一种基于程序可达图的并发程序依赖性分析方法

戚晓芳[1],徐宝文[1,2],周晓宇[1,2]
(1. 东南大学计算机科学与工程学院,江苏南京 210096; 2. 江苏省软件质量研究所,江苏南京 210096)

摘 要: 依赖性分析是一种重要的程序分析手段. 针对多线程共享变量通信机制,本文在提出一种新的并发程序表示—线程交互可达图(tIRG)的基础上,从全局分析并发程序的依赖关系,构建了以程序状态和语句二元组为节点的并发程序依赖图(MSDG). 与传统的以语句为节点的并发程序依赖图相比,MSDG图中依赖关系不仅精确,且具有可传递性,对其遍历可获得高精度的并发程序切片,精度和效率较其它高精度切片方法有显著提高.

关键词: 并发程序; 可达性分析; 依赖性分析; 程序切片
中图分类号: TP311 **文献标识码**: A **文章编号**: 0372-2112 (2007) 02-0287-05

An Approach to Analyzing Dependence of Concurrent Programs Based on Program Reachability Graphs

QI Xiao-fang[1], XU Bao-wen[1,2], ZHOU Xiao-yu[1,2]
(1. *College of Computer Science and Engineering, Southeast University, Nanjing, Jiangsu* 210096, *China*;
2. *Jiangsu Institute of Software Quality, Nanjing, Jiangsu* 210096, *China*)

Abstract: Dependence analysis is an important technique to analyze programs. This paper proposes a novel representation for multi-threaded programs with shared variables, which is called thread interaction reachability graph (tIRG). Based on tIRG, dependences in concurrent programs are analyzed globally and a new dependence graph called MSDG, which vertex is a 2-tuple composed of program state and statement, is constructed. Compared with traditional concurrent program dependence graph which vertex is statement, dependence relation in MSDG is precise and transitive. In contrast to other high-precision slicing methods, more precise slice will be obtained efficiently by traversing MSDG.

Key words: concurrent programs; reachability analysis; dependence analysis; program slicing

1 引言

依赖性分析是一种重要的程序分析手段,在程序的调试、测试、维护、度量等软件工程活动中得到了广泛应用[1~10]. 目前顺序程序依赖性分析的研究较为成熟,并发程序依赖性分析由于并发程序执行的不确定性,尚有许多问题有待解决,其中如何有效地表示并发程序的执行,解决语句间依赖关系不可传递性问题,提高依赖分析精度等一直是研究的热点[2~10]. 尽管现有的一些方法从某种程度上降低了依赖关系不可传递性问题对依赖分析结果的影响,但这些方法没有从根本上解决依赖关系不可传递性问题,依赖分析精度仍有待提高[2~9].

基于共享变量的通信是一种基本的并发机制,对它的研究具有一定的普适性. 在已有工作的基础上[2,7,10],本文针对多线程共享变量通信机制,对传统的可达性分析进行了扩充[11],生成线程交互可达图(tIRG),基于tIRG图从全局分析并发程序中的依赖关系,构造以程序状态和语句二元组为节点的并发程序依赖图(MSDG). 与传统的以语句为节点的并发程序依赖图相比,MSDG图中的依赖关系更为精确,并且具有可传递性,可用于高精度依赖性分析. 本文首先分析了依赖关系不可传递性问题,然后构建了tIRG图和MSDG图,证明了MSDG图中依赖关系的可传递性,最后讨论了MSDG图在并发程序切片中的应用.

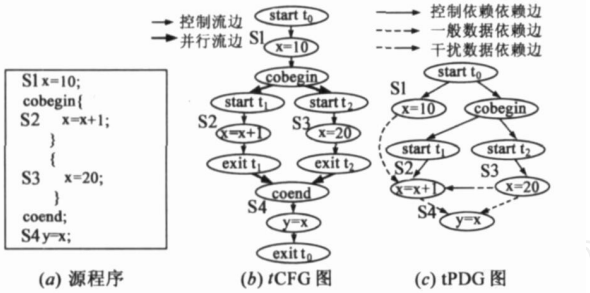
2 问题分析

本文所涉及的并发程序是所谓的轻度并发程序,其并发单元为线程,各个线程共享同一地址空间. 线程间通过共享变量进行通信,共享变量的读写是原子操作. 用 cobegin 和 coend 分别表示线程的创建和终止[8,9]. 为方便描述,文中将 cobegin、coend 以及共享变量读写语句统称为交互语句.

并发程序中的每个线程都有一个执行流程,其执行可用控制流图(CFG)来描述. CFG 图为一四元组$\langle S, E, s_{start}, s_{exit}\rangle$,S 为节点集,线程中的每个语句都对应图中的一个节点,边集 $E=\{(s_i,s_j)\mid s_i,s_j\in S$ 且语句 s_i 执行后可能立即执行 $s_j\}$,s_{start}和s_{exit}分别为线程的入口和出口节点. 若语句序列$(s_1,s_2,\cdots,s_n)$满足 $\forall 1\le i<n:(s_i,s_{i+1})\in E$,则称$(s_1,s_2,\cdots,s_n)$为该线程的

收稿日期:2006-01-04;修回日期:2006-10-19
基金项目:国家自然科学基金(No. 60373066,No. 60425206)

一个从 s_1 到 s_n 的路径. 添加从 cobegin 节点到所创建线程的入口节点以及从线程的出口节点到 coend 节点的边(称为并行流边),可连接并发程序中各线程的 CFG 图,生成线程控制流图(tCFG),表示并发程序的执行[8,9]. 图 1 给出文献[9]中的一个并发程序实例,在本文中记为实例 1,t_0 创建 t_1 和 t_2,各线程通过共享变量 x 进行通信.



(a) 源程序 (b) tCFG 图 (c) tPDG 图

图 1 实例 1 的源程序、tCFG 图和 tPDG 图

基于 tCFG 图可分析并发程序依赖关系,构造线程程序依赖图(tPDG),tPDG 图中的依赖关系分为控制依赖、一般数据依赖和干扰数据依赖[8,9]. 设 s_1 与 s_2 为并发程序的两个语句,若 s_2 能否被执行取决于 s_1 的执行,则称 s_2 控制依赖于 s_1;若 s_1 与 s_2 顺序执行,s_2 执行时使用到 s_1 定义的变量,则称 s_2 一般数据依赖于 s_1;若 s_1 与 s_2 并发执行,s_2 执行时使用到 s_1 定义的变量,则称 s_2 干扰数据依赖于 s_1;s_2 数据依赖于 s_1 或 s_2 控制依赖于 s_1,统称为 s_2 依赖于 s_1. 控制依赖一般通过分析源程序的控制语句直接获得,一般数据依赖通过数据流方程计算到达 tCFG 图中每个节点的变量定义位置进行分析(类似于顺序程序中的数据依赖分析),干扰数据依赖则通过对并发语句对和变量的定义引用信息来进行分析. 若语句序列 $(s_1, s_2, \cdots, s_n)$ 满足 $\forall 1 \le i < n : s_{i+1}$ 依赖于 s_i,则称 $(s_1, s_2, \cdots, s_n)$ 为并发程序的一个依赖序列(即 tPDG 图中的一个路径);对某依赖序列 $DS = (s_1, s_2, \cdots, s_n)$,若存在并发程序某次执行的语句序列,在该执行序列上 DS 中依赖关系可发生,则称 s_n 可传递依赖于 s_1.

为方便讨论,根据依赖序列是否为并发程序中有效的执行序列,本文将依赖关系不可传递性问题分为两类:第一类指依赖序列是一个有效的执行序列但依赖不可传递的问题,如图 2 中 S1→S2→S4 是一个有效的执行序列,但实例 1 只有两种可能的执行,S3 在 S2 前或在 S2 后执行,在这两种执行情况下 S4 均不依赖于 S1;第二类指依赖序列不是一个有效的执行序列的不可传递性问题,如图 2 实例 2 中 S6→S8→S4 不是一个有效的执行序列,S4 不可能依赖于 S6. 目前,文献[2~9]仅针对第二类不可传递问题提出了一些解决方法,对第一类不可传递问题尚未考虑[2~9](文献[2~7]中并发程序的表示、依赖分析方法以及存在问题与文献[8,9]中类似). 深入分析可发现传统的基于 tCFG 图的语句间依赖分析存在以下缺陷:在 tCFG 图上进行一般数据依赖分析时,忽略了并行流和分支流的本质区别,将并行流简单地作为分支流处理,尽管增加了干扰数据依赖分析,但没有从全局分析并行流间的数据依赖关系,造成依赖关系分析不精确;其次,语句间依赖不能有效地描述并发程序中的依赖关系,需要寻求一种新的依赖关系表示方式.

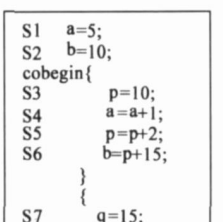
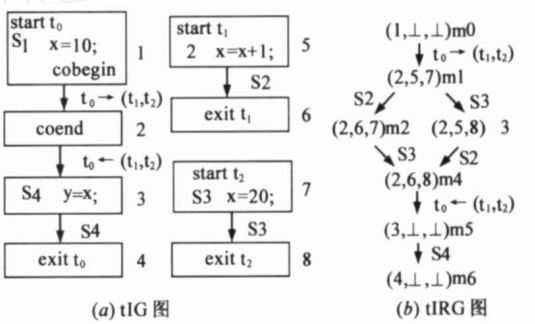


图 2 实例 2 的源程序

3 线程交互可达图

为提高依赖分析精度,本文对传统的可达性分析进行扩充[11],将并行流顺序化处理. 在进行可达性分析之前,根据交互语句将线程分块(称线程域)处理,每个线程域表示为一个节点,节点间的边表示交互活动,由此生成线程交互图,以简化线程表示,提高可达性分析效率. 给定某线程的 CFG 图,从入口语句或交互语句的后继语句起,沿 CFG 路径到达最近的交互语句或出口语句终止,被遍历的一个单入口、多出口 CFG 子图称作一个线程域.



(a) tIG 图 (b) tIRG 图

图 3 实例 1 的 tIG 图和 tIRG 图

定义 1 给定线程 t,其线程交互图(tIG)是一五元组〈N, E, n_s, F, L〉,其中:N 表示线程域集;$E = \{(n_i, n_j) \mid n_i, n_j \in N$ 且在从 n_i 执行到 n_j 之间要与其他线程进行一次交互活动$\}$;标签函数 L 对 E 中每条边映射一个标签以指明交互活动类型;初态节点 $n_s \in N$,表示线程执行入口的线程域集,终态节点集 $F \subseteq N$,表示线程出口的线程域集.

在上述定义中,线程 t 激活和等待子线程 $t_{d1}, t_{d2}, \cdots, t_{dr}$ 的标签分别为 $t \rightarrow (t_{d1}, t_{d2}, \cdots, t_{dr})$ 和 $t \leftarrow (t_{d1}, t_{d2}, \cdots, t_{dr})$,共享变量读写的标签为相应的语句标号.

给定 p 个线程的 tIGs($\langle N_i, E_i, n_s^i, F_i, L_i \rangle, 0 \le i < p$,主程序作为 t_0 处理),根据交互活动的标签,可生成线程交互可达图(tIRG). tIRG 图中的节点,也称作可达标志,由 p 元组表示,第 i 个分量为第 i 个线程中即将执行的线程域,由于线程域可表示线程的执行位置(即线程的执行状态),因此,可达标志(各线程即将执行的线程域的组合)就表示并发程序全局的执行状态(文也称程序状态). 从初始可达标志 m_s 开始,$m_s = (n_s^0, \perp, \cdots, \perp)$,$\perp$ 表示任务未激活或已终止,一次发生一个交互,后继可达标志与前驱可达标志除发生交互的线程的分量发生相应的变化外,其余分量不变.

定义 2 并发程序的线程交互可达图(tIRG)为五元组〈M, E_R, m_s, M_f, L_R〉,其中:节点集 M 为可达标志集;边集 $E_R = \{(m, m') \mid m, m' \in M$ 且在从 m 执行到 m′之间要发生一次线程交互活动$\}$;标签函数 L_R 对 E_R 中每条边映射一个标签;m_s 为初始可达标志,$M_f \subseteq M$,为终态可达标志集;可达标志 m′为可达标志 m 的直接后继当且仅当下面三条之一成立($0 \le i, j <$

p):

(1) $\exists i:((m[i],m'[i])\in E_i \wedge L(m[i],m'[i])=t_i \rightarrow (t_{d1},t_{d2},\cdots,t_{dr}) \wedge \forall 1\le k\le r:(m[dk]=\perp m'[dk]=n_s^{dk}) \wedge \forall j\neq i, d1,d2,\cdots,dr:m[j]=m'([j])$;

(2) $\exists i:((m[i],m'[i])\in E_i \wedge L(m[i],m'[i])=t_i \leftarrow (t_{d1},t_{d2},\cdots,t_{dr}) \wedge \forall 1\le k\le r:(m[dk]\in F_{dk} \wedge m'[dk]=\perp) \wedge \forall j\neq i, d1,d2,\cdots,dr:m[j]=m'[j])$;

(3) $\forall i:((m[i],m'[i])\in E_i \wedge L(m[i],m'[i])=SV \wedge \forall j\neq i:m[j]=m'[j])$.

上述(1)、(2)和(3)分别对应于由于任务的激活、等待终止以及共享变量读写三种交互而生成后继的情形.在构造 tIG 和 tIRG 图时,对不含交互语句的子程序调用语句,可将形参间的依赖映射为实参间的依赖,对含交互语句的子程序调用语句,需将子程序嵌入处理.

4　基于线程交互可达图的并发程序依赖性分析

4.1　相关概念及其性质

为区分同一语句在不同程序状态下的执行,本文将可达标志(表示程序状态)和语句进行组合,并将这种组合称作 M-S 对,用符号 Λ 表示,$M(\Lambda)$ 和 $S(\Lambda)$ 分别映射为 Λ 的状态和语句分量.对任意可达标志 m,在 m 的各线程域分量中出现的语句都有可能在 m 下执行,m 只能与这些语句进行组合(否则组合无意义),下文中出现的 M-S 对均为这种组合.在某些可达标志下可能有多个交互发生,发生不同交互后生成不同的可达标志,文中称在 m 下发生 s 交互后到达的可达标志为 m 关于 s 的直接后继,记为 SUCC(m,s).

在并发程序的执行过程中,线程间的一系列交互活动使程序状态发生一系列变迁,将各语句与其相应的程序状态组合,并按语句的执行顺序排列后所形成的 M-S 对序列称作并发程序的一个可执行序列.给定并发程序 CP,记 CP 所有可执行序列的集合为 ES(CP).若某 M-S 对序列为 ES(CP)中某个可执行序列的子序列,则称该 M-S 对序列为 CP 的一个有效序列.在实例 1 中,MSS_1 = (〈m0,start t_0〉,〈m0,S1〉,〈m0,cobegin〉,〈m1,start t_1〉,〈m1,start t_2〉,〈m1,S2〉,〈m2,S3〉,〈m4,coend〉,〈m5,S4〉)为一可执行序列,(〈m1,S3〉,〈m3,S2〉,〈m5,S4〉)为 MSS_1 的子序列,是实例 1 的一个有效序列.

性质 1　给定由 p 个线程 $T=\{t_0,t_1,\cdots,t_{p-1}\}$ 组成的并发程序 CP,设 $MSS=(\Lambda_1,\Lambda_2,\cdots,\Lambda_n)$ 为 CP 的一个 M-S 对序列,MSS 为 CP 的一个有效序列当且仅当:

(1) $\forall 1\le i<n: M(\Lambda_i)=M(\Lambda_{i+1}) \vee PATH^*(M(\Lambda_i),M(\Lambda_{i+1}))$,$S(\Lambda_i)$ 为非交互语句.

$SUCC(M(\Lambda_i),S(\Lambda_i))=M(\Lambda_{i+1}) \vee PATH^*(SUCC(M(\Lambda_i),S(\Lambda_i)),M(\Lambda_{i+1}))$,$S(\Lambda_i)$ 为交互语句;

(2) $\forall t\in T: MSS|_t=(\Delta_1,\Delta_2,\cdots,\Delta_j) \Rightarrow \forall 1\le k<j-1: PATH^*(S(\Delta_k),S(\Delta_{k+1}))$.

$MSS|_t$ 是 MSS 的一个子序列,它去除了 MSS 中所有的语句分量不是 t 中语句的 M-S 对,$PATH^*(S(\Delta_k),S(\Delta_{k+1}))$ 表示在线程的 CFG 图中从节点 $S(\Delta_k)$ 出发存在路径到达 $S(\Delta_{k+1})$,$PATH^*(M(\Lambda_i),M(\Lambda_{i+1}))$ 和 $PATH^*(SUCC(M(\Lambda_i),S(\Lambda_i)),M(\Lambda_{i+1}))$ 分别表示在 tIRG 图中从节点 $M(\Lambda_i)$ 和 $SUCC(M(\Lambda_i),S(\Lambda_i))$ 出发存在路径到达 $M(\Lambda_{i+1})$.

证明:必要性,由可执行序列和有效序列的定义可直接得到(1)和(2).充分性,设 M(MSS)为去处$(M(\Lambda_1),M(\Lambda_2),\cdots,M(\Lambda_n))$中重复出现的状态分量后形成的可达标志序列,由 MSS 满足(1),可在 CP 的 tIRG 图中可找到一个路径(即可达标志序列)满足 M(MSS)为该序列的子序列,再根据(2)和各线程的 CFG 图可构造一个可执行序列,使 MSS 为其子序列.

4.2　基于 tIRG 图的并发程序依赖图

可达图将并发程序的并行流进行了顺序化处理,在分析线程间的数据依赖时,我们不需再区分一般数据依赖和干扰数据依赖,因此本文将并发程序中的依赖分为控制依赖和数据依赖.

设 $MSS=(\Lambda_1,\Lambda_2,\cdots,\Lambda_n)$ 为并发程序 CP 一个可执行序列:若 CP 在沿 MSS 执行过程中,$S(\Lambda_i)$,$S(\Lambda_j)$ 为 CP 中同一线程的两个语句$(1\le i,j\le n)$,$S(\Lambda_j)$的执行与否取决于 $S(\Lambda_i)$的执行,则称 Λ_j 控制依赖于 Λ_i,记为 $CD(\Lambda_i,\Lambda_j)$;若 CP 在沿 MSS 执行过程中 $S(\Lambda_j)$ 使用到 $S(\Lambda_i)$所定义的变量$(1\le i,j\le n)$,则称 Λ_j 数据依赖于 Λ_i,记为 $DD(\Lambda_i,\Lambda_j)$.

由于并发程序中交互语句的先后执行顺序确定后,并发程序的执行是确定的,各交互语句只能在某特定的程序状态下执行,但非交互语句可在多个程序状态下执行.对并发程序的某次执行来说,可能有多个完全等价的可执行序列表示,实际依赖分析只需选择其中一个即可.如实例 2 中非交互语句 S5 可在 m2 和 m5 状态下执行,MSS_2 = (〈m0,start t_0〉,〈m0,S1〉,〈m0,S2〉,〈m0,cobegin〉,〈m1,S3〉,〈m1,S4〉,~~〈m2,S5〉~~,〈m2,S7〉,〈m2,S8〉,〈m5,S6〉,〈m6,coend〉)和 MSS_3 = (〈m0,start t_0〉,〈m0,S1〉,〈m0,S2〉,〈m0,cobegin〉,〈m1,S3〉,〈m1,S4〉,〈m2,S7〉,〈m2,S8〉,~~〈m5,S5〉~~,〈m5,S6〉,〈m6,coend〉)是等价的.为简化分析,文中选择非交互语句与所在线程域中的交互语句在同一状态下的可执行序列,下文只考虑这种可执行序列,上例中仅选择 MSS_3 进行分析.沿 tIRG 图由前至后,通过分析源程序的控制语句和同步语句,并与相应状态组合,就可直接获得控制依赖.数据依赖分析较为复杂,需对每个 M-S 对 Λ 定义 In(Λ)和 Out(Λ),分别记录到达 Λ 和离开 Λ 时各变量定义的位置,对可达标志 m 定义 In(m)和 Out(m,L)分别记录到达 m 和 m 发生 L 交互后的变量定义的位置.算法 1 给出了上述集合的计算方法.对 M-S 对 Λ,如果存在变量 v 和 M-S 对 Λ' 使得 $v\in Ref(S(\Lambda)) \wedge (\Lambda',v)\in In(\Lambda)$ 成立($Ref(S(\Lambda))$表示 $S(\Lambda)$中出现的使用性变量集合),则 Λ 数据依赖于 Λ'.

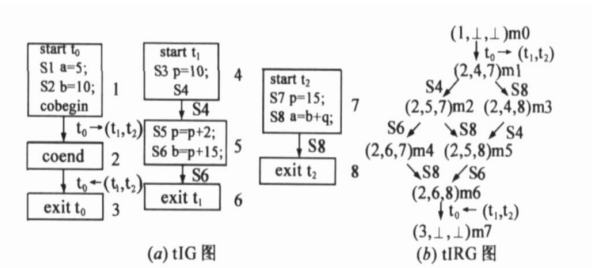


图 4　实例 2 的 tIG 图和 tIRG 图

初始化:
1. 令 tIRG 图中的每个可达标志和 M-S 对的变量定义的输入和输出集为空集,令 W = Φ,W = W ∪{ m_s} ; // 为初始可达标志
算法:
2. while W ≠Φ do
3. 从 W 中取出下一个元素 m;
4. for each(m , m') ∈E_R do
设从 $m \rightarrow m$ 发生 L 交互,L 与第 $k_1, k_2, \cdots, k_r$ 个线程域相关,
5. for each $n \in \{ m[k_1], m[k_2], \cdots, m[k_r] \}$ do
6. In(〈m , I_n〉) = In(m) ;// I_n 为 n 的入口语句
7. 沿与 n 对应的子图计算 Out(〈m , O_n〉)
// O_n 为 n 中与 L 相关的出口语句
8. Out(m , L) = Out(m , L) ∪Out(〈m , O_n〉) ;
end for
9. if (Out(m , L) ⊄In(m'))
10. In(m') = In(m') ∪Out(m , L) ;
11. W = W ∪{ m} ;
end if
end for
end while

算法 1 基于 tIRG 图的并发程序全局数据依赖分析算法

定义 3 设并发程序 CP 的 tIRG 图为〈M , E_R , m_s , M_f , L_R〉,则 CP 基于 tIRG 图的依赖图 (MSDG) 为四元组〈M , S , MS , E_D〉,其中:M 为 CP 的可达标志集,S 为 CP 的语句集,节点集 MS ⊆ M ×S,边集 E_D = { (Λ_i , Λ_j) | dep (Λ_i , Λ_j) , Λ_i , Λ_j ∈MS , dep ∈{ CD , DD}}.

性质 2 给定并发程序 CP 的 MSDG 图〈M , S , MS , E_D〉,Λ , Λ' ∈MS,若 (Λ , Λ') ∈E_D,则

当 S(Λ) 为非交互语句时, (M(Λ) = M(Λ') ∨PATH * (M (Λ) , M(Λ')) ;

当 S(Λ) 为交互语句时, SUCC (M(Λ) , S(Λ)) = M(Λ') ∨ PATH * (SUCC(M(Λ) , S(Λ)) , M(Λ')).

设 MSS = (Λ_1 , Λ_2 , ⋯, Λ_n) 为并发程序 CP 的一个 M-S 对序列, ∀1 ≤i < n :Λ_{i+1} 依赖于 Λ_i ,当存在一个 CP 的可执行序列 MSS′,满足在 MSS′上 MSS 中依赖关系可发生时,称 Λ_n 可传递依赖于 Λ_1.

定理 1 给定并发程序 CP,其 MSDG 图中依赖关系具有可传递性.

证明:由于 MSDG 图中依赖分析是在并行流进行顺序化处理的基础上进行的,因此,如果 MSDG 图中某依赖序列为并发程序可执行序列的子序列(也即有效序列),那么该依赖序列中的依赖均可在这个可执行序列中发生.

设 MSS = (Λ_1 , Λ_2 , ⋯, Λ_n) 为 MSDG 图中任意一个 M-S 对依赖序列,下面证明 MSS 为 CP 的一个有效序列. 由于 MSS = (Λ_1 , Λ_2 , ⋯, Λ_n) 为一依赖序列,由性质 2 可得, (Λ_1 , Λ_2 , ⋯, Λ_n) 满足性质 1 中的第一条. 下面证明 (Λ_1 , Λ_2 , ⋯, Λ_n) 满足性质 1 中的第二条.

设 t 为 CP 中的任意一个线程,当 $MSS|_t$ 不为空时, $MSS|_t$ = (Δ_1 , Δ_2 , ⋯, Δ_j) 时, ∀1 ≤i < j - 1 :

当 Δ_i , Δ_{i+1} 在 MSS 中相邻,即在 MSS 中 Δ_{i+1} 依赖于 Δ_i 时,由于 S(Δ_i) 和 S(Δ_{i+1}) 为同一线程 t 中语句,由数据依赖和控制依赖的定义可得, PATH * (S(Δ_i) , S(Δ_{i+1})) ;

当 Δ_i , Δ_{i+1} 在 MSS 中不相邻,即在 MSS 中 Δ_{i+1} 不依赖于 Δ_i 时,设在 MSS 中从 Δ_i 开始到 Δ_{i+1} 终止的依赖序列为 (Δ_i , Λ_u , ⋯, Λ_{u+k} , Δ_{i+1}) ,由于 S(Λ_u) 和 S(Λ_{u+k}) 不为线程 t 中语句,因此 S(Δ_i) , S(Λ_u) , S(Λ_{u+k}) , S(Δ_{i+1}) 一定为交互语句,由性质 2 可得, PATH * (SUCC (M(Δ_i) , S(Δ_i)) , M(Δ_{i+1})). 依次取 (M(Δ_i) , M(Λ_u) , ⋯, M(Λ_{u+k}) , M(Δ_{i+1})) 的线程 t 所对应的分量,在线程 t 的 tIG 图中,可构造一个从 M(Δ_i) 的 t 分量起,沿 S(Δ_i) 标签到达 M(Δ_{i+1}) 的 t 分量的路径,由线程域的构造可知, PATH * (S(Δ_i) , S(Δ_{i+1})).

MSS 为 CP 的一个有效序列, Λ_n 可传递依赖于 Λ_1 , MSDG 图中依赖关系具有可传递性.

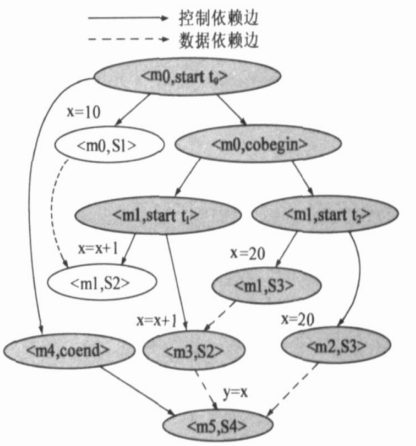


图 5 实例 1 的 MSDG 图

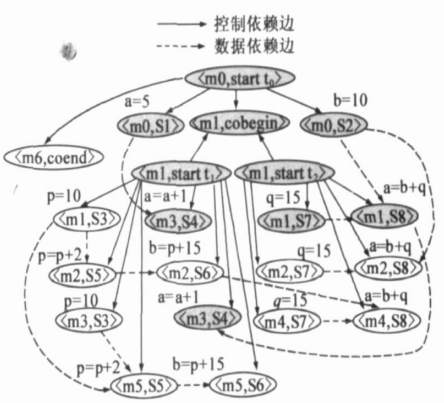


图 6 实例 2 的 MSDG 图

5 基于 MSDG 图的并发程序切片

与传统的以语句为节点的并发程序依赖图相比,MSDG 图中的依赖关系更为精确,且具有可传递性,可广泛应用于高精度的并发程序分析理解、切片计算、程序调试以及度量等软件工程活动中. 限于篇幅,下面以程序切片计算为例说明其应用.

程序切片是一种重要的程序分析理解技术,用于从原有

程序中抽取对特定语句有影响的程序成份以构成新程序,通过分析这种简短的新程序(称为程序切片)达到简化原程序分析的目的[2~9].给定语句 s,分别以所有以 s 为语句分量的 M-S 对为始点遍历 MSDG 图,由所访问 M-S 对的语句分量组成的集合即为 s 的切片.采用本文方法计算实例 1 中 S4 的切片为{S4,S2,S3,cobegin,coend},不包含 S1;计算实例 2 中 S4 的切片为{S4,S1,S2,S7,S8,cobegin},不包含 S6.目前,文献[2~7]中的方法仅解决了部分第二类不可传递问题,文献[8,9]解决了第二类不可传递问题,但未能解决第一类不可传递问题,而本文方法可解决两类不可传递问题,因此,所计算切片的精度最高.

设并发程序 CP 有 n 个语句,p 个线程,c 个交互语句,每个交互语句可生成一个线程域,CP 的线程域约为 c 个,可达标志数最多为 $(c/p)^p$,MSDG 图节点数最多为 $O(n(c/p)^p)$,基于 MSDG 图切片的最坏复杂度为 $O(n^2(c/p)^{2p})$.文献[8,9]中切片时间复杂度为 $O(n^2(n/p)^{2p})$,一般地,$c \ll n$,因此,在最坏情况下本文方法的效率较文献[8,9]中方法高.

6　结束语

针对共享变量通信机制,本文在提出线程交互可达图的基础上,从全局分析了并发程序中的依赖关系,构建了一种精确的、具有传递性的并发程序依赖图(MSDG),MSDG 图可广泛应用于并发程序的高精度依赖分析.在今后的工作中,我们将通过对 tIRG 图的约简研究来进一步提高分析效率,同时在此基础上开展并发程序度量及测试等方面的研究.

参考文献:

[1] Ferrante J, Ottenstein K J, Warren J D. The program dependence graph and its use in optimization[J]. ACM Trans on Programming Languages and Systems, 1987, 9(3): 319 - 349.

[2] Chen Z Q, Xu B W, Zhao J J. An overview of methods for dependence analysis of concurrent programs[J]. ACM SIGPLAN Notices, 2002, 37(8): 45 - 52.

[3] Cheng J. Task dependence nets for concurrent systems with Ada 95 and its applications[A]. ACM TRI-Ada International Conference[C]. St Louis, Missouri: ACM Press, 1997. 67 - 78.

[4] Zhao J J, Li B X. Dependence based representation for concurrent Java programs and it's application to slicing[A]. International Symposium on Future Software Technology[C]. Japan: Software Engineers Association, 2004, 105 - 112.

[5] Ranganath V P, Hatcliff J. Pruning the Detection of Interference and Ready Dependence for Slicing Concurrent Java Programs [R]. Kansas: Computing and Information Sciences Department, Kansas State University, 2005.

[6] 张晶,金成植. 基于控制流的多线程程序的静态切片算法[J]. 吉林大学学报, 2003, 41(4): 481 - 486.
Zhang Jing, Jin Chengzhi. Control-flow-based static slicing algorithm of threaded programs[J]. Journal of JiLin University, 2003, 41(4): 481 - 486. (in Chinese)

[7] 陈振强,徐宝文. 一种并发程序依赖性分析方法[J]. 计算机研究与发展, 2002, 39(2): 159 - 164.
Chen Zhenqiang, Xu Baowen. An approach for analyzing dependence of concurrent programs[J]. Journal of Computer Research and Development, 2002, 39(2): 159 - 164. (in Chinese)

[8] Krinke, J. Context-sensitive slicing of concurrent programs[A]. ACM SIGSOFT Symposium on Foundations of Software Engineering 2003 held jointly with 9th European Software Engineering Conference[C]. St Louis, Missouri: ACM press, 2003. 178 - 187.

[9] Nanda M G. Slicing Concurrent Java Programs: Issues and Solutions[D]. Bombay: Department of Computer Science and Engineering, Indian Institute of Technology, 2001.

[10] Qi X F, Xu B W. Dependence analysis of concurrent programs based on reachability graph and its applications[A]. International Conference on Computational Science [C]. Berlin: Springer Verlag, 2004. LNCS3036: 405 - 408.

[11] Pezze M, Taylor R N, Young M. Graph models for reachability analysis of concurrent programs[J]. ACM Trans on Software Engineering and Methodology, 1995, 4(2): 171 - 213.

作者简介:

戚晓芳　女,1972 年生于江苏海安,博士生,副教授,主要研究方向为并发程序的分析、理解与测试. E-mail: xfqi@seu.edu.cn

徐宝文　男,1961 年生于江苏东台,教授,博士生导师,主要研究方向为程序设计语言和软件工程等.

周晓宇　男,1972 年生于江苏淮安,博士生,副教授,主要研究方向为程序设计语言和软件工程等.