一种模型重构冲突消解算法

陈军冰[1,2] 王志坚[1] 陈 波[1] 钱 思[1]
(河海大学计算机及信息工程学院 南京 210098)[1] (河海大学科学研究院 南京 210098)[2]

摘 要 冲突是研究模型重构中的一个重要问题,多数研究将该问题的重点放在冲突检测上,通过对已知冲突的分析,找出冲突消解的手工实现方式。为实现模型重构的自动过程而寻找自动消解冲突的方法是主要研究内容。根据冲突发生的条件将冲突分为3种类型:同一规则的并行使用产生的冲突、对称冲突、非对称冲突。该方法建立在手工分析这3类重构冲突消解的基础上,将重构规则预设为一个规则矩阵,对图转换系统中出现的重构规则进行扫描。扫描结果对照规则矩阵,判断冲突是同一规则还是不同规则的并行使用所产生;分别对这两种情况下的冲突所操作的对象进行分析,根据已有手工消解方法有针对性地进行消解操作。这一算法可以初步实现并行执行的3类冲突的自动消解。

关键词 模型重构,关键对,冲突,冲突消解,算法
中图法分类号 TP311 **文献标识码** A

Model Refactoring Conflict Resolution Algorithm

CHEN Jun-bing[1,2] WANG Zhi-jian[1] CHEN Bo[1] QIAN Si[1]
(Computer & Information Engineering College, Hohai University, Nanjing 210098, China)[1]
(Research Academy, Hohai University, Nanjing 210098, China)[2]

Abstract Conflict resolution is a key problem in research of model refactoring, while the majority of researchers focus on conflicts detection. Conflict resolution is usually performed manually after being analyzed known conflicts. Three categories of conflicts could be resolved which include conflict of parallel applications of the same rule, symmetric conflict and asymmetric conflict. This paper concentrated on automating conflicts resolution so as to realize the automatic model refactoring. This method provides an integrated algorithm based on manual analysis of three categories of conflicts. The basic automatic resolution algorithm within refactoring according to the cause of the conflicts (the application of either the same rule or the different ones), divides the conflicts and then resolves them correspondingly. This algorithm could preliminarily realize the automatic resolution of conflicts mentioned which are caused by the parallel application of refactoring rules.

Keywords Model refactoring, Critical pairs, Conflict, Conflict resolution, Algorithm

1 前言

基于图转换[1]的模型重构[2,3]过程中,涉及到模型重构规则的并行应用问题[4]。正如进程的并行运行导致对系统资源的占有与等待而引发进程死锁现象一样,由于模型重构过程中重构规则的并行应用导致了重构规则之间产生冲突现象[5](见图1)。这些冲突将导致模型坏味道(bad smell)。由于存在重构规则冲突及潜在冲突,将影响模型重构的质量。因此,研究重构冲突检测及消解是模型重构中的一个关键问题。

模型重构规则之间的冲突关系可以通过图转换中的关键对[6]进行分析。一个关键对是一对转换(P_1, P_2),其中 $p_1(m_1): G \rightarrow H_1$ 与 $p_2(m_2): G \rightarrow H_2$ 冲突,这里 G 是最小图。亦即 G 是左手边规则 p_1 和 p_2 的联结。它(关键对)能在所有可能方面通过重叠 L_1 和 L_2 计算,例如 L_1 和 L_2 交集中包含至少一条规则是被其中规则之一或在它们各自发生的事件上都可应用于 G 的两个规则所删除或改变。

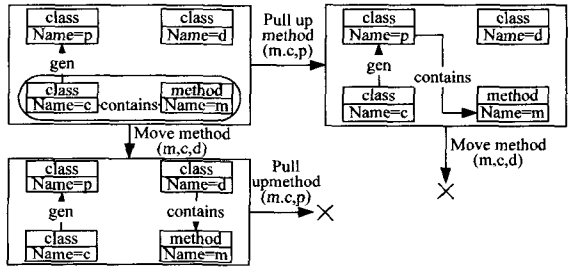

图1 重构规则 Pull up Method 与 Move method 之间产生冲突

到稿日期:2009-08-31 返修日期:2009-11-19 本文受河海大学自然科学基金项目(理科类)(2008432311),国家科技支撑计划(2008BA29B03)资助。

陈军冰(1969-),男,博士生,副研究员,主要研究方向为软件工程、软件重构等,E-mail:jbchen88@sina.com;王志坚(1958-),男,博士,教授,博士生导师,主要研究方向为软件工程、软件复用、基于网络的软件系统集成技术;陈 波(1987-),女,本科生,主要研究方向为软件重构、信息与通信等;钱 思(1988-),男,本科生,主要研究方向为软件重构、信息与通信等。

关键对集合代表所有潜在冲突。即一个关键对(p_1,p_2)存在,当且仅当 p_1 应用导致 p_2 不可应用,或者反之。如果两个规则应用至少满足上述情形之一,则两个规则应用是冲突的。

表1 冲突类型

类型	冲突
第一类:规则自身并行应用两次所产生冲突	Move Variable/ Move Method
	Pull Up Variable/ Pull Up Method
	Encapsulate Variable
	Create Superclass
	Rename Class/ Rename Variable/Rename Method
第二类:对称冲突	Move Variable 对 Pull Up Variable 冲突/Move Method 对 Pull Up Method
	Move Variable 对 Encapsulate Variable/Pull Up Variable 对 Encapsulate Variable
	Move Method 对 Encapsulate Variable/Pull Up Method 对 Encapsulate Variable
	Create Superclass 与 Rename Class
	Rename Variable 分别与 Move Variable 或 Pull Up Variable/Rename Method 分别与 Move Method 或 Pull Up Method
第三类:非对称冲突	Create Superclass 对 Pull Up Variable /Create Superclass 对 Pull Up Method
	Rename Variable 对 Encapsulate Variable
	Encapsulate Variable 对 Rename Method

共有3种规则应用产生冲突:(1)一个规则应用删除了一个图对象,该对象是另一个规则应用的匹配。(2)一个规则应用产生了一个图对象,该对象增加的图结构被另一个规则应用的否定应用条件(NAC)所禁止。(3)一个规则应用改变了另一个规则应用与之所匹配的属性[5]。

给定两个图转换 t_1 和 t_2,如果 t_1 先执行,t_2 不能执行,称为非对称冲突;如果 t_1,t_2 不管是谁先执行都导致另一个转换不能执行,称为对称冲突[7]。

根据3种规则将冲突定义、模型重构冲突类型分为3类:规则自身并行应用两次所产生的冲突、对称冲突、非对称冲突。每种冲突分别包含若干冲突情况,见表1。

2 冲突检测

AGG[8]中研究了使用关键对分析(Critical Pair Analysis)检测结构重构的冲突。该算法给定一个图转换规则集,首先计算一张表,该表显示每一个规则对之间的关键对数量。需要计算所有规则之间的关键对,对每一个关键对分别判断3种类型冲突条件。该算法实际是对关键对规则进行遍历,计算效率不高。刘辉[9]借助图转换的冲突检测机制及关键对技术,利用检测转换规则与转换约束之间的关系,给出了重构冲突检测算法。该算法针对4条特性保持约束的转换规则[9]进行检测,实际上也是一个遍历算法。LeenLambers 等[10]中给出对于两个非删除规则或一个删除、一个非删除规则集关键对的一个有效的计算冲突关键对方法。因为以下两个原因使这一方法可以更有效率:(1)如果两个规则是无删除的,那么没有必要计算任何一个 L_1 和 L_2 的重叠(m_1,m_2)。(2)如果其中一个规则是非删除的,另一个规则是删除的,那么没有必要计算所有 L_1 和 L_2 的重叠(m_1,m_2),直接计算那些引出关键对的重叠就足够了。这个算法可免除对于非删除规则对交叠的计算。上述算法都可实现模型重构规则的自动冲突检测。

3 冲突消解

根据表1描述的3类冲突,分别讨论冲突消解方法,见表2。

表2 冲突消解方法

类型	冲突	消解方法
第一类:规则自身并行应用两次所产生冲突	Move Variable / Move Method	通过更改其中一个变量的名称可以消解这种冲突。应用 Move Method 两次的消解方法类似。
	Pull Up Variable / Pull Up Method	通过删除其中一个变量,然后把另一个变量上拉入高级别类中来解决。应用 Pull Up Method 两次的解决方式也类似。
	Encapsulate Variable	通过忽略其中一个来解决。
	Create Superclass	通过忽略其中一个来解决。
	Rename Class / Rename Variable / Rename Method	对其中一个重命名赋予优先级,通过手工选择其中一个类,将其重命名。应用 Rename Variable 或 Rename Method 两次产生的冲突,解决方式同上。
第二类:对称冲突	Move Variable 对 Pull Up Variable / Move Method 对 Pull Up Method	如果将一个变量移动到一个有超类的类中,那么 Move 和 Pull up 相同变量是汇合的。这种情况下,在移动变量后仍能将其上拉。反之,变量在 Pull up 后始终能被 Move。如果要将一个变量移动到没有超类的类中,那么关键对就不汇合,因为 Pull up Variable 不能实现(由于缺少超类)。 第三种情况:同时将名称相同的两个变量 Pull up 和 Move 到相同类,这将会导致一个汇合冲突情况。可先对其中一个变量重命名,再执行重构过程。Move Method 对 Pull Up Method 产生冲突同上,解决方式也同上。
	Move Variable 对 Encapsulate Variable /Pull Up Variable 对 Encapsulate Variable	移动变量对封装变量:先移动一个变量,在新类中对其封装,从而这种情况是汇合的。先封装变量,若随后,不仅变量被移动了,而且新创建 getter 和 setter 方法,我们达到了相同的改变状态。如果新的类中不存在这样的存取方法,这些重构过程才是可能的。否则又要附加使用重命名来使得这一情形是汇合的。/ 上拉变量对封装变量:先上拉变量,可在超类中对其封装。如果先对其封装,不仅要 Pull Up Variable,也要将存取方法 Pull Up。正如在上例中提到的,为了使得这一情形是汇合的,必须另外使用重命名。
	Move Method 对 Encapsulate Variable / Pull Up Method 对 Encapsulate Variable	移动方法对封装变量:如果封装变量导致创建一个方法,同时要将相同名称的方法移动到同一类中,这一冲突能被解决,须先对要移动的方法重命名,再将其移动,最后封装变量。同时进行上拉变量和封装变量将产生类似冲突,解决方式同上。
	Create Superclass 对 Rename Class	对其中一个重构过程赋予优先级。
	Rename Variable 分别与 Move Variable 或 Pull Up Variable /Rename Method 分别与 Move Method 或 Pull Up Method	重命名变量对移动变量:先将移动变量,再在新类中将其重命名。或者先将其重命名,再移动。 上拉变量与重命名变量会导致类似冲突。同时重命名方法与移动方法也会导致相似冲突,因而,解决方法相似。
第三类:非对称冲突	Create Superclass 对 Pull Up Variable / Create Superclass 对 Pull Up Method	先应用创建超类再上拉变量,那么后一规则要应用两次,以达到相反顺序应用这两个规则所得到的结果。上拉方法也会产生同样的冲突,因而,解决同上。
	Rename Variable 对 Encapsulate Variable	先重命名变量,要对改名后的变量进行封装。如果先封装变量,那么要得到同上结果,就要同时重命名变量和存取方法。
	Encapsulate Variable 对 Rename Method	封装一个变量会产生一个新的方法。如果把一个方法重命名为这个新方法的名称,那么就会产生冲突,必须忽略其中一个重构过程或者再次进行重命名。

4 一种冲突消解算法

现有的冲突消解方法是：通过分析关键对找到扩展的潜在冲突是汇合并且能被消解的，再用手工方式分析冲突消解方法。因此，在此基础上进一步研究针对两个规则并行执行产生冲突的自动消解方法。

重构过程表示为对类、方法、变量的操作。表 3 列出重构规则与变量或方法的对应关系。

表 3 变量或方法与重构规则对应表

变量或方法	所使用的重构规则	变量或方法	所使用的重构规则
v_1			p_1
v_2	p_1	v_1	p_2
v_3			p_3
v_1			p_1
v_2	p_2	v_2	p_2

从表 3 中可知，同一个规则可同时应用于不同的变量或方法，如对两个不同的变量和方法进行重命名（仅在同一类中将它们重命名为同一名称的情况下会产生冲突）；不同规则也可应用于同一变量或方法，如将变量重命名＋上拉。

对于重构过程，可以对其重构规则进行列表，见表 4。P_m 表示相同规则并行应用两次，p_{mn} 表示不同规则的并行应用。

表 4 重构规则应用表

规则 a \ 规则 b	p_1	p_2	…	p_n
p_1	p_{11}	p_{12}	…	p_{1n}
p_2	p_{21}	p_{22}	…	p_{2n}
…	…	…	…	…
p_n	p_{n1}	p_{n2}	…	p_{nn}

表 4 中，p_{11}，p_{22}，…，p_m 表示一个重构规则并行执行两次的规则序列，如 p_{11} 表示 p_1 并行执行两次，见表 4 中对角线位置。

p_{12}，p_{21}，…，p_{mn} 表示不同重构规则并行执行的规则序列，如 p_{12} 表示 p_1 和 p_2 并行执行，见表 4 中 p_{12} 位置。

重构过程所涉及变量列表，见表 5。其作用在于，当冲突检测程序在检测到冲突时，检测表 5。依据所应用的对象，将产生冲突的两个规则进行分类，即 1）对同一变量或方法进行操作，2）对不同变量进行操作。

表 5 重构过程涉及变量表

变量 \ 变量	v_1	v_2	…	v_n
v_1	v_{11}	v_{12}	…	v_{1n}
v_2	v_{21}	v_{22}	…	v_{2n}
…	…	…	…	…
v_n	v_{n1}	v_{n2}	…	v_{nn}

根据文献[5]中冲突消解方法定义重构规则优先级，见表 6。如对称冲突 1）Move Variable vs Pull Up Variable 对同一变量上拉和移动，上拉后始终能移动，而移动后上拉就会产生冲突。2）Move Variable vs Encapsulate Variable 移动变量和封装变量，先执行前者不会产生冲突，反之则会产生冲突。3）Pull Up Variable vs Encapsulate Variable 上拉变量和封装变量同上。同时，对变量和方法操作中，上拉、移动和重命名情况类似。创建超类是依据非对称冲突 1）Create Superclass vs Pull Up Variable /Create Superclass vs Pull Up Method 而定义的。

表 6 重构规则优先级表

重构规则	优先级
p_1 上拉	1
p_2 移动	2
p_3 重命名	3
p_4 封装	4
p_5 创建超类	5

表 6 中，值越小表示优先级越高。

根据文献[5]将判断过程分为两种情况：（1）同一规则的并行使用产生的冲突；（2）不同规则应用的并行使用产生的冲突。不同规则应用又分为：a）对称冲突的解决方式，b）不对称冲突的解决方式。重构冲突自动消解的算法思路是将重构规则预设为一个规则矩阵，对图转换系统中出现的重构规则进行扫描。扫描结果对照规则矩阵，判断冲突是同一规则还是不同规则的并行使用所产生；分别分析这两种情况下的冲突所操作的对象，根据已有手工消解方法有针对性地进行消解操作。通过这一算法可以初步实现并行执行的 3 类冲突的自动消解。

算法描述如下：

Begin do 检测表 4；

If p_{11}，p_{22}，…，$p_{mn}=1$；//表示重构过程中同一规则的并行使用产生冲突（“=1”表示对同一个对象的操作冲突）。其中 p_{11}，p_{22}，…，p_{mn} 表示一个重构规则并行执行两次的规则序列，如 p_{11} 表示 p_1 并行执行两次，见表 4 中对角线位置

then do 读取表 5；

if $i=j$；//即对同一变量或方法进行操作

then 通过忽略其中一步来解决；

if $i\neq j$；//即对不同变量或方法进行操作

then 随机选取过程中某一变量或方法，对其进行重命名，再使用规则；

if p_{12}，p_{21}，…，$p_{mn}=1$，且 $m\neq n$；//表示不同规则的并行使用产生冲突。p_{12}，p_{21}，…，p_{mn} 表示不同重构规则并行执行的规则序列，如 p_{12} 表示 p_1 和 p_2 并行执行，见表 4 中 p_{12} 位置

then 读取表 6 规则优先级表；

调整并行使用的规则 a 和规则 b 的顺序，以达到消解冲突的目的；

//即调整表 4 中对角线左下规则应用顺序，按照表 4 中右上规则应用顺序进行重构过程（指定规则应用顺序）

End.

算法流程图如图 2 所示。

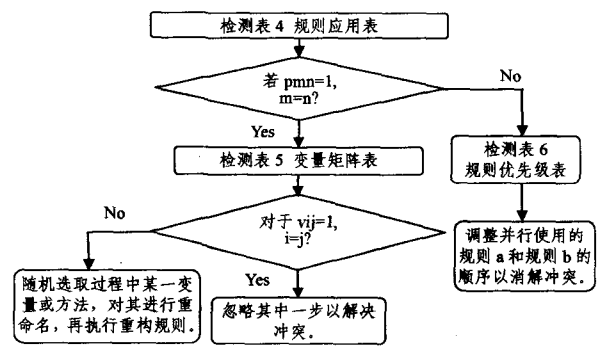

图 2 冲突消解算法流程图

结束语 本文主要研究模型重构过程中自动消解冲突的方法。算法在对冲突方式分类的基础上，首先判别冲突类型，

（下转第 173 页）

算法。以上实验结果表明，基于 OBDD 的与或图符号搜索算法具有较低的空间复杂度，可处理问题的规模与使用显式存储技术的传统算法相比有较大的提高。

表 1　符号算法与 AO＊算法执行结果比较

名称	结点数	与结点数	与结点百分比	符号算法运行时间(s)	AO＊算法运行时间(s)
a1	101	13	13%	0.02	0.015
a2	1001	155	15%	0.20	0.046
a3	5001	801	16%	1.11	0.109
a4	10001	2039	20%	2.73	0.109
a5	15001	3006	20%	4.50	0.094
a6	20000	3246	16%	5.36	0.140
a7	23000	4626	20%	7.50	0.125
a8	30001	4873	16%	8.14	溢出
a9	50001	4998	10%	18.06	溢出
a10	60000	5367	9%	25.33	溢出
a11	80000	6481	8.1%	35.39	溢出
a12	100000	8068	8.1%	46.42	溢出
a13	200002	16271	8.1%	97.89	溢出
a14	300000	62256	20%	169.89	溢出
a15	500000	102434	20%	308.45	溢出
a16	1000000	202903	20%	733.00	溢出
a17	1500001	303279	20%	1223.03	溢出
a18	2000000	403752	20%	1740.52	溢出

结束语　与或图搜索是一项重要的人工智能问题求解技术，本文对基于 OBDD 的无圈与或图符号求解技术进行了探讨，主要贡献有：

1）与或图的符号表示技术，包括与结点、或结点、叶子结点和有向弧的 OBDD 表示，以及代价有向弧的分层表示和存储技术；

2）基于以上表示的与或图自底向上解图最小代价计算方法，以及自顶向下的解图析取技术。

以后将研究有圈与或图的符号搜索算法，以及这些算法在大型装配序列规划、机器人规划中的应用。

参 考 文 献

[1] Bryant R E. Graph-based algorithms for Boolean function manipulation[J]. IEEE Trans. on Computers, 1986, 8: 677-691

[2] Drechsler R, Sieling D. Binary decision diagrams in theory and practice[J]. International Journal on Software Tools for Technology Transfer, 2001, 3(2): 112-136

[3] Bloem R, Gabow H N, Somenzi F. An algorithm for strongly connected component analysis in n log n symbolic steps[C]// Proceedings of International Conference on Formal Methods in Computer-Aided Design. 2000: 37-54

[4] Cao T, Sanderson A C. AND/OR net representation for robotic task sequence planning[J]. IEEE Trans. Systems Man Cybernet—Part C: Applications and Reviews, 1998, 28(2): 204-218

[5] DeMello L S H, Sanderson A C. A correct and complete algorithm for the generation of mechanical assembly sequences[J]. IEEE Trans. Robotics and Automation, 1991, 7(2): 228-240

[6] Homen de Mello L S. AND/OR graph representation of assembly plans[J]. IEEE Trans. Robotics and Automation, 1990, 6(2): 188-199

[7] Martelli A, Montanari U. Additive AND/OR Graphs[C]// Proceedings of the International Joint Conference on Artificial Intelligence. 1973: 1-11

[8] Martelli A, Montanari U. Optimizing Decision Trees Through Heuristically Guided Search[J]. Communications of the ACM, 1978, 21(12): 1025-1039

[9] Nilsson N J. Principles of Artificial Intelligence[M]. Palo Alto: Tioga Publishing Company, 1980

[10] Mahanti A, Bagchi A. AND/OR Graph Heuristic Search Methods[J]. Journal of the Association for Computing Machinery, 1985, 32(1): 28-51

[11] Chakrabarti P P. Algorithms for Searching Explicit AND/OR Graphs and Their Applications to Problem Reduction Search [J]. Artificial Intelligence, 1994, 65: 329-345

[12] Jiménez P, Torras C. An Efficient Algorithm for Searching Implicit AND/OR Graphs with Cycles[J]. Artificial Intelligence, 2000, 124: 1-30

[13] 谢青松，王岩冰，马绍汉. 显式与或图的一种新的贪心搜索算法[J]. 计算机研究与发展，1997，34(12)：887-892

[14] Mahanti A, Ghose S, Sadhukhan S K. A Framework for Searching AND/OR Graphs with Cycles [R]. The AI/0305001. Computing Research Repository, 2003

(上接第 143 页)

根据冲突所涉及的冲突因素，再分别实现消解。这一算法可以初步实现并行执行的 3 类冲突的自动消解。本方法建立在已知手工分类和消解的基础上，没有包含对可能产生的新的重构规则的自动识别及消解。

参 考 文 献

[1] Corradini A, Montanari U, Rossi F, et al. Algebraic approaches to graph transformation I: Basic concepts and double pushout approach[C]// Rozenberg G. editor. Handbook of Graph Grammars and Computing by Graph transformation. Volume 1: Foundations. World Scientific, 1997

[2] Mens T. On the use of graph transformations for model refactoring[C] // Generative and Transformational Techniques in Software Engineering. LNCS 4143. Springer, 2006: 219-257

[3] Mens T, Van Eetvelde N, Demeyer S, et al. Formalizing refactorings with graph transformations[J]. Journal on Software Maintenance and Evolution, 2005, 17(4): 247-276

[4] Habel A, Hoffmann B. Parallel Independence in Hierarchical Graph Transformation[M]. ICGT, 2004: 178-193

[5] Taentzer TG, Runge O. Detecting structural refactoring conflicts using critical pair analysis[J]. Electronic Notes in Theoretical Computer Science, 2005, 127(3): 113-128

[6] Plump D. Critical pairs in term graph rewriting [C]// Proc. Mathematical. Foundations of Computer Science. volume 841 of Lecture Notes in Computer. London, UK: Springer-Verlag, 1994

[7] Mens T. A formal foundation for object-oriented software evolution[D]. Department of Computer Science, Vrije Universiteit Brussel, September 1999

[8] http://user.cs.tu-berlin.de/~gragra/agg/critical_pairs.html.

[9] 刘辉，麻志毅，邵维忠. 模型转换中的特性保持的描述与验证[J]. 软件学报，2007，18(10)：2369-2379

[10] Lambers L, Ehrig H, Orejas F. Efficient detection of conflicts in graph-based model transformation [C] // Proc. International Workshop on Graph and Model Transformation (GraMoT'05). Electronic Notes in Theoretical Computer Science, Tallinn, Estonia, Elsevier Science, 2005