 means take 2
        from No. 1);\n");
      scanf("%d %u",&e,&t);
      if ((e>=1)&&(e<=n)&&(a[e]>=t))
        a[e]-=t;break;}
      else
        printf("data error! re-input…\n");
      }
if(sum()==0)
{printf("you win!");break;}
}
}
```

2.2　取石子游戏2

任给 N 堆石子，堆数 N 及每堆石子数由游戏者给出，两人(游戏者与计算机)轮流从任一堆中任取(每次只能取自一堆)，规定每方每次最多取 K 颗，计算机先取，取最后一颗石子的一方获胜。试设计一种方法，使计算机有较多的获胜的机会。

分析：记第 i 堆石子数为 x_i，令 $y_i=x_i \bmod (K+1)$，$i=1,\cdots,n$，定义 $T=y_1 \text{ xor } y_2 \cdots \text{xor } y_i \text{ xor } \cdots \text{ xor } y_n$。如果初始状态时 T＝0，则为平衡态，先走的计算机一方没有获胜策略，如果初始状态时 T≠0，由定理1，必存在某个 $y_i>T_i$，可在第 i 堆中取 y_i-T_i 颗，使对方处于 T＝0的平衡状态，在以后的各轮中，对方在第 i 堆中取 r 颗，若 $x_i>k$，计算机就在同一堆中取 k＋1－r 颗，这时 y_i 不变。若 $x_i \leqslant k$，则重新计算 y_i 及 T，按问题2.1的方法选取，这时，计算机有必胜策略。具体实现细节不再赘述。

对上述算法略做修改，便可解决问题2.1、2.2的对偶问题：取最后一颗石子的一方为失败。

参 考 文 献

1　Kernighan B W，Ritchie D M. The C Programming Language(2nd). Prentice Hall，1988 48，207