面向过程程序 变量 封装 (15)

计算机科学1993Vol.20№.4

73-77

面向过程程序中变量的封装

TP31

陈学锋 王振宇 郭福顺

(哈尔滨工业大学计算机系，哈尔滨150006)

摘 要

We apply program decomposition slicing to the encapsulation of global variables in procedure-oriented programs by constructing the modificaion functions and using functions of the variables from the decompositon slices. This gives programmers a staightforward technique for reforming the structure of existing programs and makes it easier to understand or change the programs.

一、引 言

用面向过程语言(如Pascal，C等)编写的软件从整体结构上看可以分成三种相互独立的语言成份：过程、变量和类型。由于三者之间缺乏联系和约束，使得无论何种类型的变量（全局的）都可被每个（无论何类型的）过程存取。因此软件在整体结构上层次不清，关系过于繁杂。当软件规模大到一定程度，如10万行的C程序[1]时，软件的正确性就难以保证了，在改正已有错误的同时可能引入新的潜在错误。这种程序难于维护的原因之一是一段代码的修改对其它代码通过变量传递的影响难于确定[2]。因此，我们希望限制过程与变量的联系。

Gallagher等[3]在Weiser[4]的程序分片概念基础上提出了程序分解片的概念，即对指定变量（或变量集）计算有贡献的语句子集，并用于软件维护。我们在此基础上，利用变量上建立的分解片构造变量的修改函数集和使用函数集来封装对该变量的存取。用这种方法封装软件中关键的变量可使软件的整体结构变得较为直观和简洁，有利于软件的理解和维护。

二、程序分解片

为了讨论方便，我们首先给出有关概念的定义或解释。[3],[4],[5]。

定义1 一个二元组(v, n)称为一个分片准则，其中v为变量或变量集，n为语句。

在以下的讨论中，我们把变量与变量集混用，过程与函数混用。

定义2 设(v, n)为某分片准则，一个程序按该准则产生的一个分片S(v, n)是一个独立的可执行程序，对于相同的输入，分片与原程序使v在语句n处具有相同的值。

定义3 设O(P, v)为程序P中输出v的输出语句集，1是P最后执行的语句，N=O(P,v)∪{1}，则语句集$\bigcup_{\bullet\in N}$S(v, n)称为v的分解片，记为S(v)。图2，3，4分别是图1中程序对nl，nw,nc的分解片。该程序是UNIX实用程序wc的梗概[3]，有关计算程序分片的技术参见[4],[5],[6],[7],[8]。一个分解片中删除了所有输出语句后得到的分片称为输出受限分解片。

定义4 设S(v)和S(ω)为输出受限分解片，ω≒v，若S(v)真包含于S(ω)，则称S(v)

陈学锋 博士生，主要研究方向：操作系统剪裁、程序源码转换、软件工程，王振宇 博士生，主要研究方向：并行程序设计、实时操作系统。收到日期：92—12--05。

强依赖于S(ω)。

定义5 若输出受限分解片S(ω)不强依赖于任何其它分片，则称S(ν)为最大分解片。

定义6 设S(ν)和S(ω)为输出受限分解片，在S(ν)∩S(ω)中的语句称为相关语句，非相关语句称为无关语句。图2中语句12，13是S(nl)的无关语句，图3中14—19，图4中的11为无关语句，其余为相关语句。

定义7 一个变量称为相关变量，是指在它的分解片中存在对它的一个赋值语句是相关的；一个变量称为无关变量，是指对它的所有赋值语句均是无关的；一个相关变量称为强相关的，是指它对应的分解片是强依赖的；否则称为弱相关的。图2至图4中变量nl，nw，nc均为无关变量；c，inword为强相关变量。

```
1  #define YES 1
2  #define NO 0
3  main( )
4  {int c, nl, nw, nc, inword,
5  inword=NO;
6  nl=0;
7  nw=0;
8  nc=0;
9  c=getchar( );
10 while( c !=EOF){
11 nc=nc+1;
12 if (c=='\n')
13 nl=nl+1;
14 if(c==' '| |c=='\n'| |c=='\t')
15 inword=NO;
16 else if(inword==NO) {
17 inword=YES;
18 nw=nw+1;
19 }
20 c=getchar( );
21 }
22 ptintf("%d\n", nl);
23 ptintf("%d\n", nw);
24 ptintf("%d\n", nc);
25 }
```

图1 wc.c

```
3  main( )
4  {int c, nl,
6  nl=0;
9  c=getchar( );
10    While (c!=EOF{
12    if (c=='\n')
13    nl=nl+1;
20    c=getchar ( ) ;
21    }
22    ptintf ("% d\n", nl);
25    }
```

图2 分解片S(nl)

```
1  #define YES 1
2  #define NO 0
3  main( )
4  {int c, nw, inword,
5  inword=NO;
7  nw=0;
9  c=getchar( );
10    While (c!=EOF){
14    if (c==' '| |c=='\n'| |c=='\t')
15    inword=NO;
16    else if (inword==NO){
17    inword=YES;
18    nw=nw+1
19    }
20    c=getchar ( ) ;
21    }
23    ptintf ("% d\n", nw);
25    }
```

图3 分解片S(nw)

```
3  main( )
4  {int c, nc,
8  nc=0;
9  c=getchar( );
10    While (c!=EOF){
11    nc=nc+1;
20    c=getchar ( ) ;
21    }
24    ptintf ("%d\n", nc );
25    }
```

图4 分解片S(nc)

三、软件结构图

我们用软件结构图来描述软件整体结构内各种成员之间的关系[9]。一个软件结构图是顶点和弧均带权的有向图，顶点表示各类控制、数据和类型成员，弧表示成员间的各类关系；顶点权用于区别顶点的类型，如：语句、复合语句、过程、变量、复合变量以及类型名等，弧权用于区别弧的类型，如：数据依赖、控制依赖、调用、修改、引用、类型引用等。

软件结构图从抽象角度描述软件中各类成员间的各种关系。如果把图1中过程main看成控制成员，变量nl，nw，nc，inword，c等看成数据成员，则图1对应一个结构图，如图10(a)。其中弧上的权m表示修改，u表示使用。从中我们可以看出所有变量均由

main处理，整体结构不分层次。

四、无关变量的封装

为了调整软件结构，我们希望把软件中某些重要的变量封装起来，形成近似于抽象数据类型的独立部件。首先我们考虑无关变量的封装。由于无关变量值的计算对其它变量的计算无影响，我们可以只考查关于该变量的分解片，实际上，无关变量在其它分解片上是不可见的。

设v为无关变量，S(v)为v对应的分解片，I(v)是S(v)中与其它最大分解片无关的语句集，D(v)为S(v)中与其它最大分解片相关的语句集。显然有$I(v)\cup D(v)=S(v)$，$D(v)\cap I(v)=\Phi$和$I(v)\neq\Phi$。若$D(v)=\Phi$，则S(v)已把v从程序其它部分分开，经过换名S(v)就是v的修改——使用函数。若$D(v)\neq\Phi$，则在S(v)中按语句顺序D(v)中的语句和I(v)中语句交替出现。设I(v)被划分为$I_1, I_2, \cdots, I_k$，根据控制相关和数据相关关系可以证明：一个语句是相关语句，则该语句在控制流和数流上依赖的语句都是相关的；一个语句是无关的，则嵌在该语句内的语句都是无关的。故$I_i(i=1, 2, \cdots k)$是一个（嵌套）语句或语句序列。每个I_i可以做为一个函数体独立出来，它们引用的值从S(v)中调用处用值参传递或直接从相应变量中提取，从而形成v的修改函数集M(v)。由于v是无关变量，v的使用仅在S(v)中，故v的使用函数集U(v)为简单的恒等函数集。

```
*    int nl=0;
*    calculateNl(c)
*    int c;
*    {
12   if(c=='\n')
13   nl=nl+1;
*    }
```

图5 nl的修改函数

```
*    int nw=0;
*    calculateNw(c)
*    int c;
*    {
*    static int inword=NO;
14   if(c==' '||c=='\n'||c=='\t')
15   inword=NO;
16   else if (inword==NO) {
17   inword=YES;
18    nw=nw+1;
19    }
*    }
```

图6 nw的修改函数

```
*    int nc=0;
*    calculateNc ( )
*    {
11    nc=nc+1;
*     }
```

图7 nc的修改函数

```
*    int getNl ( )
*    {
*      return nl;
*    }
```

图8 nl的使用函数

```
3    main ( )
4    {int c;
9    c=getchar ( );
10   while(c!=EOF)  {
*    calculateNl (c);
*    calculateNw (c);
*    calculateNc ( );
20    c=getchar( );
21    }
22    ptintf("%d \n", getNl( ));
23    ptintf("%d \n", getNw( ));
24    ptintf("%d \n", getNc( ));
```

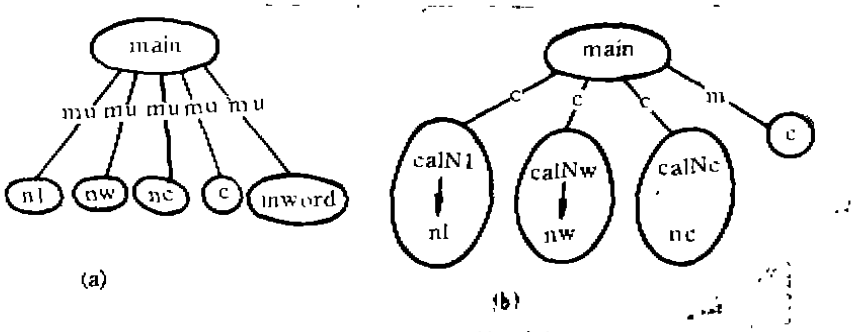


图10 wc.c的结构图

```
25  }
```

图9 wc。c的重构

以图1为例，nl，nw，nc为无关变量，分离无关语句为相应修改函数，参见图5至图7。nl的使用函数见图8。主程序的主要内容是三个最大分解片的公共部分，即S(c)，见图9。对应的软件结构图为图10(b)。新程序的结构在层次和概念上较清晰，如：字的计算方法的改变只需改变有关nw部分的内容，程序的其它部分不受影响，见图11。

```
*   int nw=0;
*   calculateNw(c)
*   int c;
*   {
*   static int lastch=0;
14   if(isspace(lastch) &&isalpha(c))
18   nw=nw+1;
*    lastch =c;
*   }
```

图11 nw的新修改函数

五、弱相关变量的分离与封装

从第二节知，弱相关变量对应的分解片仍为最大分解片。但该变量的值对其它变量的计算有影响，所以该变量将在相关语句中出现，因而对该变量的赋值语句也会出现在相关语句中，因此对该变量的修改和使用将出现在多个分解片中。我们考虑出现该变量的所有分解片。

设v是弱相关变量，S(v)是v的分解片，$S(\omega_1)$，…，$S(\omega_n)$分别为有v出现的最大分解片。令$P_n=\bigcup_{i=1}^{n}S(\omega_i)\cup S(v)$。显然v在$P_n$中是与其它分解片无关的，但我们希望把v从$S(\omega_1)$，…，$S(\omega_n)$中分离出来，故不能简单地应用上节的方法。

首先我们用上节的方法把S(v)中的无关语句过程化，然后考查S(v)的相关语句中对v定值的语句集D((v)和使用v值的语句集U(v)。对D(v)中的语句ni，把仅被ni依赖的语句集Di独立为一个函数。该函数初始值由值参传递。这样的过程一直做到P_n中v的所有定值点被过程化，从而形成v的修改函数集。对U(v)中的语句nj把仅被nj依赖的语句集Dj独立为一个函数，函数的初始值由值参传递，并返回计算结果。如果使用函数与修改函数的调用点相邻，则使用函数可以并入修改函数。我们仍借用[3]中的例子，见图12。其中只有a和e是弱相关变量，其余为强相关变量。语句8、10、11是S(a)的无关语句，语句9是S(e)的无关语句，将变量a和e分离并封装成两个子单元见图13。

```
1    main( )
2    {
3    int a, b, c, d, e, f;
4    c=4;
5    b=c;
6    a=b+c;
7    d=a+c;
8    f=d+h;
9    e=d+8;
10   b=30+f;
11   a=b+c;
12   }
```

图12 弱相关变量a和e

```
/* 变量a的子单元：*/
*  int a;
*  calculateA1(d, b, c)
*  { int f;
8    f=d+b;
10   b=30+f;
11   a=b+c;
*    }
*    int calculateA2(b, c)
*    {
6    a=b+c;
7    return a+c;
*  }
/* 变量c的子单元：*/
*  int e;
*  calculateF(d)
*  {
9    e=d+8;
*  }
/* 主程序，*/
1  main( )
2  {
3  int b, c, d;
4  c=4;
5  b=c;
*  d=calculateA2(b, c);
*  calculateA1(d, b, c);
*  calculateF(d);
12 }
```

图13 变量a和e的封装

六、过程分解片

一个实用的程序通常是由多个过程组成的。程序分解片主要适用于单过程程序，这里我们把程序分解片的概念扩展为能够适

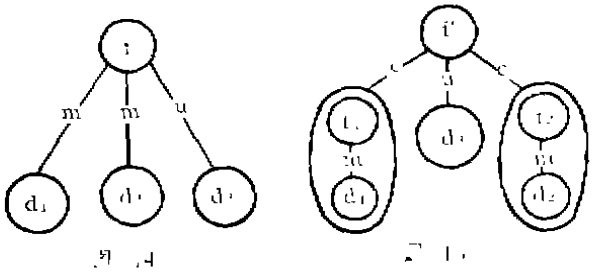
用于多个过程程序的情形，称它为过程分解片。

定义8 一个过程F针对全局变量v在语句n处的过程分片为一个过程S，对任意一组输入(F的参数及用到的全局变量的一组值)，F和S使v的值在语句n处相同，记为$S_f(v,n)$。

定义9 一个过程F对全局变量v的过程分解片为$\bigcup_{n\in D_f} S_f(v,n)$，其中$D_f$是F中所有对v定值的语句的集合，记为$S_f(v)$。

类似地可定义过程分解片间的相关关系及语句间的相关关系。

设v为一个全局变量，如果修改v的过程只修改v一个全局变量，则这些过程就形成了v的修改函数集。否则v与其它全局变量之间存在公共修改函数。如果希望分离并封装v，则必需分解修改v的并修改其它全局变量的过程。不失一般性，设过程f修改全局变量d_1和d_2并使用d_3，如图14所示。若d_i(i=1,2)在$S_f(d_i)$中是无关的，则相应无关语句构成d_i的修改函数f_i；若d_i不是无关的，则修改d_i的语句及只被这些语句依赖的语句形成d_i的修改函数集，如图15所示。



七、结束语

我们利用程序分解片和过程分解片来封装某些变量(或变量的集合，如结构型变量)，即对该变量的存取必须通过它的修改函数和使用函数（使用函数主要用于隔离数据的表示形式)。可以证明对由无关语句形成的函数的修改不影响其它变量的计算，并且相关变量之间的相互影响必然是通过各自的修改函数实施的。通过程序重构，在程序的整体结构中消除了一些某共享变量，代之以以数据为中心的子单元，增加了软件结构的模块性和层次性，对程序的理解、维护和重用是有帮助的。

参考文献

[1] H. Schildt, Using Turbo C++, Osborne McGraw-Hill Press., 1990

[2] N. Schneidewind, the State of Software Maintenance, IEEE Trans. Software Eng. Vol. SE-13, Mar. 1987

[3] K. B. Gallagher, et al., Using Program Slicing in Software Maintenance, IEEE Trans. Software Eng. Vol. SE-17, No. 8, 1991

[4] M. Weiser, Program Slicing, IEEE Trans. Software Eng. Vol. SE-10, July 1984

[5] 陆奇，张福波，钱家骅，程序分片：其改进算法与在程序验证中的应用，计算机学报，1988年，第4期

[6] H. Agrawal, et al., Dynamic Program Slicing, In Proc. of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation, White Plains, New York, 1990

[7] K. J. Ottenstein, et al., The Program Dependence Graph In a Software Development Environment, ACM SIGPLAN Notices, Vol. 19, No. 5 1984

[8] S. Horwiitz, et al., Interprocedural Slicing Using Dependence Graphs, ACM Trans. Programming Languages and System, Vol. 12, No. 1 1990

[9] 郭福顺，陈宏锋，软件抽象结构的自动分析，小型微型计算机系统，第13卷，第10期，1992年

[10] J. Ferrante, et al., The Program Dependence Graph and Its Use In Optimization, ACM Trans. on Programming Languages and Systems, Vol. 9, No. 3, 1987