ISSN1000-9825 Journal of Software 软件学报 2000,11(6):836~840

一种 Ada83 服务性任务向 Ada95 保护对象变换的方法*

李帮清 徐宝文

(东南大学计算机科学与工程系 南京 210096)

(武汉大学软件工程国家重点实验室 武汉 430072)

E-mail: bwxu@seu.edu.cn

摘要 任务是 Ada 语言支持并发程序设计的基础,它提供了一种进程同步和通信的良好机制.但也存在一类被动的、专为其他任务服务的服务性任务.服务性任务的存在增加了系统的负担,降低了系统的性能.如果能将服务性任务变换为保护对象,则可以明显地提高系统的性能,减少维护的费用.该文提出了一种在源程序级将 Ada83 服务性任务变换为 Ada95 保护对象的方法.先给出服务性任务应满足的条件、识别和表示服务性任务的方法,最后讨论此方法的可行性.与其他方法相比,此方法具有假设条件少、识别和变换效率高以及容易验证等特点.

关键词 Ada,服务性任务,保护对象,并发程序设计,数据流图.

中图法分类号 TP311

Ada 语言中任务间的会合机制是一种高级的通信机制,利用它可以避免因使用诸如信号灯等低级通信原语而带来的困难[1,2].但是会合机制不是一种轻型(lightweight)的通信机制,并不能完全满足某些应用需求.很多情况下,在用任务机制对共享数据或其他共享资源进行控制时需要增加额外的任务.由于这种类别的任务是为其他任务服务的,它只是应其他任务的要求而动作,我们称之为服务性任务(其功能类似于管程).不管有没有任务与之会合,服务性任务在其作用域范围内始终是运行的,这就造成了系统资源的浪费,而且有可能因使用不当而产生难以控制的竞争条件.同时,大量的上下文切换增加了系统的负担,降低了系统的性能.保护对象是 Ada95 新增的用于任务通信和互斥的设施.保护对象封装了一组数据以及可施用于其上的操作.它自动实现对所保护数据的互斥写、并发读操作,而且保护操作之间的同步也是自动的,这使得它比低级机制具有更高的安全性与灵活性[1,3].同时,保护对象的 requeue 语句还可以在必要时将调用任务再排队到相容的保护入口的队列中去,从而可以对可能出现的竞争条件优先进行控制[3,4].可见,将满足一定条件的服务性任务变换为保护对象可明显改善程序的可读性,提高系统的性能,增加系统的安全性.

并行程序的转换与变换,特别是不同并行机制之间的变换,是一个很有理论意义与实用价值的研究课题.并行机制之间的有效变换不仅可以明显地提高系统的性能、减少维护的费用,而且也有助于并行技术及编译技术的研究.为了有效地维护 Ada 程序、提高软件的性能,同时也为了研究各种并行机制之间的关系,我们进行了 Ada83 服务性任务向 Ada95 保护对象变换技术的研究.本项研究主要解决两个问题:(1) 什么任务是服务性任务?如何识别?(2) 如何进行变换?在文献[5]中,C. D. Locke 等人提出用保护记录替代被动任务,并给出了同种服务的被动任务及保护记录的实现方法,但其讨论不够详尽,没有给出具体的变换方法和变换的条件.文献[6]提出了一种简单的基于源程序级的变换方法.作为此工作的延续,本文在文献[6]的基础上,给出了识别服务性任务的方法和表示服务性任务的数据流图,并讨论了根据此变换方法、语法树和数据流图来实现具体的算法.

---

* 本文研究得到华英文教基金资助.作者**李帮清**,1975 年生,硕士生,主要研究领域为并发程序设计,软件工程.**徐宝文**,1961 年生,教授,博士生导师,主要研究领域为程序设计语言,软件工程,智能软件与网络软件技术.

本文通讯联系人:徐宝文,南京 210096,东南大学计算机科学与工程系

本文 1998-12-15 收到原稿,1999-06-07 收到修改稿

1 服务性任务的识别和表示

1.1 服务性任务所应满足的条件

由于变换的目标是保护对象,所以服务性任务应满足一定的保护对象所需满足的条件.由文献[3],保护入口不能包含可能阻塞的操作,而我们的变换方法将把任务体用保护入口来实现,所以任务体也不应包含可能阻塞的操作.同时,因为保护对象的体不能直接声明实体[3],所以任务体中不嵌套包含任务声明,但被嵌套的子程序可用保护对象的私有子程序表示出来.

服务性任务是一种被动的任务,它只因入口被调用或选择备选被选中才动作,因此,在语法上它应表现为:在任务体中除了基本循环语句、选择语句和接收语句外,其他语句应包含于一个接收语句或选择备选中,且每个选择备选的第1条语句是接收语句或终止语句.

由于实际存在的任务各种各样,这给变换带来了一定的困难,因此,我们提出一个假设:除了选择、接收和基本循环语句之外,其他语句(包括初始化语句序列)均为基本语句序列,即赋值语句、循环语句、条件语句、分支语句和过程调用语句的复合语句.

1.2 服务性任务的识别

服务性任务的识别方法是基于语法分析和数据流图进行的.先将所有任务的入口声明与调用关系用简单的数据流图表示出来(类似于文献[7]的M1形),图中每个出度为零的结点所代表的任务将是一个候选服务性任务,再由语法分析得到一个语法树和任务的数据流图.在识别时,针对语法树和数据流图的结点所包含的内容进行分析,并不断改造数据流图,等到识别完毕,保护对象就可以直接由数据流图生成.

根据语法树和数据流图进行分析主要是通过分析找出那些不满足条件的结点.由于服务性任务不能包含可能阻塞的操作,即(任务、保护对象)入口调用语句、延迟语句、中止语句、任务的产生与激活操作、保护子程序调用语句、调用包含可能阻塞操作的子程序及由语言定义的操作文件的输入、输出程序包中的子程序[3]*,因此,所有任务体的语法树包含这些类型的结点都不是服务性任务.由于服务性任务不能包含任务声明,所以任务体的语法树中根结点和与 **begin** 相对应结点之间的所有结点都不应是任务声明结点,否则就不是服务性任务.因为在服务性任务中每个选择备选的第1条语句是接收语句或终止语句,且任务体中除了选择、接收、基本循环语句和初始化语句序列之外,其他语句应包含于一条接收语句或选择备选中,在语法树中表现为所有 **select** 或 **or** 结点的第1个儿子应为接收语句或终止语句;第1个选择或接收语句结点之后的所有结点,除了选择、接收、基本循环语句和必要的控制结点(**when** 或 **while** 等结点)之外,其父结点类型均为接收语句或选择备选(**select** 或 **or**)结点.同时,要分析基本语句的组成方式,即除了选择、接收、基本循环语句和必要的控制结点外,其他结点所对应的语句应为赋值语句、循环语句、条件语句、分支语句和过程调用语句的复合语句.将这些情况综合起来,一次遍历就可以识别出服务性任务.

1.3 服务性任务的表示

为了便于变换的实现,在识别出服务性任务之后,我们用一定形式的数据流图将其表示出来.数据流图DFD (data flow diagram)由变换(transform)、数据流(data flow)和数据流上的标识(mark)组成.数据流图 G 可以形式地定义为七元式:

$$(T, A, DF, CF, f_{DF}, f_{IN}, f_{OUT}).$$

其中,$T=CT \cup SB \cup SE$,分别表示普通的变换(包括 **begin** 和 **end** 结点)结点、选择语句的开始结点和结束结点;$A=CA \cup NB \cup NE$,分别表示普通接收语句(不嵌套包含其他接收语句)结点、嵌套包含其他接收语句的接收语句的开始结点和结束结点;DF 表示相联结的两个结点之间数据流的关系;CF 表示相联结的两个结点之间控制流的关系;f_{DF} 是对偶 (t,a) 到信息表 $GUARD$ 的映射(亦即边 (t,a) 的一个标识),表示执行接收语句前应满足的

---

* 根据文献[3],接收语句、选择语句和保护对象的外部再排队操作也是可能阻塞的操作,但在变换之后,接收语句和选择语句都将不复存在,由相应的保护入口或保护子程序来实现;而保护对象的外部再排队操作显然不会在任务体中出现.

条件(哨兵),其中 $t\in T\cup A, a\in A$,$GUARD$ 是所有布尔表达式的集合,初值为接收语句的哨兵集(若没有,则为 $\{TRUE\}$),在变换的过程中将不断改变;f_{IN}是对偶(t,a)到信息表 IN 的映射,表示其源结点的输入参数集(**in**,**in out** 模式,若没有,则为$\varnothing$),其中 $t\in NB, a\in T\cup A$,IN 是所有输入参数的集合;f_{OUT}是对偶(t,a)到信息表 OUT 的映射,表示其源结点的输出参数集(**out**,**in out** 模式,若没有,则为$\varnothing$),其中 $t\in T\cup A, a\in NE$,OUT 是所有输出参数的集合.从图论的观点看,$N=T\cup A$ 为结点集;$E=DF\cup CF$ 为边集;f_{DF},f_{IN},f_{OUT}为边上的标识函数.我们用矩形框表示 T,用椭圆表示 A.

2 变换思想与方法

服务性任务提供的服务通过入口调用语句完成,并由相应的接收语句实现[2],而保护对象提供的服务是通过保护入口和保护子程序的调用实现的[3].所以,在变换时将任务入口变换成保护入口或保护子程序.由于任务体中各接收语句之间存在着控制流关系[2],而保护入口和保护子程序之间不存在显式的控制流关系[3],为保证原控制流关系不变,我们为保护入口和保护子程序构造了必要的变量和语句以显式地控制它们的调用顺序.接收语句之间的控制流关系在识别服务性任务时已表示在数据流图中.

2.1 规格说明的变换

任务的规格说明主要包含入口声明,此即任务提供的服务.保护对象的规格说明包括两个部分:对外界可见的保护入口声明和保护子程序声明以及保护对象私有的对象声明、保护入口声明和保护子程序声明[3].为了将接收语句的哨兵和它们之间的控制流关系显式地表示出来,我们将所有任务入口声明都变换为保护入口声明,相应的接收语句的哨兵变换成保护入口体的入口栅,若无哨兵,则入口栅为真;当变换结束时,将所有入口栅仍为真(即$f_{DF}(t,a)=\{TRUE\}$)的保护入口变换为保护子程序,以提高运行效率.

2.2 任务体的变换

由于任务体比较复杂,不失一般性,我们分别处理顺序接收语句序列、嵌套接收语句、选择语句和同一入口有多个接收语句这 4 种情况,将它们结合起来就可得到任务体的变换方法.值得注意的是,由于保护对象的体中不能直接声明任何实体,所有声明均存在于保护对象规格说明的私有部分,所以,任务体的声明部分应移到此私有部分.同时,在后面的变换过程中所构造的所有附加变量和保护入口也应在此私有部分加以声明和初始化.

为保证任务体中各接收语句的控制流关系不变,需要为每个保护入口或保护过程构造布尔变量来控制它们的执行次序.由于这些附加变量直接决定了保护入口体或保护过程体是否能立即执行,使用它们的最可能的位置是在保护入口体的入口栅部分(使之成为入口栅的一部分),这就是在任务规格说明变换的方法中将接收语句都变换为保护入口的原因.

为表示方便,在下面的描述中将忽略语句的细节部分,而用不同的大写字母来代替.

2.3 顺序接收语句序列的处理

任务体中顺序接收语句序列的执行次序是从前到后逐个执行的,相应的保护入口也应逐个执行,这可以通过对与每个保护入口对应的附加变量的交替赋值来实现.对附加变量的赋值原则为:在每个保护入口中将自己的附加变量赋为假(禁止自己),而将控制流关系上的下一个保护入口的附加变量赋为真(使能下一个);赋初值的原则是将(控制流关系上)第 1 个保护入口的附加变量赋为真,而将其他保护入口的附加变量赋为假.以下同.

假设存在接收语句序列,其入口标识符依次为 $A_1\ldots A_i\ldots A_n(1\leqslant i\leqslant n)$.构造变量的方法是为每个 A_i 构造一个布尔变量 $Var\_A_i\_S$,将它增加到 A_i 的入口栅中去,与原入口栅进行"与"操作(如,入口栅由 R_i 变为(R_i) **and** $Var\_A_i\_S$,在数据流图中体现为 $f_{DF}(A_{i-1},A_i)$由$\{R_i\}$变为$\{(R_i)$ **and** $Var\_A_i\_S\}$),在 A_i 入口的体最后增加两条对相应变量的赋值语句,比如,$Var\ A_i\_S=FALSE$;$Var\_A_{i\bmod n+1}\_S=TRUE$;赋值顺序由控制流关系决定.

2.4 嵌套接收语句的处理

对于嵌套的接收语句,由于其内外层接收语句可以与不同的任务会合,且外层会合要等到所有内层会合完成后才可完成[3],而在保护对象中,保护入口不可嵌套调用,但保护对象的 **requeue** 语句可以用来模拟嵌套接收

语句的执行,因此,在数据流图中,我们把嵌套的外层接收语句用两个结点(开始结点和结束结点)来表示,并根据外层接收语句体构造最多 n 个私有保护入口(假设此接收语句体包含 n 个顺序的接收语句).

假设存在嵌套接收语句(如图 1 所示),其中 $n\geqslant 0$,S_i 表示基本语句(序列),可能为空.为每个非空语句(序列)$S_i(1<i<n+1)$构造新的私有保护入口,入口名为 $NewEntry\_A\_E_i$,它们的参数与 A 相同(如果有的话),体由 S_i 组成.对于语句(序列)S_{n+1}不论其是否为空,必须构造新的保护入口 $NewEntry\_A\_E_{n+1}$(其参数与 A 相同,体为 S_{n+1}或暂时为空),因为它对应于原外层会合的结束部分.语句(序列)S_1 形成与 A 对应的保护入口;其轮廓(profile)与 A 相同,体为 S_1.为每个新增入口构造一个布尔变量 $Var\_A\_N_i$ 作为入口栅.在数据流图中为每个新的保护入口构造一个结点,插入到相应的位置(即删除边 (B_{i-1},B_i),添加结点 $NewEntry\ \ A\_E_i$,添加边 $(B_{i-1},NewEntry_A_E_i)$和 $(NewEntry\_A\_E_i,B_i)$,将 S_1 插入到与 A 对应的语法树的结点中,并将 A 的结束结点改名为 $NewEntry\_A\_E_{n+1}$),A 的 f_{IN} 和 $NewEntry\_A\_E_{n+1}$ 的 f_{OUT} 由初始 A 的 f_{IN} 和 f_{OUT} 决定,各 f_{DF} 为 $\{Var\_A\_N_i\}$.这些新增加的保护入口对应于相应的语句(序列)S_i,在原有的控制流关系基础上与 A 及诸 B_i 形成新的控制流关系.入口 A 和 B_i 的布尔变量的构造、检测、设置和清除方式及诸 $Var\_A\_N_i$ 的检测、设置和清除方式就和顺序序列 $A;B_1;NewEntry\_A\_E_2;\ldots B_n;NewEntry\_A\_E_{n+1};$(如果对应的语句(序列)$S_i$ 非空)一样,可按第 2.3 节的方法进行.在 A 及 $NewEntry_A_E_i(0<i<n+1)$的最后(对新增变量的赋值语句的后面)增加新的语句:**requeue** $NewEntry\_A\_E_j$;其中 $0<i<j\leqslant n+1,\forall\ k;i<k<j,S_k$ 为空;当 $i=1$ 时表示入口 A.从而实现对应于内层会合的调用在执行时,对应于外层会合的调用任务在等待.入口 $NewEntry\ \ A_E_{n+1}$不包含**requeue**语句,它只包含对相应变量的赋值.

```
accept A(...) do
  S1;
  accept B1(...) do
    ...
  end B1;
  S2;
    ...
  accept Bn(...) do
    ...
  end Bn;
  Sn+1;
  end A;
```

Fig. 1
图 1

对于外层接收语句的参数,由于它们可以在内层被引用和赋值,而在变换成保护对象后已将各个接收语句分开处理,所以必须为每个参数构造一个相应于保护对象的全局变量 A_Para_i(设 $Para_i$ 是 A 的第 i 个参数)来代替对应于此参数在保护对象中的操作.A_Para_i 的类型与 $Para_i$ 相同.在对应于外层会合的开始处(保护入口 A 的开始处,即开始结点中)将所有 **in**,**in out** 模式(即 f_{IN} 集)的参数赋值给相应的临时变量;而在对应于外层会合的结束处(保护入口 $NewEntry\ \ A_E_{n+1}$的最后、对新增布尔变量的赋值语句之前,即结束结点中)将所有与 **out**,**in out** 模式(即 f_{OUT} 集)的参数对应的临时变量赋值给这些参数.

2.5 选择语句的处理

选择语句可以包含多个选择备选,这些选择备选的执行是互斥的[2].因为每个选择备选的第 1 条语句是接收语句或终止语句,这些接收语句在控制流关系中应处于同等位置,所以用第 1 个选择备选的第 1 个接收语句来构造变量.终止语句是用来结束任务的执行的,变换成保护对象后,可以忽略,因为保护对象是完全被动的,不需要显式的终止语句[3].而对于选择备选中不被任一接收语句包含的语句(序列),可以用与相应接收语句对应的保护入口体来执行.

如果某选择语句包含于一个顺序接收语句序列中,那么在构造变量时取第 1 个选择备选的第 1 个接收语句的入口名来构造,这个变量被此选择语句在控制流关系中的前一接收语句所设置,并被每个选择备选的第 1 个接收语句的相应的保护入口体检测.若此选择语句之后还存在接收语句,则在对应于每个选择备选的第 1 个接收语句的保护入口体的最后清除此变量,并设置与下一保护入口相应的布尔变量.

2.6 同一入口多个接收语句情况的处理

在任务体中,对应于同一入口可以有多个接收语句,它们的哨兵和体可能各不相同,甚至某些接收语句可能有哨兵,而另一些没有[2].而在保护对象中,对应于每一保护入口或保护子程序只能有一个体[3],因此,应将各接收语句的体和哨兵适当地组合起来,形成相应保护入口的体.假设存在对同一个入口 A 的多个接收语句,按这些接收语句单独出现(即忽略其他接收语句)的次序将它们从 1 开始编号,对于第 i 条对语句,将在前面为它构造的布尔变量 Var_A_S,$Var\ \ A\_N_j$ 和临时变量 A_Para_i 的全部出现分别改为 $Var\ \ A\_S\_i$,$Var\_A\_N_j\_i$ 和

$A\_Para_i\_i$.程序中依赖于这些变量的其他部分也应作相应改变.最后,将多个接收语句组合成一个保护入口(在数据流图中,各结点不变;生成代码是将它们合并在一起),入口栅的构造方法如下:将每个接收语句的哨兵与对应的新构造的布尔变量进行"与"运算,成为一个变量组,各变量组进行"或"运算构成入口栅.用 if 语句将变量组与对应的体组合起来构成保护入口的体,各变量组分别成为条件语句的不同分支的检测条件;而分支的内容对应于任务体中接收语句的内容,并在每个分支中分别对相关的布尔变量进行赋值.

将以上几种情况综合起来即可得到具体的变换算法.在变换时,主要是变换数据流图,数据流图变换完毕即可生成保护对象的代码.在以上讨论中,凡是涉及到修改代码之处均在语法树中进行.

3　讨　论

本文的方法还是基于一定的假设条件,如何将条件放宽并对所生成的保护对象进行优化是下一步要进行的工作.由于保护对象是被动的,它自动实现数据的同步与互斥,使得将服务性任务变换为保护对象后可以明显提高系统的性能.另外,从方法学的角度来说,会合是面向控制的通信机制,而保护对象是面向数据的设施,用它对共享数据进行保护更符合面向对象(OO)的思想,这是一种公认的良好的程序设计方法[1,2].

参考文献

1　Andrews G R. Concurrent Programming: Principles and Practice. Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc., 1991

2　ANSI/MIL-STD-1815A-1983 (ISO 8652 1987). Reference Manual for the Ada Programming Language, 1983

3　ISO/IEC 9652: 1995(E). Ada reference manual——language and standard libraries, 1995

4　XU Bao-wen. Ada95 protected objects and data-oriented synchronization. Computer Research and Development, 1997,34(1):72～77
(徐宝文. Ada95 保护对象与面向数据的同步. 计算机研究与发展,1997,34(1):72～77)

5　Locke C D, Mester T J, Vogel D R. Replacing passive tasks with Ada 9X protected records. ACM Ada Letters, 1993,13(2):91～96

6　Li Bang-qing, Xu Bao-wen, Yu Hui-ming. Transforming Ada serving tasks into protected objects. In: Proceedings of ACM SIGAda Annual International Conference (SIGAda'98). Washington, DC: ACM Press, 1998. 240～245

7　Canfora G, Cimitile A, Carlini U D. A reverse engineering process for design level document production from Ada code. Information and Software Technology, 1993,35(1):23～34

An Approach for Transforming Ada83 Serving Tasks to Ada95 Protected Objects

LI Bang-qing　　XU Bao-wen

(*Department of Computer Science and Engineering　Southeast University　Nanjing*　210096)
(*State Key Laboratory of Software Engineering　Wuhan University　Wuhan*　430072)

Abstract　Tasks are the basic facilities of concurrent programming in Ada, which provides a good mechanism of synchronization and communication. However, there exists a kind of serving tasks that are passive and are used as servers. The existence of serving tasks makes systems have a little overhead and degrades system performance. When serving tasks are transformed to protected objects, the performance of systems can be improved. In this paper, the authors propose a method of transforming Ada83 serving tasks to Ada95 protected objects at the source code level. First, conditions that a serving task must satisfy are discussed. Then the methods of identifying and transforming serving tasks are discussed, followed by discussions on the feasibility of the methods. Compared with other methods, the proposed methods have less hypotheses and higher efficiency and are easier to verify.

Key words　Ada, serving task, protected object, concurrent programming, data flow diagram.