扩展有限状态机 EFSM 的后向切片*

缪 力, 张大方+

(湖南大学 计算机与通信学院,湖南 长沙 410082)

Computing Backward Slice of EFSMs

MIAO Li, ZHANG Da-Fang+

(College of Computer and Communication, Hu'nan University, Changsha 410082, China)

+ Corresponding author: Phn: +86-731-8821980, Fax:+86-731-8821980, E-mail: jt_zdf@hnu.cn

Received 2003-12-08; Accepted 2004-07-06

Miao L, Zhang DF. Computing backward slice of EFSMs. *Journal of Software*, 2004,15(Suppl.):169~178.

Abstract: Slicing is a well-known reduction technique. Most of the research on slicing is code-based. There has been limited research on specification-based slicing and model-based slicing. EFSM is a very important specification model, but a practical EFSM model is often large and complex and is hard to understand and modify. EFSM-based slicing is very useful to test and analysis EFSM models. The dependence analysis based EFSM dependence graph is much more than a graph reachability problem, so that traverse algorithms based marking visited nodes can not to be used in EFSM dependence graphs. In this paper, we discuss dependence analysis in EFSMs, and define a dependence transform function and a reverse dependence transfer function to describe dependence transform formally. Based the analysis and definition, a backward slicing algorithm is given to find all transitions which may affect an interesting transition.

Key words: EFSM; dependence analysis; software specification; program slicing

摘 要: 切片是一种重要的约减技术,基于代码的程序切片已经得到广泛的研究和大量的应用,但基于规格和模型的切片研究不多.EFSM 是一种重要的规格模型,将切片技术引入 EFSM 对于分析和测试基于 EFSM 的软件模型具有重要的作用.由于一个实际的 EFSM 相当复杂,导致对 EFSM 模型的分析和修改非常困难.EFSM 的切片对于测试和分析 EFSM 模型具有重要意义.由于 EFSM 依赖关系的特殊性,依赖图的可达性问题对于 EFSM 依赖图中并不是一个简单的图可达性问题,而且现有的基于标志已访问节点的遍历算法不适于解决该问题.对 EFSM 的依赖关系的传递性进行了详细的讨论和分析,定义了一个递归的依赖传递函数来描述变迁的影响如何通过依赖关系传递,而计算后向切片则可以通过逆依赖传递函数来描述,最后给出一个基于变迁的 EFSM 后向切片算法.

关键词: 扩展有限状态机 EFSM;依赖性分析;软件规格;程序切片

* Supported by the National Natural Science Foundation of China under Grant No.60373000 (国家自然科学基金)

作者简介: 缪力(1972－),男,湖南长沙人,博士生,主要研究方向为软件测试与软件可靠性;张大方(1959－),男,教授,博士生导师,主要研究方向为容错计算,软件测试与软件可靠性,网络测试.

1 引 言

形式化方法是减少软件测试的耗费,提高软件的可靠性的重要手段,扩展有限状态机 EFSM 是一种得到普遍重视和应用的形式化的软件规格模型,基于 EFSM 的测试方法已经得到了大量的研究[1,2,4,5,8].对于一个大规模的软件系统,人们往往需要针对系统模型中某些特性进行分析和测试,由于一个实际的 EFSM 相当复杂,直接对 EFSM 模型的分析非常困难,而根据某些特定性质(本文指变迁)简化 EFSM 模型,在 EFSM 模型中找到所有影响这些特定变迁的其他变迁,对 EFSM 模型的分析和测试有重要意义.文献[1,2]提出一种能基于对 EFSM 依赖性分析的约减方法,有效地约减了对 EFSM 模型中特定的性质生成的测试集,文献[3]提出了一种对 EFSM 模型进行切片分析的方法,将程序切片技术推广到 EFSM 模型.

切片是一种重要的约减方法,基于代码的程序切片已经得到了广泛的研究和应用[9~12],但基于软件模型的切片研究不多[3,6,7],其中文献[6]针对层次状态机计算切片,其控制依赖表示的是事件的因果关系,文献[7]计算 Z 规格的切片,其中数据依赖仅仅考虑谓词依赖关系,而且文献[6,7]都不是通过构造依赖图计算切片,只有文献[3]提出的切片技术是针对 EFSM 模型的,并将类似于程序切片的数据依赖和控制依赖关系引入了 EFSM,通过 EFSM 依赖图计算切片.依赖性分析是切片技术的基础,对软件模型(本文针对 EFSM)的依赖性分析和对程序的依赖性分析有很大不同.在计算切片中,一个重要的问题是依赖关系的传递性问题,也就是依赖图的连通性问题.由于程序中的依赖关系是传递的,我们可以把程序的依赖关系画成程序依赖图,而计算切片的问题就成了图的可达性问题,程序切片可以通过基于标志已访问节点的遍历算法遍历依赖图得到.对于 EFSM 依赖图来说,首先,EFSM 在一个变迁中可以定义多个变量,而变迁的数据依赖关系在依赖图中仅仅通过一条加标记的边表示;其次,EFSM 的控制依赖表示的是变迁间的依赖关系,不像程序的控制依赖表示的是谓词条件语句与辖制域内的语句的关系;这导致依赖关系的传递性问题在 EFSM 依赖图中并不是一个简单的图可达性问题,进一步的,由于 EFSM 依赖关系的特殊性,一般的基于标志已访问节点的遍历的方法存在问题,简单的说,两个变迁间 t, t'可能存在多种依赖关系,变迁 t(准确的说是该变迁的"影响")通过不同的依赖关系到达变迁 t'可能影响该变迁是否能通过某种依赖关系到达另一个变迁 t''.并发程序的切片也不是一个图的可达问题[10,11],但仍然可以通过基于标志已访问节点的遍历算法解决.在这一点上,EFSM 的切片也不同于并发程序切片.文献[1,2]基本上都没有谈到这个问题,文献[3]的算法是基于标志已访问节点的遍历的方法.

本文对 EFSM 的依赖性分析进行了详细的讨论和分析,对文献[1~3]给出的依赖关系进行了改进,增加了数据控制依赖关系的定义,因为数据控制依赖具有数据依赖和控制依赖的两重特性,对依赖关系的传递性有重要作用;定义了一个递归的依赖传递函数来描述变迁的影响如何通过依赖关系传递,而计算后向切片则可以通过逆依赖传递函数来描述;最后给出了一个基于变迁的 EFSM 后向切片算法.

2 背景知识和现有算法的问题

定义 2.1.[4] 一个扩展有限状态机(EFSM)是一个对(S,T),其中 S 是状态的集合,T 是状态间变迁的集合,每个变迁 $t\in T$ 是元组(s, i, P, op, o, up, s'),其中

$s, s'\in S$ 是初始状态和终结状态,

$i\in I$ 是输入,I 是输入的集合,D_i是输入向量的集合,一个输入向量的每个元素对应于与 i 相关的一个输入参数,

$o\in O$ 是输出,O 是输出的集合,D_o是输出向量的集合,一个输出向量的每个元素对应于与o相关的一个输出参数,

P, op, 和 up 是定义在输入参数和上下文变量 V 上的函数,

P: $D_i\times D_V\rightarrow$ {True, False}是谓词, D_V 是上下文变量 V 的集合;

op: $D_i\times D_V\rightarrow D_o$是一个输出参数函数;

up: $D_i\times D_V\rightarrow D_V$ 是一个上下文修改函数;

EFSM 的运行方式如下:在任意一个状态,扩展有限状态机接受输入并且计算谓词,为真的谓词决定了在这

个状态和这个输入的情况下可被激活的变迁.通过执行被激活的变迁,机器产生了输出,并且通过该变迁中的上下文修改函数修改上下文,然后到达下一个状态.扩展有限状态机就这样从初始状态经过一步步的变迁到达最终状态.每个变迁都是不可中断的原子操作.

设 t 是一个变迁,我们用 $S_b(t)$表示 t 出发的状态;$S_e(t)$是 t 进入的状态;$use(t)$是在变迁 t 中引用的变量集合,包括谓词(条件)和行为(赋值)以及输出;$def(t)$是在变迁 t 中定义的变量集合,包括行为和输入;例如,在图 1 中,$use(t7)=\{x3,s\}$,$def(t7)=\{x5,s\}$.

数据依赖反映了变迁间通过变量赋值影响的关系,称变迁 t_i 和 t_k 之间存在数据依赖,如果存在一个变量 v,且:(1) $v\in def(t_i)$, (2) $v\in use(t_k)$, (3) 在 EFSM 中存在一条从 t_i 到 t_k 的路径(变迁序列),在该路径上变量 v 不被重定义.例如,在图 1 中,变迁 $t2$ 通过变量 $x3$ 和 $x2$ 对 $t3$ 产生数据依赖,变迁 $t1$ 通过变量 $x1$ 对变迁 $t2$ 产生数据依赖.

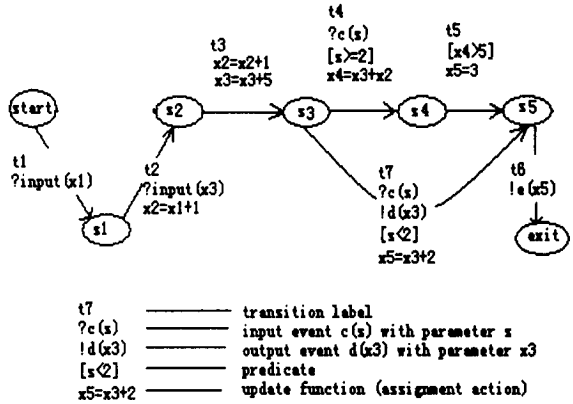

图 1 一个 EFSM 模型

控制依赖反映了变迁间通过激活影响的关系,如果一个变迁是否激活能影响另一个变迁的激活,说明,(1)第一个变迁的激活是下一个变迁激活的必要条件;(2)处于第 1 个变迁的开始状态时,第 1 个变迁可能不被激活,而可能激活另一个变迁并能不通过下一个变迁到达终止状态.为了给出一个准确的定义,我们首先给出逆辖制的概念.

定义 2.2. 逆辖制(post-dominance $\xrightarrow{pdom}$).

设 s 和 s'是两个状态,t 是一条从状态 s 出发的变迁,称状态 s'逆辖制状态 s(记为 $s'\xrightarrow{pdom}s$)当且仅当从状态 s 出发到达终止状态的所有路径都必须经过状态 s';称状态 s'逆辖制变迁 t 当且仅当从状态 s 出发经过变迁 t 到达终止状态的所有路径都必须经过状态 s',将非逆辖制记为 $\xrightarrow{not-pdom}$.

定义 2.3. 控制依赖[1].

设 t_i, t_j 是两个变迁,称变迁 t_i 控制依赖变迁 t_j,当且仅当: (1) $S_b(t_j)\xrightarrow{not-pdom}S_b(t_i)$, (2) 状态 $S_b(t_j)\xrightarrow{pdom}t_i$.例如在图 1 中,$t4$ 对 $t5$ 产生了控制依赖.

一个 EFSM 的依赖图是一个有向图 G,其中图的节点对应于 EFSM 中的变迁,依赖图的边表示变迁间的控制依赖和数据依赖关系,依赖图中任意两个节点间最多有两条直接相连的边,如果这两个节点(变迁)间同时存在数据依赖和控制依赖关系.

图 2 是图 1 中 EFSM 模型的依赖图,其中数据依赖边上的变量标识表示与该数据依赖相关的变量集.

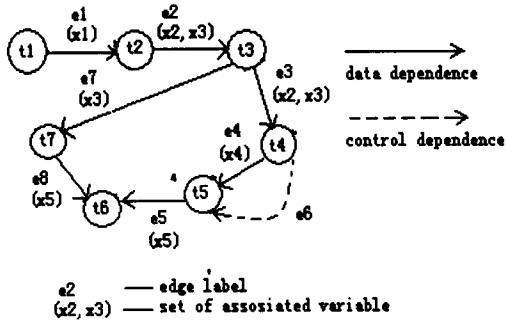

图 2 一个 EFSM 的依赖图

文献[1~3]中最重要的算法就是反向遍历算法,该算法在文献[3]中给出了一个描述,算法的目的在于,对于一个感兴趣的特定变迁,通过从该变迁开始的在依赖图上的反向遍历,找到所有可能影响该变迁的所有其他变迁.类似算法在程序切片中也称计算反向切片算法.本文简称反向遍历算法为遍历算法,因为对于有向图而言,与一般正向遍历相比,反向遍历仅仅是逆箭头而行而已.文献[3]给出的算法是一种基于标志已访问节点的遍历算法,该算法的特点是,对于已经访问过的节点,下次不再访问.

以图 2 为例,我们计算影响 t6 的变迁集合,根据文献[3]的算法,如果变迁 *t*7 首先被访问,则标志 *t*7 为已访问,从 *t*7 开始反向遍历到达 *t*3,标志 *t*3 为已访问,然后从 *t*3 开始反向遍历到达 *t*2,标志 *t*2 为已访问,然而此时 *t*1 是不能到达的,因为 *t*1 不能通过路径 *e*1,*e*2,*e*7,*e*8 影响 *t*6.

然后算法访问 *t*5,标志 *t*5 为已访问,从 *t*5 开始反向遍历到达 *t*4,然后算法访问 *t*4,标志 *t*4 为已访问,从 *t*4 开始反向遍历到达 *t*3,发现 *t*3 已经被访问因而无法继续遍历到达 *t*1,于是 *t*1 没有被计算在能影响 *t*6 的变迁集合内,切片结果为{*t*7,*t*3,*t*2,*t*5,*t*4},但是正确的切片结果显然应是{*t*7,*t*3,*t*2,*t*5,*t*4,*t*1}.

3 EFSM 中的依赖关系的传递性

两个变迁的依赖关系反映了一个变迁对另一个变迁的影响,这种影响分为两种,一是影响变迁是否被激活,例如控制依赖.控制依赖反映了这样一种思想:如果一个变迁 t 是否被激活与另一个变迁 t'有关,这说明,任何可能到达 t 的路径需要激活变迁 t',所以任何可能到达 t 的路径都要经过 t',这就是控制依赖定义中的 $S_b(t) \xrightarrow{pdom} t'$,再就是,变迁 t'需要被激活,所谓需要被激活,一是 t'可能不被激活所以才需要激活,二是 t'不被激活则存在一条从 $S_b(t')$出发不到达 t 的路径,这就是控制依赖定义中的 $S_b(t) \xrightarrow{not-pdom} S_b(t')$.二是影响变迁中变量的值.数据依赖反映了这种影响.有一种数据依赖可以影响变迁的激活,但不是象控制依赖那样通过一个变迁的激活来影响另一个变迁的激活,而是通过一个变迁中定义的变量被另一个变迁激活条件(谓词)中引用来影响.我们将这种数据依赖称为数据控制依赖以区别于普通的数据依赖.

EFSM 的依赖关系的传递性与程序中依赖关系的传递性有很多不同.例如由于在 EFSM 的一个变迁中可以对多个变量赋值,因此变迁间的数据依赖关系不是当然传递的,如图 3 所示.

显然 *t*1 和 *t*2 有数据依赖,*t*2 和 *t*3 有数据依赖,但 *t*1 不能影响 *t*3,*t*1 和 *t*3 之间没有数据依赖关系,即 *t*1 对 *t*2 的数据依赖不能通过 *t*2 对 *t*3 的数据依赖传递.*t*1 对 *t*2 的数据依赖是否传递受 *t*2 到 *t*3 的数据依赖相关变量的约束.

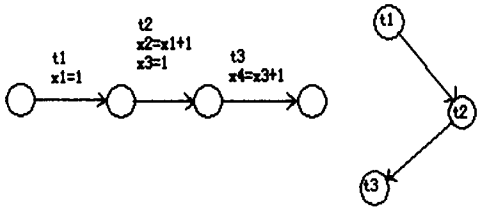

图 3

影响变迁是否激活必然影响变迁中变量的值,所以与变迁谓词相关的数据依赖关系,即本文定义的数据控制依赖关系可以通过数据依赖关系传递,而且不受数据依赖相关变量的约束,如图 4 所示.

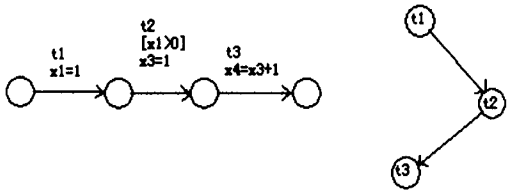

图 4

显然 $t1$ 和 $t2$ 有关于变量 $x1$ 的数据依赖,$t2$ 和 $t3$ 有关于变量 $x3$ 的数据依赖,$t1$ 对 $t2$ 的数据依赖不受 $t2$ 到 $t3$ 的数据依赖相关变量的约束,在图 2 中 $t1$ 对 $t3$ 有影响.

对于 EFSM 依赖图而言,数据依赖和控制依赖的关系也不像在程序依赖图中一样,因为数据依赖关系不能影响变迁的激活,因此数据依赖关系不能通过控制依赖关系传递,如图 5 所示.

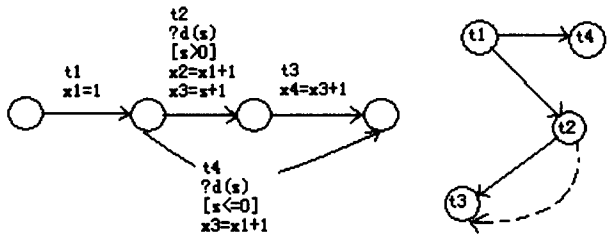

图 5

显然 $t1$ 和 $t2$ 有关于变量 $x1$ 的数据依赖,$t2$ 和 $t3$ 有控制依赖存在,而 $t1$ 不能对 $t3$ 有影响.这说明这种数据依赖不能通过控制依赖传递,但数据控制依赖可以通过控制依赖传递,如图 6 所示.

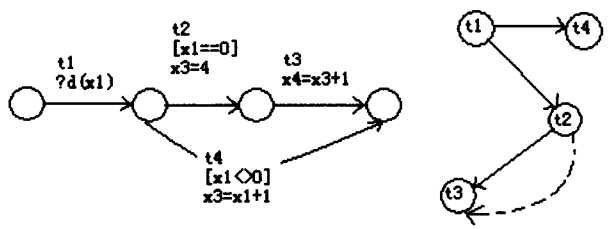

图 6

图中 $t1$ 和 $t2$ 有关于变量 $x1$ 的数据控制依赖,$t2$ 和 $t3$ 有控制依赖存在,这时 $t1$ 能对 $t3$ 有影响.

设 t 是 EFSM 中的一个变迁,我们给出以下定义:$A_use(t)$表示在变迁 t 中行为(赋值)中引用的变量集合,$C_use(t)$表示在变迁 t 中谓词(条件)引用的变量集合,$O_use(t)$表示在变迁 t 的输出中引用的变量集合,$use(t)=A\_use(t)\cup C\_use(t)\cup O\_use(t)$.$A_def(t)$表示在变迁 t 的行为中定义的变量集,$I_def(t)$表示在变迁 t 的输入中定义的变量集,$def(t)=A\_def(t)\cup I\_def(t)$.例如,在图 1 中,$A\_use(t7)=\{x3\}$,$C\_use(t7)=\{s\}$,$O\_use(t7)=\{x3\}$,$A\_def(t7)=\{x5\}$,$I\_def(t7)=\{s\}$.

为了便于对依赖关系进行一致的描述,我们把所有的依赖关系都定义为统一的形式.

定义 3.1. 依赖关系Γ.

对于一个 EFSM M,其中两个变迁的依赖关系可以表达为一个四元组$\Gamma=(TF, TS, V, B)$,TF 是依赖关系中的第一个变迁(First Transition),TS 是依赖关系的第 2 个变迁(Second Transition).

对于一个数据依赖关系,其中 TF 为定义变量集 V 的变迁,TS 为在行为(赋值)和输出事件中引用变量 V 的变迁,V 是与数据依赖相关的变量集,$V=def(TF)\cap(A\_use(TS)\cup O\_use(TS))\neq\varnothing$,$B$ 标识依赖的类型,对于数据依赖,B="*data*".以下谈到的数据依赖如非特指,都用在此的数据依赖的定义.

我们将对变迁中条件谓词的数据依赖视为一种特殊的数据依赖,称为数据控制依赖,其中 TF 为定义变量集 V 的变迁,TS 为在条件谓词中引用变量集 V 的变迁,V 是与数据控制依赖相关的变量集,$V=def(TF)\cap C\_use(TS)\neq\varnothing$,$B$="*data-control*".

对于控制依赖,$V=\varnothing$,B="control",TF 对 TS 产生控制依赖.

例如,在图 1 中,$t4$ 到 $t5$ 的数据控制依赖关系描述为($t4$,$t5$,{$x4$},"*data-control*"),$t4$ 到 $t5$ 的控制依赖关系描述为($t4$,$t5$,∅,"*control*"),$t2$ 到 $t3$ 的数据依赖关系描述为($t2$,$t3$,{$x2$,$x3$},"*data*").

根据我们的定义,依赖图的任意两个节点间最多有 3 条直接相连的边.

4 依赖传递函数

一个变迁的影响能到达另一个变迁的必要条件是,该变迁在依赖图中存在至少一条路径到达另一个变迁,依赖图中的路径是由一系列依赖关系构成,本文称为依赖链.

定义 4.1. 依赖链.

一个 EFSM M 中的依赖链$\Gamma''=(\Gamma_1,\Gamma_2,\ldots,\Gamma_n)$是一个有各种依赖关系构成的依赖序列,$\forall\Gamma_i\in\Gamma(M)$且有$\forall i<n$: $TS(\Gamma_i)=TF(\Gamma_{i+1})$.

一条依赖链相当于 EFSM 的依赖图中的一条路径.

在数据依赖中,由于变迁间变量的定义和引用的不同方式可以造成的不同影响: (1) 如果是在行为中引用变量,那么引用的变量被用于定义其他的变量,我们用 $A_def(t,W)$表示在变迁 t 的行为中用变量集 W 中的元素定义的变量集,例如,在图 1 中,$A\_def(t7,\{x3\})=\{x5\}$; (2) 如果是在输出中引用变量,那么引用的变量不被用于定义其他的变量,仅仅只是对这个变迁产生影响,我们用 $O_def(t,W)$表示在变迁 t 的输出中引用变量集 W 中的元素产生的影响,定义 $O\_def(t,W)=\begin{cases}\Delta,\text{如果}W\cap O\_use(t)\neq\varnothing\\ \varnothing,\text{否则}\end{cases}$,Δ表示这样一种想法,如果用集合表示影响,则∅表示没有影响,集合越大表示影响越大,而Δ表示有影响,却是最弱的一种影响,如同一个只有空元素的集合{∅},所以对任意非空变量集 $W\neq\Delta$,有$\Delta\cap W=\varnothing$,且$\Delta\supset\varnothing$.例如,在图 1 中,$O\_def(t6,\{x5\})=\Delta$; (3) 如果是在谓词中引用变量,那么引用的变量对这个变迁的激活产生影响,我们用 $C_def(t, W)$表示在变迁 t 的谓词中引用变量集 W 中的元素产生的影响,定义 $C\_def(t,W)=\begin{cases}\nabla,\text{如果}W\cap C\_use(t)\neq\varnothing\\ \varnothing,\text{否则}\end{cases}$,其中 ∇表示激活定义,因为影响变迁是否激活必然影响变迁中所有定义的变量,所以对任意变量集 W 有$\nabla\supseteq W$.例如,在图 1 中,$C\_def(t5,\{x4\})=\nabla$;

定义$\Delta\cup\nabla=\nabla$,$\Delta\cap\nabla=\Delta$.$def(t,W)=A\_def(t,W)\cup O\_def(t,W)\cup C\_def(t,W)$.

依赖关系必然通过依赖链传递,但能传递的依赖关系受到依赖关系的类型及相关变量集的约束,我们引入依赖传递函数的概念描述这个问题.

定义 4.2. 依赖传递函数 *Tdep*.

依赖传递函数反映了依赖关系通过依赖链传递的可传递性.

对于一个依赖链$\Gamma^k=(\Gamma_1, \Gamma_2, \ldots, \Gamma_k)$,若 $k=0$,有

$TDep(\Gamma^0)=\nabla$;

若 $k>0$,则有

$$TDep(\Gamma^k)=\begin{cases} def(TS(\Gamma_k), V(\Gamma_k)\cap TDep(\Gamma^{k-1})), \text{如果} B(\Gamma_k)\neq \text{"}control\text{"} \text{并且} TDep(\Gamma^{k-1})\neq\varnothing \\ \nabla, \text{如果} B(\Gamma_k)=\text{"}control\text{"} \text{并且} TDep(\Gamma^{k-1})=\nabla \\ \varnothing, \text{否则} \end{cases}$$

对于一个 EFSM M 中的依赖链$\Gamma^n=(\Gamma_1, \Gamma_2, \ldots, \Gamma_n)$,如果依赖传递函数 $TDep(\Gamma^n)\neq\varnothing$,说明变迁 $TF(\Gamma_1)$通过依赖链Γ^n对变迁 $TS(\Gamma_n)$有影响.

以图 1 中的一条依赖链$\Gamma^3=(\Gamma_1, \Gamma_2, \Gamma_3)$ = ((*t*3,*t*4,{*x*2,*x*3},"data"), (*t*4,*t*5,{*x*4},"*data-control*"), (*t*5,*t*6,{*x*5},"*data*")) 为例 :$TDep(\Gamma^1)=def(TS(\Gamma_1), V(\Gamma_1))=\{x4\}$, 因为 $V(\Gamma_2)\cap TDep(\Gamma^1)=\{x4\}\cap\{x4\}\neq\varnothing$, 所以, $TDep(\Gamma^2)=\nabla$, $TDep(\Gamma^3)= def(TS(\Gamma_3), V(\Gamma_3)\cap TDep(\Gamma^2))=def(t6, \{x5\}\cap\nabla) = def(t6, \{x5\})=\Delta\neq\varnothing$.

我们称变迁 *t* 可以通过依赖关系影响变迁 *t'*,记为 $t \xrightarrow{dep-affect} t'$,即指存在一条依赖链$\Gamma=(\Gamma_0, \Gamma_1, \ldots, \Gamma_n)$,$TDep(\Gamma^n)\neq\varnothing$,其中 $TF(\Gamma_0)=t, TS(\Gamma_n)=t'$.

5 EFSM 的后向切片算法

对某个特定的变迁,通过传递的依赖链,一个变迁就可能对另一个变迁的激活或行为产生影响,我们将切片技术引入 EFSM,通过变迁的后向切片得到可能会影响到该变迁的变迁集合.

定义 5.1. 变迁的后向切片.

对一个 EFSM M,给定切片准则 *t* 为 *M* 中的一个变迁,则变迁 *t* 的后向切片 *BSlicing*(*t*)为所有可以通过至少一条依赖链影响 *t* 的变迁的集合,$BSlicing(t)=\{t' \mid t' \xrightarrow{dep-affect} t\}$.

由于计算后向切片一般都是基于兴趣点(特定变迁)向后遍历,其过程与影响通过依赖关系正向传递的过程相反,我们可以将其视为对通过依赖关系传递的影响求逆,并称这种反过来的影响称为逆影响.

定义 5.2. 逆传递链.

一个 EFSM 中的依赖链$\Gamma^n=(\Gamma_1, \Gamma_2, \ldots, \Gamma_n)$的逆依赖链 $R^n=(R_1, R_2, \ldots, R_n)$,其中 $R_k=\Gamma_{n-k+1}$, $n\geq k\geq 1$.

以图 1 中的一条依赖链$\Gamma^3=(\Gamma_1,\Gamma_2,\Gamma_3)=(e2,e7,e8)=((t2,t3,\{x2,x3\},\text{"data"}),(t3,t7,\{x3\},\text{"}data\text{"}),(t7,t6,\{x5\},\text{"}data\text{"}))$ 为例: 对应的逆依赖链为 $R^3=(R_1,R_2,R_3)=(e8,e7,e2)=((t7,t6,\{x5\},\text{"}data\text{"}),(t3,t7,\{x3\},\text{"}data\text{"}),(t2,t3,\{x2,x3\},\text{"}data\text{"}))$.

设 *t* 是 EFSM 中的一个变迁,我们给出以下定义:*A_use*(*t*, *W*)表示在变迁 *t* 的行为中为了定义变量集 *W* 而使用的变量集.例如,在图 1 中,*A_use*(*t*7, {*x*5}) ={*x*3};考虑到在输入定义变量和用常量定义变量的情况,我们定义

$$use(t,W)=\begin{cases} A\_use(t,W)\cup\Delta, \text{如果} W\cap def(t)\neq\varnothing \\ \varnothing, \text{否则} \end{cases}$$,例如,在图 1 中,*use*(*t*2, {*x*3}) =Δ.

定义 5.3. 逆依赖传递函数 *RTDep*.

对于一个逆依赖链 $R^k=(R_1, R_2, \ldots, R_k)$,若 $k=0$,有

$RTDep(\Gamma^0)=\nabla$;

若 $k>0$,则有

$$RTDep(R^k)=\begin{cases} use(TF(R_k),V(R_k)\cap RTDep(R^{k-1})),\ \text{如果}B(R_k)=\text{"}data\text{"}\text{并且}RTDep(R^{k-1})\neq\varnothing \\ use(TF(R_k),V(R_k)),\ \text{如果}B(R_k)=\text{"}data-control\text{"}\text{并且}RTDep(R^{k-1})\neq\varnothing \\ \Delta,\ \text{如果}B(R_k)=\text{"}control\text{"}\text{并且}RTDep(R^{k-1})\neq\varnothing \\ \varnothing,\ \text{否则} \end{cases}$$

对于一个 EFSM M 中的逆依赖链 $R^n=(R_1, R_2, \ldots, R_n)$,如果逆依赖传递函数 $RTDep(R^n)=\varnothing$,说明变迁 $TS(R_n)$ 通过逆依赖链 R^n 对变迁 $TF(R_1)$ 没有逆影响.

以图 1 中的一条逆依赖链 $R^3=(R_1, R_2, R_3)=(e8, e7, e2)=((t7, t6, \{x5\}, \text{"}data\text{"}), (t3, t7, \{x3\}, \text{"}data\text{"}), (t2, t3, \{x2,x3\}, \text{"}data\text{"}))$ 为例:$RTDep(R^1)=use(TF(R_1),V(R_1))=\{x3\}$, $RTDep(R^2)=use(TF(R_2),V(R_2)\cap RTDep(R^1))=use(t3, \{x3\}\cap\{x3\})=\{x3\}$,$RTDep(R^3)=use(TF(R_3), V(R_3)\cap RTDep(R^2))=use(t2, \{x3\}\cap\{x3\})=\Delta\neq\varnothing$.

通过定义逆依赖传递函数,我们便可以直接用逆依赖传递函数描述后向切片算法.

基于标志已访问节点的算法的问题在于,从不同的路径到达同一个节点或同一条边代表了不同的逆依赖链,得到不同的 *RTDep* 计算结果.所以我们需要重复访问某些节点和边.从逆依赖传递函数 *RTDep* 的定义看到,只有数据依赖关系(边)可能存在不同路径有不同信息的情况,所以对于访问过的控制依赖关系(边)和数据控制依赖关系(边),可以安全地标识已访问,下次不必再访问.对于数据依赖关系(边),我们只能标识访问的变量集,如果数据依赖的相关变量集未被全部标识为已访问,就不能标识数据依赖边已访问.

后向切片算法.

输入: EFSM 依赖图 *G*,切片准则变迁 *t*

输出: 影响 t 的变迁集合 *BS*

```
1   初始化变量集 BS=∅,W=∅,标志所有的Γ(边)未被访问
2   for 所有直接到达 t 的依赖关系Γ do
3    BS=BS∪TF(Γ)
4    标志Γ已访问;
5    if B(Γ)=="control" then //Γ是控制依赖
6     W=Δ
7    else
8     W=use(TF(Γ), V(Γ))
9    endif
10   SearchBack(TF(Γ), W)
11  endfor

    Procedure SearchBack(t_b, V_b)
12  BZ=false      //BZ 是传递标志
13  V_u = ∅       // V_u是依赖关系传递的参数
14  if V_b ≠ ∅ then   // RTDep ≠ ∅
15   for 所有直接到达 t_b 的依赖关系的变迁Γ and Γ未被访问 do
16    if B(Γ)== "data" then     //Γ是数据依赖
17     V_u = use(TF(Γ), V_b ∩ V(Γ)) //计算依赖传递性相关的变量集
18     if V_u ≠ ∅ then   //说明能通过数据依赖传递
19      V(Γ) = V(Γ) − (V_b ∩ V(Γ)) //移去相关变量集
20      BZ= true
21     endif
22     if V(Γ) ==∅ then   //当数据依赖的相关变量集为空时标志Γ已访问
```

```
23        标志Γ已访问
24       endif
25      elseif B(Γ)== "data-control" then   //Γ是数据控制依赖
26       标志Γ已访问;
27       V_u= use(t_b, V(Γ)) //数据控制依赖的相关变量集
28       BZ= true
29      elseif B(Γ)== "control" then   //Γ是控制依赖
30       标志Γ已访问;
31       V_u = Δ
32       BZ= true
33      endif
34      if BZ then //可以传递
35       BS = BS∪ TF(Γ)
36       SearchBack(TF(Γ), V_u)   //从 t_u 开始向后遍历
37      endif
38     endfor
39    endif
  end SearchBack
```

用后向切片算法对图 2 的计算过程可由读者自行验证,算法可以得到正确的切片结果:{*t*7,*t*3,*t*2,*t*5,*t*4,*t*1}.

对于一个 EFSM 的依赖图 *G*,若其中包括 *TC* 条控制依赖边,*TD* 条数据依赖边,*TCD* 条数据控制依赖边,其中任意一个数据依赖边最多有 *VD* 个变量标志,则切片算法的时间复杂度为 *TC*+*TCD*+*TD*×*VD*.

6 结 论

切片是一种重要的约减技术,将切片技术引入EFSM对于分析和测试基于EFSM的软件模型具有重要的作用.由于 EFSM 依赖关系的特殊性,依赖图的可达性问题在 EFSM 依赖图中并不是一个简单的图可达性问题,与并发程序切片相比,尽管并发程序的依赖图也不是直接可达的,但仍然可以通过基于标志已访问节点的遍历算法解决,而对于 EFSM 依赖图,现有的基于标志已访问节点的遍历算法不适于解决该问题.本文对 EFSM 的依赖关系的传递性进行了讨论和分析,定义了一个递归的依赖传递函数来描述变迁的影响如何通过依赖关系传递,而计算后向切片则可以通过逆依赖传递函数来描述,最后给出一个基于变迁的 EFSM 后向切片算法.

References:

[1] Korel B, Tahat L, Bader A. Model based regression test reduction using dependence analysis. In: Proc. of the IEEE Int'l Conf. on Software Maintenance (ICSM 2002). 2002. 214~223.

[2] Vaysburg B, Tahat L., Korel B. Dependence analysis in reduction of requirement based test suites. In: Proc. of the ACM Int'l Symp. on Software Testing and Analysis (ISSTA 2002). 2002. 107~ 111.

[3] Korel B, Tahat L, Bader A. Slicing of state-based models. In: Proc. of the IEEE Int'l Conf. on Software Maintenance (ICSM 2003). 2003. 34~43.

[4] Alexandre Petrenko, Sergiy Boroday, Roland Groz. Confirming configurations in EFSM. FORTE. 1999. 5~24.

[5] Lee D, Yannakakis M. Principles and methods of testing finite state machines, a survey. Proceedings of the IEEE, 1996,84(8): 1090~1123.

[6] Heimdahl M, Whalen M. Reduction and slicing of hierarchical state machines. ACM SIGSOFT Software Engineering Notes, 1997,22(6):450~467.

[7] Chang J, Richardson D. Static and dynamic specification slicing. In: Proc. of the 4th Irvine Software Symposium. 1994.

[8] WANG JG, WU JP. An extended finite state machine based generation method of test suite. Journal of Software, 2001,12(8):1197~1204 (in Chinese with English abstract).

[9] Tip F. A survey of program slicing techniques. Journal of Programming Languages, Sept.1995,3(3):121~189.

[10] Krinke,J. Static slicing of threaded programs. In Program Analysis for software tools and engineering (PASTE 98). ACMSOFT, 1998. 35~42.

[11] Nanda MG , Ramesh S. Slicing Concurrent programs. ISSTA'00, 2000. 180~190.

[12] Xu BW, Chen ZQ, Zhou XY. Slicing object-oriented Ada95 programs based on dependence analysis. Journal of Software, 2001,12(Suppl.):208~213 (in Chinese with English abstract).

附中文参考文献:

[8] 王建国,吴建平.基于扩展有限状态机的协议测试集生成研究.软件学报,2001,12(8):1197~1204.

[12] 徐宝文,陈振强,周晓宇.基于依赖性分析的面向对象 Ada95 程序切片.软件学报,2001,12(Suppl.):208~213.