一种基于程序DD图的无约束边生成算法

叶俊民[1,2] 王俊杰[1] 董 威[2] 齐治昌[2]
(华中师范大学计算机科学系 武汉 430079)[1] (国防科学技术大学计算机学院 长沙 410073)[2]

摘 要 基于相似路径集进行软件故障定位是众多有效故障定位方法中的一种,该方法利用测试技术、程序切片和削片技术给出具体的软件故障定位报告。在实现上述方法时,求出程序DD图(Decision-to-Decision Graph)的无约束边就是关键步骤。目前,针对这一关键步骤的研究中,虽然取得了一定进展,但如何基于程序DD图生成无约束边,尚需要进一步研究。首先选用十字链表结构存储程序的DD图,进而计算出该程序DD图中各边对应的主宰树和蕴含树,在此基础上求出程序DD图中无约束边。通过实验验证,提出的无约束边生成算法是一种有效的方法。

关键词 故障定位,无约束边,程序DD图

Algorithm for the Generation of Unconstrained Edges Based on the Decision-to-Decision Graph

YE Jun-min[1,2] WANG Jun-jie[1] DONG Wei[2] QI Zhi-chang[2]
(Department of Computer Science, Central China Normal University, Wuhan 430079, Chiha)[1]
(School of Computer, National University of Defense Technology, Changsha 410073, China)[2]

Abstract Using similar paths to localize the program faults is a method among many effective ones. Based on the techniques such as testing, program slicing and dicing, this paper presented a detailed report of the location of the faults in a program. When implementing the method, the generation of the unconstrained edges in a decision-to-decision graph is very important. At present, mang researches have been done, however, how to generate the unconstrained edges based on a decision-to-decision graph still needs further research. In our research, we chose orthogonal linked list as the data structure to store the decision-to-decision graph, then calculated the dominator tree and the implication tree of the graph. On this basis, we generated the unconstrained edges in a decision-to-decision graph. The experimental result shows that the proposed method is effective in the generation of the unconstrained edges.

Keywords Faults localization, Unconstrained edges, Decision-to-decision graph

1 引言

在程序自动调试和程序故障诊断研究中,Renieris[1]等人提出的"近邻模型"(Nearest Neighbor)认为:与失效路径最相似的成功路径比随机选择的成功路径更有利于故障定位,这一研究结论得到了实验的证实,从而推动了基于相似路径集的软件故障定位研究,这一研究涉及测试路径生成、测试输入数据生成[2-4]和动态程序切片[5-7]等技术,根据程序削片思想[7]分析成功测试路径与失效测试路径之间的差异,采用相似路径计算方法以定位程序故障。在上述思路中,定位故障的前提是计算相似路径,而相似路径计算的关键又在于求出程序DD(Decision-to-Decision Graph)图中的无约束边,在无约束边求出后,用替换无约束边的方法生成相似路径集[8],并最终实现程序故障定位。

当前在无约束边生成研究中,Mao[9]根据程序DD图的潜在特征和无约束边的定义,提出了求解程序DD图中无约束边的近似算法;Thomas[10]提出求图中控制点的算法,该方法可以作为求无约束边的基础。总之,虽然求解无约束边的研究工作已经取得了一定进展[11,12],但该问题尚需要进一步研究。

2 基于程序DD图的无约束边生成算法

2.1 研究基础

DD(Decision-to-Decision)图是一种决策到决策的有向图。有向图(Directed Graph) $G=(V,E)$,V是顶点的集合,E是有向边的集合,$e=(T(e),H(e))\in E$是一对有序的邻接结点,$T(e)$是尾,$H(e)$是头。若$H(e)=T(e')$,则e和e'是相邻边。$ind(n)$和$outd(n)$分别是结点n的入度和出度。

定义1(DD图) $G=(V,E)$有两条区分边e_0和e_k(唯一进入的边和唯一离开的边),e_0可到达E的任何一条边,E的任何一条边都可以到达e_k,对任何$n\in V, n\neq T(e_0), n\neq H(e_k), ind(n)+outd(n)>2, ind(T(e_0))=0, outd(T(e_0))=1, ind(T(e_0))=0, outd(T(e_0))=1, ind(H(e_k))=1, outd(H(e_k))=0$。

定义2(主宰关系) 设$G=(V,E)$是一个DD图,e_0和e_k分别是G的唯一进入边和唯一离开边。边e_i主宰e_j,当且仅

到稿日期:2008-03-04 本文得到国家自然科学基金(60673118),湖北省自然科学基金(2007ABA034)资助。
叶俊民(1965—),男,博士,教授,目前从事高可信软件工程、软件质量保障等方面研究,E-mail:jmye@mail.ccnu.edu.cn;王俊杰 研究兴趣为软件工程;董 威 博士,副教授,研究兴趣为可信软件工程;齐治昌 教授,博士生导师,研究兴趣为软件工程。

当 e_0 到 e_j 的任何一条路径都通过 e_i。

通过主宰关系可将 DD 图转换为一颗主宰树 $DT(G)$，若该树中的任意一对邻接结点(e_i,e_j)存在 $e_i=Parent(e_j)$，则称其为 e_j 的直接主宰。除了根结点外，任何结点都有唯一一个直接主宰。

定义 3(蕴含关系) 设 $G=(V,E)$是一个 DD 图，e_0 和 e_k 分别是 G 的唯一进入边和唯一离开边。边 e_i 蕴含 e_j，当且仅当从 e_i 到 e_k 的任何一条路径都通过 e_j。

通过蕴含关系，可将 DD 图 G 转换为一颗蕴含树 $IT(G)$，若该树中的任意一对邻接结点(e_i,e_j)存在 $e_i=Parent(e_j)$，则称其为 e_j 的直接蕴含。在上述概念的基础上，利用 Lengauer 等人提出 LT 算法[2]，可以通过程序流图(及其对应 DD)求出对应的主宰树和蕴含树。

定义 4(无约束边) 如果一条边 e_u 既不主宰其他的结点也不被其他结点所蕴含，则 e_u 称为无约束边。无约束边有 3 个特征：1)DD 图中的自环边是无约束边；2)对于 DD 图中的两条边 e_i 和 e_j(e_i(e_j)，若 $H(e_i)=H(e_j)$，$T(e_i)=T(e_j)$，则 e_i，e_j 是无约束边；3)若边 e 为 DD 图的无约束边，$ind(H(e))$ 与 $outd(T(e))$值均大于等于 2。

根据定义 2 和定义 3，有如下定理：

定理 1 在程序 DD 图 G 中，若边 e_i 主宰边 e_j，那么在同一程序 DD 图的逆图 G^{-1} 中，必存在边 e_i 蕴含边 e_j。

证明：在程序 DD 图中，设从边 e_0 到边 e_j 只存在如下 3 条路径，其中，路径 1：$e_0,e_{a1},e_{a2},\cdots,e_{ab},e_i,e_{c1},e_{c2},\cdots,e_{cd},e_j$，路径 2：$e_0,e_{f1},e_{f2},\cdots,e_{fg},e_i,e_{h1},e_{h2},\cdots,e_{hl},e_j$，路径 3：$e_0,e_{m1},e_{m2},\cdots,e_{mn},e_i,e_{p1},e_{p2},\cdots,e_{pq},e_j$，根据主宰关系定义，可得：边 e_i 主宰 e_j。在该程序 DD 图的逆图 G^{-1} 中，假设 e_0 和 e_k 仍然是 G^{-1} 唯一进入边和唯一离开边，那么从边 e_j 到边 e_k 仍然只存在如下 3 条路径，路径 1：$e_j,e_{cd},\cdots,e_{c2},e_{c1},e_i,e_{ab},\cdots,e_{a2},e_{a1},e_k$，路径 2：$e_j,e_{hl},\cdots,e_{h2},e_{h1},e_i,e_{fg},\cdots,e_{f2},e_{f1},e_k$，路径 3：$e_j,e_{pq},\cdots,e_{p2},e_{p1},e_i,e_{mn},\cdots,e_{m2},e_{m1},e_k$，因此根据蕴含关系定义，可得：边 e_i 蕴含边 e_j。

定理 2 在程序 DD 图 G 的逆图 G^{-1} 中，通过各条边之间的主宰关系转化成的主宰树，必定和同一程序 DD 图 G 通过各条边之间的蕴含关系转化成的蕴含树是完全相同的。

证明：根据定理 1，在程序 DD 图 G 的逆图 G^{-1} 中，对于任意两条边 e_i，e_j，若满足边 e_i 主宰边 e_j，那么在程序 DD 图 G 中，必然存在边 e_i 蕴含边 e_j。因此 G^{-1} 中各边之间的主宰关系即为 G 中的蕴含关系，故由 G^{-1} 转化成的主宰树和由 G 转化成的蕴含树是相同的。

定理 3 程序 DD 图 G 中的无约束边集合为 $UTL(G)$和 $ITL(G)$的交集，其中 $UTL(G)$表示主宰树 $UT(G)$中的叶子节点，$ITL(G)$表示蕴含树 $IT(G)$中的叶子节点。

证明：$UTL(G)$即为主宰树中不主宰其他边的边；$ITL(G)$即为蕴含树中不蕴含其他边的边。根据无约束边定义，可得无约束边集合即为 $UTL(G)$和 $ITL(G)$的交集。

2.2 无约束边生成算法

本算法的设计思想如下：选择支持双向遍历并且空间复杂度不高的十字链表建立并存储程序 DD 图，由于十字链表结合了有向图的邻接表和逆邻接表，因此可存储 DD 图及其逆图中的信息；采用 Thomas 算法[5]求出程序 DD 图中各边对应的主宰树和蕴含树，根据定理 3 求出无约束边。

根据算法思想进一步说明相关的设计决策如下。

第一，基于十字链表的 DD 图存储结构，可将该 DD 图 G 中边的关系转化成中间图 $G1$ 中顶点之间的关系。设 G 转化为 $G1$ 的策略如下：1)G 中边作为 $G1$ 中顶点；2)在 G 中，若 e_1 是某顶点的入边，e_2 是该顶点的出边，那么 $G1$ 中顶点 e_1 到顶点 e_2 有一条边；(3)特别地，在 G 中，如果某顶点的入边是 e_1，$e_2,e_3,\cdots,e_k,\cdots,e_m$，出边是 $e_{m+1},e_{m+2},e_{m+3},\cdots,e_{m+n}$，那么，在 $G1$ 中，分别连接从顶点 e_1 到顶点 $e_{m+1},e_{m+2},e_{m+3},\cdots,e_{m+n}$ 的边，从顶点 e_2 到顶点 $e_{m+1},e_{m+2},e_{m+3},\cdots,e_{m+n}$ 的边，……，从顶点 e_k 到顶点 $e_{m+1},e_{m+2},e_{m+3},\cdots,e_{m+n}$ 的边，……，从顶点 e_m 到顶点 $e_{m+1},e_{m+2},e_{m+3},\cdots,e_{m+n}$ 的边。根据定理 1－定理 3，可将程序 DD 图转化成算法所需要的中间图 $G1$，$G1$ 存放着程序 DD 图 G 求程序无约束边的中间信息。

第二，将 $G1$ 作为 Thomas 算法的输入，则可得到 $G1$ 中顶点之间的主宰关系，也即得到了程序 DD 图中各边之间的主宰关系。在本文中的主宰关系用主宰树存储。设主宰树用数组 dom 存储，对于任意一个节点 i，若 dom[i] = j，则在主宰树中节点 i 的父亲节点是 j。

第三，根据 2.1 节中的定理 2，要通过蕴含关系将程序 DD 图转化成蕴含树，只需要求出程序 DD 图的逆图对应的主宰树即可。因此只要再将十字链表表示的 DD 图的逆图中各边的关系转化成图 $G2$ 中各顶点之间的关系，即可求出图 $G2$ 中各顶点之间的主宰关系，同理可以得到程序 DD 图的逆图中各边之间的主宰关系，也即程序 DD 图中各边之间的蕴含关系。本文中得到的蕴含关系用蕴含树进行存储。设蕴含树用数组 im 存储，对于任意一个节点 i，若 im[i] = j，则在蕴含树中节点 i 的父亲节点是 j。

第四，根据 2.1 节中的定理 3，即可求出程序 DD 图中的无约束边。

具体无约束边的生成算法如图 1 所示。

算法功能：求程序 DD 图中的无约束边；

输入：程序 DD 图的有关信息，包括：

　　顶点和边的个数

　　各条边的起始节点和终止节点；

输出：DD 图中无约束边的编号；

Step1. 根据输入程序 DD 图的有关信息，建立程序 DD 图的十字链表；

Step2. 根据十字链表中表示的程序 DD 图原图中的信息，建立图 G1，DD 图中边的关系转化成图 G1 中顶点间的关系；

Step3. 求解中间图 G1 中各顶点之间的主宰关系，得出 DD 图中各边之间的主宰关系，生成主宰树；

Step4. 根据十字链表中表示的程序 DD 图的逆图中的信息，建立 G2，DD 图的逆图 G^{-1} 中边的关系转化成图 G2 中边的关系；

Step5. 求解图 G2 中各顶点之间的主宰关系，得出 DD 图中各边之间的蕴含关系，生成蕴含树；

Step6. 在主宰树和蕴含树中叶子节点对应的边即为所求的程序 DD 图中的无约束边。

图 1　无约束边的生成算法

2.3 算法的计算复杂性分析

在图 1 的算法中，对于一个有 n 个顶点 m 条边的图，求解其主宰关系的时间复杂度是 $O(m\log n)$；对于输入中的一个有 n 个顶点 m 条边的程序 DD 图，根据输入信息建立十字链

表表示的 DD 图的时间复杂度是 O(1)；将十字链表表示的 DD 图原图(逆图)中的边转化成中间图中的顶点，因为需要扫描所有顶点，每个顶点都对应一个所有入边和出边的二重循环，那么最坏情况下的时间复杂度为 $O(nm^2)$，求中间图中各顶点之间的主宰关系时，因为涉及到转化后的中间图中顶点的个数和边的条数，所以该算法的最坏情况下的时间复杂度为 $O(m^2 2\log m)$。

3 实验结果

在上述算法的基础上，下面通过一个如图 2 所示的实例说明本算法的求解过程。根据 Step2，可获得如图 3 所示的图 $G1$；按照 Step3，通过求解 $G1$ 中各顶点之间的主宰关系，得出程序 DD 图中各边之间的主宰关系，并求出如图 4 所示的主宰树，在此的主宰树用数组 dom 存储，具体结论是：dom[0]=−1，dom[1]=0，dom[2]=0，dom[3]=1，dom[4]=1，dom[5]=1，dom[6]=1，dom[7]=1，dom[8]=1，dom[9]=1，dom[10]=1，dom[11]=0；根据 Step4，求出程序 DD 图的逆图 G^{-1} 中各边之间的关系并转化成如图 5 所示的 G2；根据 Step5，求出 G2 中各顶点之间的主宰关系，以得出 DD 图中各边之间的蕴含关系，并生成如图 6 所示的蕴含树，在此的蕴含树用数组 im 存储，具体结论是：im[0]=11，im[1]=11，im[2]=11，im[3]=11，im[4]=11，im[5]=11，im[6]=11，im[7]=11，im[8]=11，im[9]=11，im[10]=11，im[11]=−1。

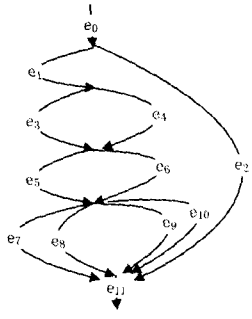

图 2 程序 DD 图

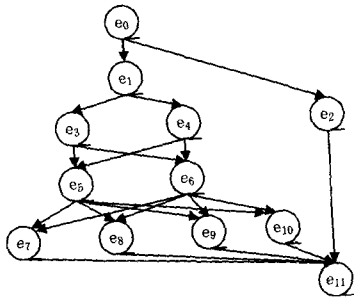

图 3 求主宰树生成的中间图 G1

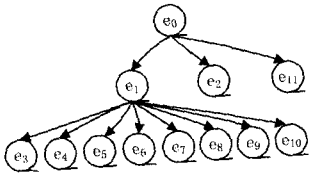

图 4 主宰树

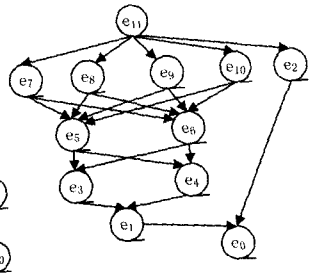

图 5 求蕴含树生成的中间图 G2

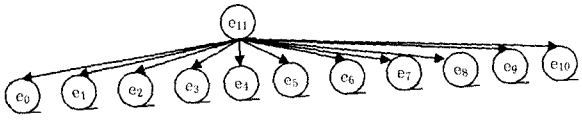

图 6 蕴含树

通过图 4 和图 6 可知，在主宰树和蕴含树中均为叶子节点的节点只有 $e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}$，也就是说该程序 DD 图中的无约束边即为：$e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}$。

结束语 求出程序 DD 图中的无约束边，对于采用相似路径集进行故障定位的研究至关重要用，本文提出了一种基于程序的 DD 图求解程序中无约束边的算法，在算法中选取了十字链表作为程序 DD 图的存储结构，通过求解 DD 图及其逆图中各边的主宰关系，求出程序 DD 图对应的主宰树和蕴含树，进而求出程序 DD 图中的无约束边。通过实验验证，本文提出的算法能够有效生成程序 DD 图的无约束边。进一步的工作是，在无约束边已知的前提下，研究软件故障的定位。

参 考 文 献

[1] Renieris M, Reiss S P. Fault Localization with Nearest Neighbor Queries// Proceedings of the 18th International Conference on Automated Software Engineering (ASE2003). Montreal, Canada, October 2003: 30-39

[2] Denise A, Gaudel M C, Gouraud S D. A Genetic Method for Statistical Testing[C]// the Proceedings of 15th International Symposium on Software Reliability Engineering (ISSRE2004). Saint-Malo, Bretagen, France, November 2004: 25-34

[3] 宫云战. 软件测试[M]. 北京：国防工业出版社，2006

[4] Weiser M. Program Slicing: Formal, Psychological and Practical Investigation of an Automatic Program Abstraction Method. Ph. D. Thesis. Univserisity of Michigan, Ann Arbor, Michigan, 1979

[5] Agrawal H, Horgan J R, London S, et al. Fault Localization using Execution Slices and Dataflow Test // the Proceedings of the 6th International Symposium on Software Reliability Engineering. Toulouse, France, October 1995: 143-151

[6] Agrawal H, Horgan J R. Dynamic Program Slicing. In the Proceedings of the ACM SIGPLAN 1990 Conference on Programing Language Design and Implementation. White Plains, New York, SIGPLAN Notices, 1990, 25(6): 246-256

[7] Lyle J R, Weiser M. Automatic Program Bug Location by Program Slicing// the Proceedings of the 2nd International Conference on Computers and Applications. June 1987: 877-882

[8] 朱凯. 一种基于相似路径集生成的程序故障定位方法. 硕士论文. 2007

[9] 毛澄映，卢炎生. 分支测试中测试路径用例的简化生成方法[J]. 计算机研究与发展，2006，43(2)：321-328

[10] Lengauer T, Targan R E. A Fast Algorithm for Finding Dominators in a FlowGraph. ACM Transactions on Programming Languages and Systems, 1979, 1(1): 121-141

[11] Lowrv E, Medlock C. Object Code Optimization[J]. Communications of the ACM(C. ACM), 1969, 12(1): 13-22

[12] Purdom P W, Moore E F. Algorithm 430: Immediate Predominators in a Directed Graph[J]. Communications of the ACM, 1972, 15(8): 777-778