ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.1, January 2010, pp.84−97
doi: 10.3724/SP.J.1001.2010.03525

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于二分决策图的特征模型验证方法*

闫 华[1,2], 张 伟[1,2], 赵海燕[1,2+], 梅 宏[1,2]

[1](高可信软件技术教育部重点实验室(北京大学),北京 100871)

[2](北京大学 信息科学技术学院 软件研究所,北京 100871)

BDD-Based Approach to the Verification of Feature Models

YAN Hua[1,2], ZHANG Wei[1,2], ZHAO Hai-Yan[1,2+], MEI Hong[1,2]

[1](Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China)

[2](Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

+ Corresponding author: E-mail: zhhy@sei.pku.edu.cn, http://www.sei.pku.edu.cn/~zhhy

Yan H, Zhang W, Zhao HY, Mei H. BDD-Based approach to the verification of feature models. *Journal of Software*, 2010,21(1):84−97. http://www.jos.org.cn/1000-9825/3525.htm

Abstract: The feature model is a reusable requirements model generated from the domain analysis. The reuse of feature models is usually achieved by a customizing-based approach. One important issue in feature models' customization is the verification problem, caused by the fact that there are usually constraints among features, and that a valid customizing result must satisfy all these constraints. Because of the NP-hard nature of this problem, it is usually difficult to verify feature models in an efficient way. This paper presents a BDD (binary decision diagram)-based approach to verifying feature models by only traversing once to the nodes in BDDs, an approach that makes an efficient use of the BDD data structures based on the unique characteristics of feature models' verification. It should be pointed out that this approach does not attempt to resolve the NP-hard difficulty of the verification problem in a general sense, but just tries to improve the scalability and efficiency of methods for feature models' verification based on the utilization of this problem's uniqueness. Experimental results show that this BDD-based approach is more efficient and can verify more complex feature models than the previous method.

Key words: feature model; verification; BDD (binary decision diagram); domain engineering; software reuse

摘 要: 特征模型是领域分析活动产生的具有复用价值的软件需求模型.对特征模型的复用通常采用定制的方式.特征模型定制中的一个重要问题是验证问题.该问题的存在是因为特征之间往往具有一定的约束关系,而一个合法的定制结果必须保证特征之间所有约束关系的被满足性.由于特征模型验证问题 NP-hard 所具有的性质,如何高效地进行特征模型的验证就成为一件相对困难的事情.在深入挖掘特征模型验证问题特殊性的基础上,将这种特殊性和二分决策图的结构特点进行了有效的结合,提出了一种通过对二分决策图的一次遍历即能实现特征模型验证的方法.需要指出的是,该方法并非试图在一般意义上解决特征模型验证问题中 NP-hard 的困

* Supported by the National Natural Science Foundation of China under Grant Nos.60528006, 60703065, 60873059 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2006AA01Z156, 2007AA01Z123 (国家高技术研究发展计划(863)); the National Basic Research Program of China under Grant No.2005CB321805 (国家重点基础研究发展计划(973))

Received 2008-05-16; Revised 2008-08-28; Accepted 2008-10-28

难性,而是尽可能地利用该问题的特殊性,以提高处理特征模型定制问题的规模和效率.实验数据表明,相比较以前采用的验证方法,基于 BDD(binary decision diagram)的方法在处理特征模型验证问题的规模和效率上都具有显著的提高.

关键词: 特征模型;验证;BDD(binary decision diagram);领域工程;软件复用

中图法分类号: TP311 **文献标识码:** A

特征模型是捕获特定领域可复用软件需求的一种重要技术手段[1−7].其基本思想是把软件需求组织成一组具有客户/用户价值的软件特征以及这组特征之间的依赖关系,从而提高软件需求的易定制性.其主要目的则是通过定制,实现对特定领域可复用软件需求方便和有效的复用.即,在开发特定领域内的软件应用时,开发者不需要从零开始进行软件需求的获取和分析,而是通过对当前领域特征模型的定制(从中选择出一组可被当前软件应用复用的特征)实现大粒度的复用,并在定制结果的基础上进行下一步的软件工程活动.

特征模型定制中的一个重要问题是验证问题[6].这个问题的存在是由于特征之间的非完全独立性所导致的:特征之间往往会存在特定的约束关系,而一个合法的定制结果必须要保证特征之间所有约束关系的被满足性.因此,当对特征模型做出一个定制决策**后,我们必须要对这个定制决策进行及时的验证,以确保其没有破坏特征之间的约束关系(称为特征模型定制决策的验证).否则,一个不正确的定制决策将会被隐式地传播到后继的定制活动中,从而对定制活动产生负面影响.另外,在进行定制活动之前,我们首先要保证特征之间约束关系的正确性(称为特征模型约束关系的验证).否则,定制活动本身即失去了存在的必要性.在本文中,我们将使用“特征模型的验证问题”同时指代上述两种验证问题.

特征模型验证问题的困难性是由于其NP-hard的性质所导致的.在本质上,特征模型的验证问题是一种约束可满足性问题(constraint satisfaction problem,简称CSP),而研究者已经认识到:在一般意义上,CSP问题是一种NP-hard问题[8].根据我们的经验,当一个特征模型中包含的特征数量较为庞大且特征之间的约束关系较为复杂时,使用第三方的模型检查工具对其进行验证,往往需要耗费较长的时间,甚至导致验证工具进入活锁状态.

针对上述问题,本文提出了一种基于二分决策图(binary decision diagram,简称BDD)的特征模型验证方法.其主要特点在于充分考察了特征模型验证问题的特殊性,并针对这种特殊性对BDD的数据结构进行了高效的利用.具体而言,本文提出的方法只需要通过对特征模型对应的BDD中结点的一次遍历即能实现对特征模型的验证.需要指出的是,本文的方法并非试图在一般意义上解决模型检查问题中NP-hard的困难性,而是尽可能地利用特征模型验证问题自身具有的特殊性,尽量提高处理特征模型定制问题的规模和效率.实验数据表明:相比较我们以前提出的基于SMV***的特征模型验证方法[7],本文提出的基于BDD的方法在处理特征模型验证问题的规模和效率上都有了显著的提高.

本文第 1 节主要介绍特征模型定制问题相关的背景知识.第 2 节详细阐述基于 BDD 的特征模型验证方法.第 3 节给出本文基于 BDD 的验证方法的相关实验数据,并与我们以前验证方法的实验数据进行对比.第 4 节对相关工作进行分析和说明.最后,第 5 节对全文进行总结.

1 背景知识

本节主要介绍与特征模型定制问题相关的背景知识,主要包括 3 个方面:特征模型的基础知识、特征模型的 3 条验证准则以及 BDD 的基本原理和特点.

1.1 特征模型

一般而言,特征模型主要由 3 种成分构成:特征、特征之间的精化关系以及特征之间的约束关系.特征是一

** 本文中,“对特征模型的一个定制决策”是指决定是否将一个特征保留在定制结果中(绑定该特征),或是从定制结果中删除该特征.

*** SMV(symbolic model verifier,符号模型检查工具)是由卡内基·梅隆大学开发的一个基于BDD的通用型模型检查工具[9].

种具有客户/用户价值的软件特点,指代了一组具有紧密关联关系的单个需求.精化关系提供了将具有不同粒度或抽象层次的特征组织成层次结构的基本方式.特征之间的精化关系可以进一步划分为分解化、特殊化以及属性化 3 种类型.由于这 3 种具体类型的精化关系与特征模型的验证没有直接的联系,因此,本文将不对其进行更详细的介绍.

特征之间存在 3 种类型的约束关系:二元约束、组约束以及组合约束.下面我们分别介绍这 3 种类型的约束关系常见的子类型以及各个子类型基于命题逻辑的形式化定义.

二元约束是指存在于两个特征之间的约束关系.表 1 给出了两种类型的二元约束及其形式化定义.其中谓词 *bound*(*F*:Feature)指代了特征 F 是否处于被绑定状态.

Table 1 Two types of binary constraint and formal definition

表 1 二元约束的两种类型及其形式化定义

Binary constraint	Formal definition
requires(*A*,*B*:Feature)	$bound(A)\rightarrow bound(B)$
excludes(*A*,*B*:Feature)	$\neg bound(A)\vee\neg bound(B)$

组约束是指存在于一组特征之间的约束关系.表 2 给出了 3 种类型的组约束及其形式化定义.

Table 2 Three types of group constraint and formal definition

表 2 组约束的 3 种类型及其形式化定义

Group constraint	Formal definition
requires-group(*A*,*B*,…,*C*:Feature)	$\forall F1,F2\in(A,B,\ldots,C)\cdot requires(F1,F2)$
excludes-group(*A*,*B*,…,*C*:Feature)	$\forall F1,F2\in(A,B,\ldots,C)\cdot excludes(F1,F2)$
loose-group(*A*,*B*,…,*C*:Feature)	True

组合约束是指存在于两组绑定谓词之间的约束关系.表 3 给出了两种类型的组合约束及其形式化定义.其中涉及的绑定谓词(binding-predicate)在表 4 中给出了相应的定义.

Table 3 Two types of composite constraint and formal definition

表 3 组合约束的两组类型及其形式化定义

Composite constraint	Formal definition
requires (*p*,*q*:Binding-Predicate)	$p\rightarrow q$
excludes (*p*,*q*:Binding-Predicate)	$\neg p\vee\neg q$

Table 4 Three types of binding-predicate and formal definition

表 4 绑定谓词的 3 种类型及其形式化定义

Binding-Predicate	Formal definition
Or (*A*,*B*,…,*C*:Feature)	$bound(A)\vee bound(B)\vee\ldots\vee bound(C)$
And (*A*,*B*,…,*C*:Feature)	$bound(A)\wedge bound(B)\wedge\ldots\wedge bound(C)$
Xor (*A*,*B*,…,*C*:Feature)	$bound(A)\otimes bound(B)\otimes\ldots\otimes bound(C)$

特征模型中的约束关系有两种来源:一种是由特征建模人员显式加入的约束关系;另一种是由特征之间的精化关系隐式产生的约束关系.具体而言,这种隐式的约束关系体现为:在一个精化关系连接的两个特征中,子特征的绑定必须依赖于父特征的绑定.也就是说,只有在父特征已被绑定的情况下,子特征才有被绑定的可能.显然,这种约束关系是二元约束关系的一种具体类型:*requires*(*A*,*B*:Feature).

关于特征模型的更详细信息可参阅文献[1,7,10]等.

1.2 特征模型的验证准则

从特征模型验证的视角来看,特征模型可以被抽象为一组特征以及这组特征之间存在的一组约束关系.根据特征绑定状态的不同,特征模型中的特征可以被划分至 3 个集合:已绑定集合、已删除集合以及待决策集合.其中,已绑定集合包含了所有已经处于被绑定状态的特征;已删除集合包含了所有已经被删除的特征;待决策集合则包含了所有尚未被做出绑定决策的特征.一个定制决策的实施会导致一个待决策特征的绑定状态发生改变:或变为已绑定,或变为已删除.

我们以前的研究提出了验证特征模型的 3 条准则.即,如果下列 3 条准则未被满足,则表明或者特征模型的

约束关系存在缺陷,或者对特征模型的定制决策存在缺陷[7].

- 准则 1.存在对待决策集合中所有特征的一组绑定决策,使得当这组决策被实施后,特征模型中的所有约束关系均未被破坏;
- 准则 2.待决策集合中的任一特征都有被绑定的可能,且这种被绑定的可能性不会破坏特征模型中的任一约束关系;
- 准则 3.待决策集合中的任一特征都有被删除的可能,且这种被删除的可能性不会破坏特征模型中的任一约束关系.

von der Maβen 和 Lichter 在考察特征模型常见缺陷的基础上,建立了一个系统性的特征模型缺陷分类框架[10].我们先前的研究表明[9]:上述 3 条验证准则能够检查出该缺陷分类框架中所有异常类型和不一致类型的缺陷.表 5 给出了上述 3 条验证准则对于特征模型缺陷的具体检查效果.关于上述 3 条验证准则以及特征模型缺陷分类框架的更详细信息可参阅文献[7,10],本文不再赘述.基于上述 3 条准则对于特征模型缺陷检查的有效性,本文将继续采用其作为特征模型的验证准则.这 3 条准则也构成了特征模型验证问题的特殊性之一.在第 2 节中,我们将详细阐述如何基于这种特殊性实现对特征模型的高效验证.

Table 5 Three criteria's effects on checking deficencies in feature models

表 5 3 条验证准则对于特征模型缺陷的检查效果

Deficiency	Number of types	Result
Anomaly	6	Four types can be detected before customization. Two types can be detected in customization
Inconsistency among constraints	4	All the four types can be detected in customization
Inconsistency among customizing decisions	7	All the seven types can be detected in customization

虽然上述3条验证准则对于特征模型缺陷的检查非常有效,但实现对这3条验证准则可满足性的自动检查却是一件相对困难的事情.直观而言,准则 1 的可满足性检查是一个二元CSP问题,其时间复杂度为$O(2^n)$;这里,n为待决策集合中特征的数量.对任一待决策特征而言,准则 2 和准则 3 的可满足性检查也可以分别归结为一个二元CSP问题,其时间复杂度为$O(2^{n-1})$.基于上述分析,上述 3 条验证准则可满足性的检查可以转化为 $2n+1$ 个CSP问题,其所需的总时间复杂度为$O((n+1)\cdot 2^n)$.显然,采取将上述 3 条准则简单地转化为 $2n+1$ 个CSP问题,并对这些CSP问题的可满足性进行检查的验证方式,不是一种高效的手段.

1.3 二分决策图

二分决策图(BDD)是一种表示布尔函数的压缩型数据结构[11].对任意命题逻辑表达式均可建立相应的BDD,所得到的BDD与该表达式的真值表等价;但由于其消除了真值表中的冗余信息,因此,在存储空间上,BDD一般要比真值表小很多.虽然BDD不能从理论上保证将搜索空间从指数级压缩到多项式级,但它对搜索空间的压缩仍然具有显著的效果(比如将 2^n压缩至 $2^{n/2}$).通过这种压缩,BDD能够解决模型检查领域中许多原来无法解决的规模较大的问题.

给定一个命题逻辑表达式以及该表达式中变量的一个顺序,即可以唯一确定一个BDD.例如,对于逻辑表达式$(a\leftrightarrow b)\wedge(c\leftrightarrow d)$,采用变量顺序$\langle a,b,c,d\rangle$和$\langle a,c,b,d\rangle$建立的 BDD 分别如图 1 中的左部和右部所示.可以看到,一个 BDD 通常有多层构成,每一层中的结点对应于逻辑表达式中的一个命题变量.每一个结点与右侧层中的结点通过 true 分支(图中带箭头的实线)和 false 分支(图中带箭头的虚线)相连接,分别表示当前结点对应的变量取值为 true 和 false 的两种情况.BDD 的最右层包含 true 和 false 两个结点,分别对应当前逻辑表达式取值为 true 和 false 的两种情况.从最左层结点到最右层 true 结点的一条路径代表的含义是:当逻辑表达式中的变量按照该路径指明的方式取值时,当前逻辑表达式的取值为 true.例如,对于图 1 中左部的 BDD,路径$\langle a$ true,b true,c false,d false,true$\rangle$表示:当a=true,b=true,c=false,d=false 时,$(a\leftrightarrow b)\wedge(c\leftrightarrow d)$的取值为 true.同样,从最左层结点到最右层 false 结点的一条路径则指出了当前逻辑表达式取值为 false 的一种情况.

另外,从图1中还可以看到,对于同一个命题逻辑表达式,采用不同的变量顺序建立的BDD,其结点数量会存

在差异.当命题表达式较为复杂时,变量顺序对 BDD 结点规模的影响将是巨大的;一个不当的变量顺序将导致对 BDD 搜索时间的显著增加,甚至因为存储空间的急速增长而无法成功建立 BDD.因此,如何找到一个最优的变量顺序使得产生的 BDD 的结点数量最少,是建立 BDD 时首先要考虑的一个问题.

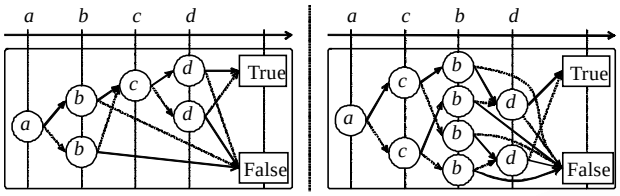


Fig.1 Two BDDs corresponding to the logical expression: $(a\leftrightarrow b)\wedge(c\leftrightarrow d)$

图 1 逻辑表达式$(a\leftrightarrow b)\wedge(c\leftrightarrow d)$对应的两个 BDD

目前已有很多成熟的BDD开发包(例如JDD[12]).通过开发包的接口,不仅可以直接将任意逻辑表达式转换为BDD,而且可以完成更高级的BDD运算和操作.关于BDD的更详细内容可参阅文献[12−14]等.

2 基于二分决策图的特征模型验证

特征模型验证问题是NP难的.我们以往的经验表明,当特征模型包含的特征数量较为庞大且约束关系较为复杂时,采用第三方工具(例如SMV[13])对其验证往往效率低下,甚至陷入活锁状态.因此,有必要在这个问题上进行研究,找到一种更高效的方法对特征模型进行验证.在本文论述的方法中,采用BDD作为高效验证特征模型的工具,因为它在模型检查和形式化验证等领域已被广泛应用,并被证明十分高效[11,14].我们在使用BDD验证特征模型时,重点考察了特征模型的结构特点,并结合BDD的优良特性,从而形成了一套高效的验证方法.该验证方法包括两个部分:(1) 为特征模型建立BDD,即BDD的建立方法;(2) 在BDD上验证特征模型,即BDD的使用方法.其中,方法的第 1 部分又涉及两个重要问题,分别是:(1) 建立BDD的逻辑变量顺序是什么;(2) 建立BDD时逻辑表达式的合并顺序是什么.这两个问题处理的得当与否将直接影响验证的效率,但在BDD的研究上,研究者尚无法对所有应用给出这两个问题的通解[14].因此,针对特征模型验证这类特殊的BDD应用,必须充分考虑特征模型的结构特点,妥善解决这两个问题,才能保证高效的验证.在BDD的使用方法上,特征模型的验证与一般的模型检查问题和形式化验证问题相比更为复杂,验证方法的第 2 部分将对它展开讨论.

本节将详细论述基于 BDD 的特征模型验证方法.因为前文已经扼要介绍了 BDD 的基本概念和原理,为了方便理解,本节将首先介绍如何使用 BDD 验证特征模型,然后再论述 BDD 的建立方法.

2.1 特征模型验证准则的检查

特征模型的 3 条验证准则是构成特征模型验证问题的特殊性之一.我们将这种特殊性和 BDD 数据结构的特点进行了有效的结合,发现了一种通过对特征模型 BDD 结点的一次遍历即能检查出这 3 条准则是否被满足的方法.

在特征模型的 BDD 已经建立的情况下,对准则 1 的被满足性的检查相对简单:只要 BDD 的根结点不是直接连接到 false 结点,那么准则 1 就是被满足的.因为准则 1 用于检验约束关系的可满足性,如果约束关系不可满足,那么约束关系对应的真值表的表值将全部取 false,BDD 则将该真值表压缩为只含一个 false 结点,所以只需检查 BDD 的根结点是否直接连接到 false 结点.而对准则 2 和准则 3 的被满足性的检查则需要通过对 BDD 结点进行一次遍历.图 2 展示了对这两个准则进行检查的基本思想.

对于一个特征 *A*,为了检查其是否具有被绑定的可能性,我们只需检查其对应的所有 BDD 结点的 true 分支是否全部都连接到了 false 结点上(如图 2 左部所示):如果是,则表明该特征已经失去了被绑定的可能,即,准则 2 没有被满足;否则,该特征仍然具有被绑定的可能性.同理,对于一个特征 *B*,为了检查其是否具有被删除的可能性,我们只需要检查其对应的所有 BDD 结点的 false 分支是否全部都连接到了 false 结点上(如图 2 右部所示):

如果是,则表明该特征已经失去了被删除的可能,即,准则 3 没有被满足;否则,该特征仍然具有被删除的可能性.

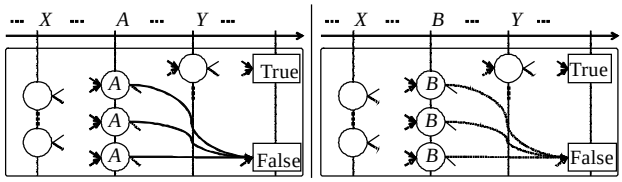


Fig.2 The idea of checking Criterion 2 and 3

图 2 准则 2 和准则 3 的检查方法示意

为了在算法上实现上述的检查方法,我们还需要考虑 BDD 中存在跨越分支的情况.跨越分支的存在是为了保证 BDD 的精简性,这种精简性会使得特定的变量结点从 BDD 中被删除,从而导致上述检查方法的失效.图 3 展示了这样的一种情况.在图 3(a)所示的 BDD 中,特征 *A* 的所有变量结点的 true 分支都连接到了 false 结点上,但它仍然具有被绑定的可能.导致这种情况的原因在于一个分支跨越了特征 *A* 所在的层,使得特征 *A* 对应的一个结点被删除了.如果恢复这个被删除的结点,我们会得到一个与图 3(a)所示的 BDD 等价但存在冗余性的 BDD,即图 3(b)所示的 BDD.在这个 BDD 中,特征 *A* 的所有变量结点的 true 分支并没有全部连接到 false 结点上,因此,它仍然具有被绑定的可能.我们需要在算法中考虑跨越分支对于上述检查方法的影响,从而确保检查结果的准确性.

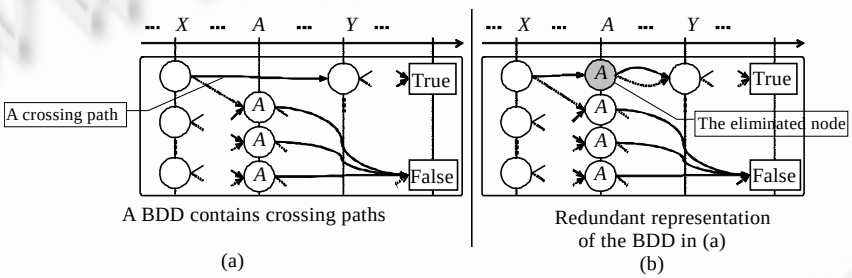


Fig.3 Crossing paths in BDD and their redundant representations

图 3 BDD 中的跨越分支及其冗余表示方式

基于上述分析,我们开发了一种用于检查准则 2 和准则 3 的算法,详细信息见算法 1.其中,函数 Node get_true_branch(Node *e*)返回结点 *e* 通过 true 分支所连接的另一个结点;函数 Node get_false_branch(Node *e*)返回结点 *e* 通过 false 分支所连接的另一个结点;函数 Boolean isNonCrossedLayer(layer)返回当前层 layer 是否存在被跨越分支穿越的情况.这 3 个函数是在建立 BDD 的过程中被同时构造的.局部变量 isCriterion2Violated 和 isCriterion3Violated 分别表示准则 2 和准则 3 是否被违反,集合 violatedFeatures 存储违反准则 2 和准则 3 的特征名字.根据算法返回的 violatedFeatures 是否为空,可判断特征模型是否违反准则 2 和准则 3.

算法 1. 特征模型验证准则 2 和准则 3 的检查算法.

输入:特征模型的 BDD;

输出:违反准则 2 和准则 3 的特征集合.

```
Verify(root){
    Set violatedFeatures=∅;
    for (layer=getLayer(root) to getLayer(0)){
        if (isNonCrossedLayer(layer)==true){
            isCriterion2Violated=true;
            isCriterion3Violated=true;
```

```
            for each node e of layer{
                if (get_true_branch(e)!=false_node) isCriterion2Violated=false;
                if (get_false_branch(e)!=false_node) isCriterion3Violated=false;
            }
            if (isCriterion2Violated==true||isCriterion3Violated==true){
                featureName=getFeatureName(layer);
                violatedFeatures.add(featureName);
            }
        }
    }
    return violatedFeatures;
}
```

2.2 特征模型二分决策图的建立

设$C_1,C_2,\ldots,C_n$是特征模型中包含的一组约束关系,将它们进行合取得到的逻辑表达式为$\wedge_{i=1,\ldots,n}\ C_i$,则这个逻辑表达式对应的BDD就称为特征模型的BDD.特征模型BDD的建立需要考虑两个问题.首先,因为变量顺序决定BDD的最终规模,所以需要找到一种较优的变量顺序,使得最终建立的BDD的规模不要太大.其次,即使BDD的最终规模较小,不恰当的计算顺序也可能产生规模非常大的中间结果[****](类似于矩阵连乘的情况),而这个中间结果的计算可能在时间和空间上都无法忍受.一旦解决这两个问题,就可以通过现有的BDD开发包(例如JDD[12])建立BDD.因此,必须寻求一种较优的BDD合并顺序,从而可以基于单个约束关系的BDD,通过合并产生特征模型的BDD.

针对上述两个问题,本文提出了建立特征模型 BDD 的两种启发式策略.需要指出的是,这两种启发式策略是针对精化关系的,而没有关注由特征建模人员显式加入的约束关系.这出于两点考虑:首先,一般而言,精化关系是数量最多的约束关系,它是验证效率的瓶颈,因此优化策略必须重点考虑它.其次,显式加入的约束关系往往各式各样,不像精化关系那样具备一般的形式,因此难以总结其普遍规律并给出优化策略.在我们目前实现的 BDD 建立算法中,首先为显式加入的约束关系建立 BDD,然后再将这些 BDD 与通过优化策略建立的精化关系的 BDD 加以合并,从而产生完整的特征模型的 BDD,它包括了特征模型中的全部约束关系.

策略 1:将特征树的后根深度优先遍历顺序作为建立特征模型 BDD 的逻辑变量顺序.

这里,特征树是指特征之间通过精化关系形成的树状结构.这种策略的效果是:在仅考虑精化关系产生的约束关系的情况下,通过这种策略建立的特征模型 BDD 具有较小的结点数量.下面,我们通过两个具有代表性的示例对这种策略的效果进行说明.

首先,考虑如图 4(a)所示的特征模型,其中仅包含一棵具有 3 个结点的特征树.由这棵特征树中的精化关系导致的约束关系如图 4(b)所示.对于这个特征模型,采用一种后根深度优先的变量顺序$\langle B,C,A\rangle$建立的特征模型 BDD 如图 4(c)左部所示;采用一种非后根深度优先的变量顺序$\langle B,A,C\rangle$建立的特征模型 BDD 如图 4(c)右部所示.可以看到(例如采用穷举的方式),在这种情况下,后根深度优先变量顺序对应的 BDD 的结点数量是最少的.

其次,考虑如图 5(a)所示的特征模型,其中包含两棵分别具有两个结点的特征树.由这两棵特征树中的精化关系导致的约束关系如图 5(b)所示.对于这个特征模型,采用一种后根深度优先的变量顺序$\langle B,A,Y,X\rangle$建立的特征模型 BDD 如图 5(c)左部所示;采用一种非后根深度优先的变量顺序$\langle B,Y,A,X\rangle$建立的特征模型 BDD 如图 5(c)

**** 在逻辑表达式为$\wedge_{i=1,\ldots,n}\ C_i$相对复杂的情况下,直接针对这个逻辑表达式建立特征模型的BDD往往会产生规模巨大的中间结果,从而导致无法成功地建立BDD.本文采用的方式是首先针对约束关系的若干集合建立相应的BDD,然后通过对这些BDD的合并产生特征模型的BDD.在对这些BDD进行合并时,若合并顺序不当,也会产生规模巨大的中间结果.

右部所示.在这种情况下,后根深度优先变量顺序对应的 BDD 的结点数量也是最少的.

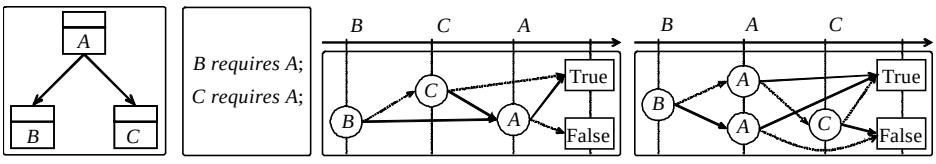


Fig.4 Smallest BDD corresponding to one feature tree

图 4 单棵特征树对应的最小 BDD 示例

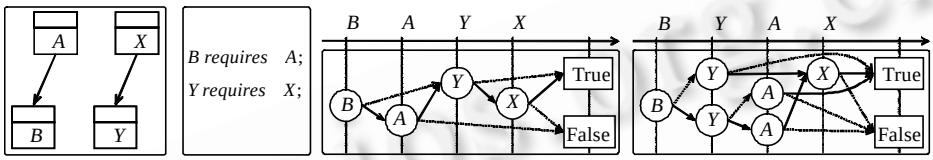


Fig.5 Smallest BDD corresponding to two feature trees

图 5 两棵特征树对应的最小 BDD 示例

在上述两个示例中,精化关系分别导致两个约束关系,需要为两个约束关系的合取式建立BDD,以便进一步验证.在这两个示例中,按照策略 1 的变量顺序,建立的 BDD 规模是最小的.在一般的特征模型中,存在大量精化关系,在不恰当的变量顺序下,对它们的约束关系进行合取将导致 BDD 的规模发生指数爆炸.策略 1 控制了BDD的规模,最理想的情况是其和的变量个数呈线性关系.但当特征模型规模较大时,精化关系导致的约束关系错综复杂,对这些约束关系进行合取将导致 BDD 规模在某一时刻开始可能呈指数速度增长.在这种情况下,策略 1 在一定范围内延缓了指数爆炸的发生.在很大程度上,策略 1 降低了 BDD 规模发生指数爆炸的可能.

策略 1 是对计算结果的优化,它会使得最终建立的特征模型 BDD 的规模较小.下面要介绍的策略 2 则是针对计算过程的优化,即,如何采取一种较优的 BDD 合并策略,从而能够尽可能地减小中间结果的规模.

策略 2:使用由精化关系形成的父子结构作为构建特征模型 BDD 的基本合并单元.

这里,由精化关系形成的父子结构是指由一个父特征、其所有子特征以及父子特征之间的精化关系形成的结构.这种策略的效果是:在仅考虑精化关系产生的约束关系的情况下,通过首先建立所有父子结构的 BDD,然后再对这些 BDD 进行合并的方式,其中间结果的规模较小.

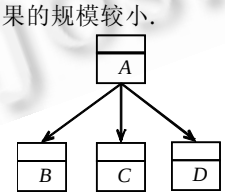


Fig.6 Parent-Child relation caused by refinement

图 6 由精化关系形成的父子结构示例

考虑图 6 中的父子结构,其中含有 3 个精化关系.由这 3 个精化关系导致的 3 个约束关系分别为 *B requires A*,*C requires A*,*D requires A*.假设当前建立特征模型 BDD 的工作已经通过合并产生了一个中间结果 BDD,记作 *p*.现在要处理上述 3 条约束.首先计算它们的 BDD,记 *B requires A* 的 BDD 为 *x*,*C requires A* 的 BDD 为 *y*,*D requires A* 的 BDD 为 *z*.然后要将 *x*,*y*,*z* 与 *p* 合并.这时候存在两种合并顺序:一种顺序是将 *x*,*y*,*z* 逐个地与 *p* 合并,另一种顺序是先将 *x*,*y*,*z* 合并,记作 *q*,再将 *q* 与 *p* 合并.我们的实践表明,第 2 种顺序的效率比第 1 种要高得多.基于这种观察,即产生了上述的策略 2.

BDD的合并类似于矩阵的连乘,虽然最终结果相同,但不同合并的顺序将导致不同的计算代价.当精化关系数量庞大时,策略2将大幅降低建立BDD的时间代价和空间代价.

利用上文介绍的启发式策略和BDD开发包的接口,容易得到建立特征模型二叉决策图的算法,详细信息见算法2.其中,Build函数通过调用*MakeOrder*,*BuildRefinementsBdd*和*BuildConstraintsBdd*这3个子函数计算出特征模型的BDD,并将其返回.*BDD.createBddVar*(*x*)表示为*x*建立一个BDD逻辑变量,*BDD.andTo*(*x*,*y*)表示在BDD上对*x*与*y*作合取运算,构造函数*BDD*(*x*)表示为逻辑表达式*x*建立BDD,这3个函数均为BDD开发包(例如JDD[12])提供的标准接口.*getFeatureTrees*()函数返回特征模型中所有特征树的根,*getConstraints*()函数返回所有显示加入的约束关系,这两个函数由特征模型提供.

算法2. 建立特征模型二叉决策图的算法.

输入:特征模型;

输出:特征模型的BDD.

```
Build(featureModel){
    for each root of featureModel.getFeatureTrees()
        MakeOrder(root);
    BDD refinementsBdd=BDD(true);
    for each root of featureModel.getFeatureTrees()
        refinementsBdd=BDD.andTo(refinementsBdd, BuildRefinementsBdd(root));
    BDD constraintsBdd=BuildConstraintsBdd(featureMode.getConstraints());
    BDD bdd = BDD.andTo(refinementsBdd, constraintsBdd);
    return bdd;
}
MakeOrder(root){                          //按策略1的顺序建立BDD的逻辑变量
    if (root.hasChildren==false){
        BDD.createBddVar(root);
        return;
    }
    for each child of root
        MakeOrder(child);
    return;
}
BuildRefinementsBdd(root){                //按策略2,为精化关系建立BDD
    LinkedList queue=∅;
    BDD refinementsBdd=BDD(true);
    for each child of root.chidren
        queue.addLast(child);
    while (queue.isEmpty()==false){
        Feature cur=queue.removeFirst();
        BDD tmp=BDD(true);
        for each constraint of cur.getRefinements()
            tmp=BDD.andTo(tmp, new BDD(constraint));
        refinementBdd=BDD.andTo(refinementsBdd,tmp);
        for each child of cur
            queue.addLast(child);
    }
    return refinementsBdd;
```

```
}
BuildConstraintsBdd(constraints){              //为建模人员显式加入的约束关系建立 BDD
    BDD constraintsBdd=BDD(true);
    for each constraint of constraints
        constraintsBdd=BDD.andTo(constraintsBdd,new BDD(constraint));
    return constraintsBdd;
}
```

虽然BDD是真值表的压缩表示,但是它不能将真值表的指数级空间大小压缩到多项式级.因为存储BDD的空间是指数级的,建立BDD的代价相当于对它进行一次遍历,所以,算法的空间代价是$O(2^n)$,时间代价也是$O(2^n)$.

3 工具支持及实验分析

目前我们已经将特征模型的定制和验证功能集成到我们在前期开发的特征建模支持工具中.图 7 是该工具的一张界面截图.其中,右上部区域展示了特征模型的内容以及模型中各个特征在当前定制版本中的绑定状态;左上部区域展示了当前特征模型所关联的一组定制版本,以及当前正在被编辑的定制版本;下部区域则是特征模型的验证窗口.在点击验证窗口工具栏上的启动按钮后,工具将自动对特征模型的当前定制版本进行验证,并将验证结果输出至验证窗口中.

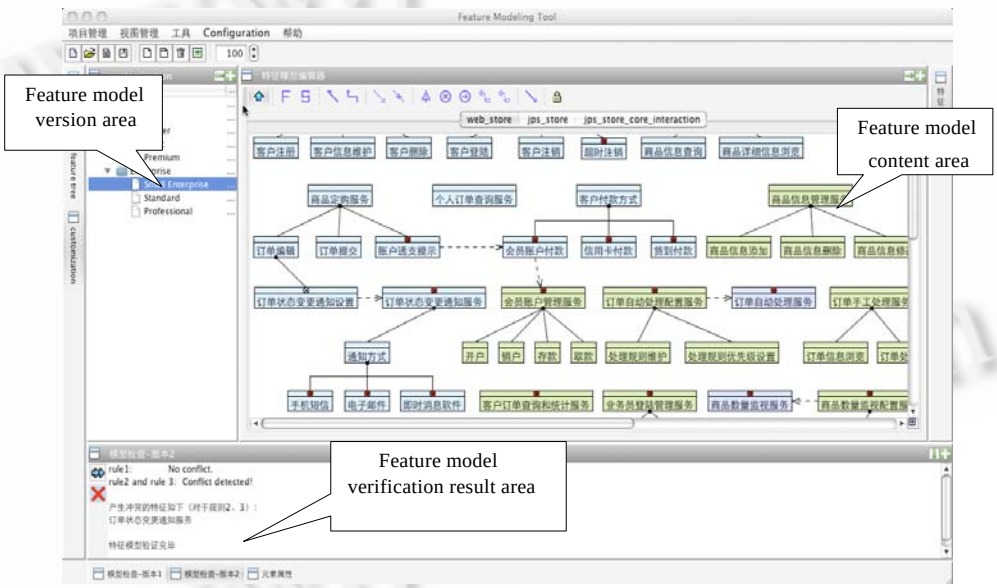


Fig.7 Feature modeling tool with the support of modeling, customization and verification

图 7 支持特征模型建模、定制与验证的软件工具

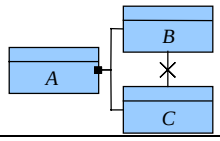
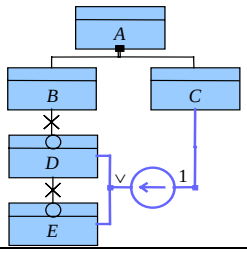
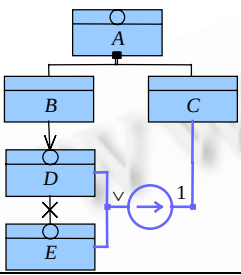
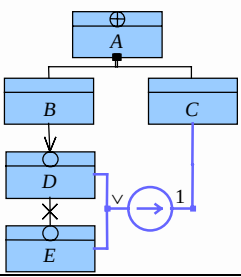
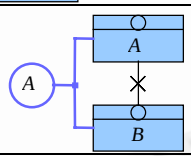
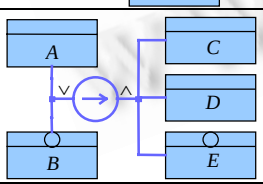
下面,我们首先通过若干包含冲突的小例子说明本文方法的有效性,然后对客户端电子邮件软件领域的特征模型实例[15]进行测试以进一步说明有效性,最后通过对大规模测试用例的实验结果说明本文方法的验证能力和验证效率.

表 6 列举了几组简单的特征模型,它们包含了各种典型冲突所发生的情况.经过测试,我们的工具可以得到完全正确的结果.

Table 6 Verification results of various sorts of conflicts (Column “Feature model” and column “Verification

result" are real pictures cut from supporting tool, please refer to Fig.7)

表 6 对各种冲突的验证效果(其中"特征模型"和"验证结果"两列均来自工具截图,请参见图 7)

Feature model	Constraints	Binding state	Verification result
	*excludes*(*B*,*C*)	Undecided={} Bound={*A*,*B*,*C*} Removed={}	Rule 1: Conflict detected! Rule 2 and Rule 3: No conflict
	*excludes*(*B*,*D*), *excludes*(*D*,*E*), *requires*(xor(*C*), or(*D*,*E*))	Undecided={*D*,*E*} Bound={*A*,*B*,*C*} Removed={}	Rule 1: No conflict Rule 2 and Rule 3: Conflict detected! Conflicting features: *D*, *E*
	*excludes*(*D*,*E*), *requires*(*B*,*D*), *requires*(or(*D*,*E*), xor(*C*))	Undecided= {*A*,*B*,*C*,*D*,*E*} Bound={} Removed={}	Rule 1: No conflict Rule 2 and Rule 3: No conflict
	*excludes*(*D*,*E*), *requires*(*B*,*D*), *requires*(or(*D*,*E*), xor(*C*))	Undecided={*D*,*E*} Bound={*A*,*B*,*C*} Removed={}	Rule 1: No conflict Rule 2 and Rule 3: Conflict detected! Conflicting features: *D*, *E*
	*excludes*(*A*,*B*), *requires-group*(*A*,*B*)	Undecided={*A*,*B*} Bound={} Removed={}	Rule 1: No conflict Rule 2 and Rule 3: Conflict detected! Conflicting features: *A*, *B*
	*requires*(or(*A*,*B*), and(*C*,*D*,*E*))	Undecided={*B*,*E*} Bound={*A*,*C*,*D*} Removed={}	Rule 1: No conflict Rule 2 and Rule 3: Conflict detected! Conflicting features: *E*

在表 6 的第 1 个例子中,约束条件不可能全部被满足,因此违反了准则 1,我们的工具检测出了这个冲突.第 2 个例子中,因为特征 *B* 和特征 *D* 是互斥的,且 *B* 已经被绑定,所以 *D* 必须被删除;而在特征 *C* 被绑定且 *B* 被删除的情况下,特征 *C*,*D*,*E* 之间的组合约束又要求 *E* 被绑定,但是 *D*,*E* 却在 Undecided 集合中,因此被我们的工具检测出来.第 3 个例子中没有任何冲突.第 4 个例子中,因为特征 *B* 是特征 *A* 的必选子特征,且 *A* 被绑定,所以 *B* 也被绑定,而特征 *B* 依赖于特征 *D*,因此 *D* 也必须被绑定;特征 *D*,*E* 之间的互斥关系导致 *E* 必须被删除,但是 *D*,*E* 却在 Undecided 集合中,因此被我们的工具检测出来.第 5 个例子中,特征 *A*,*B* 之间的组约束关系要求 *A*,*B* 或者

全部被绑定,或者全部被删除,而 A,B 之间的互斥关系导致 A,B 不能被全部绑定,因此 A,B 只能被全部删除,但 A,B 却在 Undecided 集合中,因此被我们的工具检测出来.第 6 个例子中,因为特征 A 被绑定,所以复杂约束左侧的组谓词成立,这导致右侧的组谓词也必须成立,因此特征 E 必须被绑定,但 E 却在 Undecided 集合中,因此被我们的工具检测出来.

图 8 和图 9 是客户端电子邮件软件领域的特征模型[15](由于篇幅所限,图中省略了与验证不相关的部分).其中,图 8 展示了特征模型的精化关系视图,图 9 展示了特征模型的约束关系视图.整个特征模型包含约 80 个特征,由精化关系导致的依赖关系约 60 条,由建模人员显示加入的约束关系约 20 条,绝大多数特征被约束关系覆盖.通过我们的工具对该特征模型进行验证,耗费的时间为 0.1s.我们的测试环境是一台配置了 2.0G Hz CPU, 512MB内存以及Windows XP操作系统的笔记本电脑.

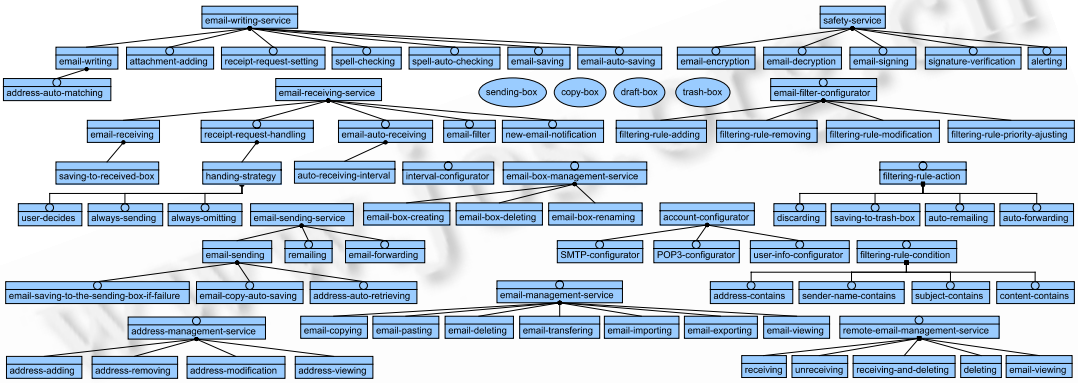


Fig.8 Feature refinement view of the e-mail client software

图 8 客户端电子邮件软件的特征精化关系视图

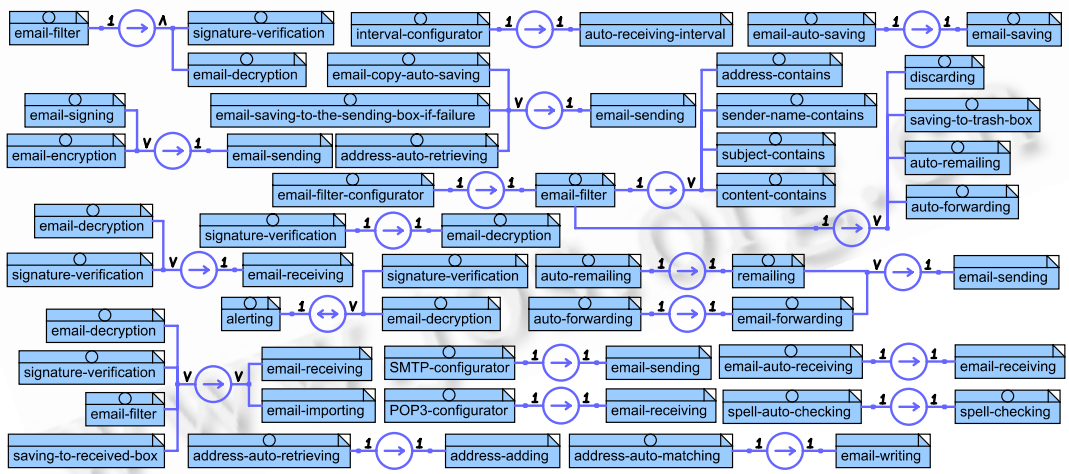


Fig.9 Feature constraint view of the e-mail client software

图 9 客户端电子邮件软件的特征约束关系视图

为了检查本文方法的验证能力和验证效率,我们设计了两组用于测试的特征模型.每组特征模型分别包含了 20 个独立的特征模型,每个特征模型包含的特征数量分别是 10,20,…,90 和 100,200,…,1 000.第 1 组特征模型中仅包含了较简单的二元约束关系;第 2 组特征模型则进一步增加了一些较为复杂的组约束关系和组合约束关系.作为比较,我们分别使用本文提出的基于BDD的方法和我们在前期研究中使用的基于SMV的方法[9]对这两组特征模型进行验证.其中,基于SMV的方法是将特征模型的验证问题分解为 $2n+1$ 个CSP问题,然后再将这 $2n+1$ 个CSP问题输入到模型检查工具SMV中进行求解.我们的测试环境是一台配置了 2.0G Hz CPU, 512MB内

存以及Windows XP操作系统的笔记本电脑.

图 10 展示了两种特征模型验证方法的实验数据.从图中可以看到,基于 SMV 的方法基本上无法处理特征数量超过 100 的特征模型.而当采用本文基于 BDD 的方法验证特征数量为 100 的特征模型时,只需要不到 1s 的时间.对于含有 500 个特征且约束关系复杂的特征模型,本文的方法仅需要 66.7s.而对于约束关系简单的特征模型,本文方法只需用时 37.2s 即能验证特征数量为 1 000 的特征模型.从这些实验数据可以看到,本文提出的基于 BDD 的验证方法在验证能力和验证效率上都比我们以前采用的基于 SMV 的方法有了大幅度的提高.

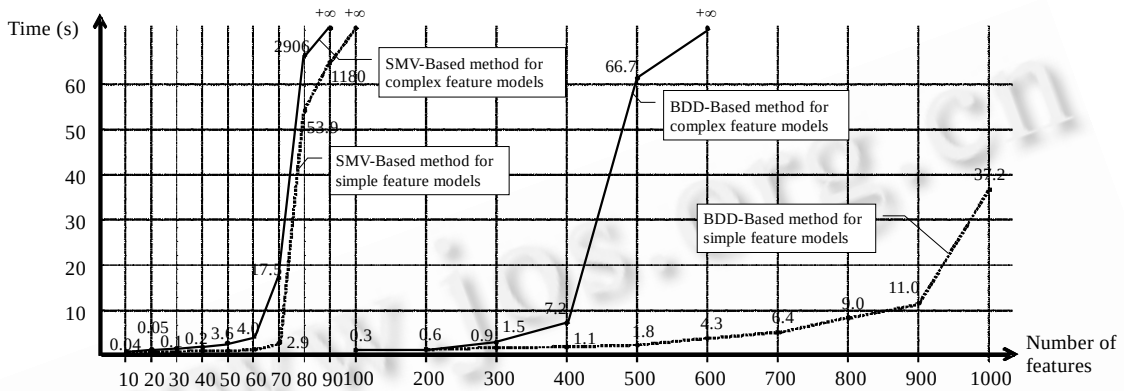


Fig.10 Experimental results of the two verification approaches

图 10 两种特征模型验证方法的实验数据

4 相关工作

特征模型的概念最早是由Kang等人[1]在面向特征的领域分析方法中首次提出来的,并被软件复用领域中的许多研究者作了进一步的发展[2−4,6].其中,在特征模型的验证问题上,Mannion[6]第 1 次提出了基于命题逻辑的验证方法.在此基础上,我们在先前的研究中对特征模型约束关系的不同类型进行了区分,并对每种类型的约束关系给出了其形式化定义以及相应的可视化表示符号,从而方便了特征建模人员对约束关系的建立和维护[7].我们还提出了特征模型的 3 条验证准则[7],并且在一个第三方建立的特征模型缺陷框架[10]中证明了其有效性[9].然而,对于如何自动化地检查这 3 条准则的被满足性,我们只是将其简单地转化成 $2n+1$ 个CSP问题,并使用模型检查工具SMV验证这些CSP问题.

本文基于 BDD 的特征模型验证方法受到了 Czarnecki 的研究工作的启发.在文献[16]中,Czarnecki 使用了一个商业化的 BDD 包对特征模型进行了验证.然而,Czarnecki 的工作仅仅考虑了特征之间的二元约束关系以及存在于父子特征之间的局部型复杂约束关系.另外,Czarnecki 也没有给出如何基于 BDD 进行特征模型验证的具体技术细节.

Batory在文献[15]中提出了一种基于逻辑真值维持系统的特征模型验证方法.该方法在时间复杂度和空间复杂度上与本文所提方法相同,均为$O(2^n)$.其中,n是特征模型中特征的数量.在该方法中,将特征模型中的约束关系转变成一个合取范式的时间复杂度为$O(2^n)$;检查缺陷的时间复杂度也为$O(2^n)$,因为该方法需要遍历合取范式中的所有析取子句.在本文的方法中,建立BDD的时间复杂度和空间复杂均为$O(2^n)$,遍历BDD的时间复杂度也为$O(2^n)$.但是,正如我们在文献[9]中所指出的,Batory的方法虽然能够检查出本文的方法所能检查出的大部分约束关系缺陷和定制决策缺陷,但这些缺陷被检查出的时间点却要晚于本文的方法.也就是说,相比较而言,本文的方法能够更早地发现约束关系或定制决策中存在的缺陷,这对于及时发现不正确的定制决策,以避免其传播到后继的定制活动中具有重要的意义.

5 总 结

本文提出了一种基于 BDD 的特征模型验证方法.该方法的主要特点在于其充分考虑了特征模型验证问题

的特殊性,并将这种特殊性和 BDD 数据结构的特点进行了紧密的结合,从而使得只需对 BDD 进行一次遍历即能实现对特征模型的验证.实验结果表明,这种方法较之我们以前采用的方法在验证能力和效率上都有了大幅度的提高.

本文所提出的特征模型验证方法仍然具有进一步优化的可能性.特别地,在 BDD 的建立方法上,本文提出的两种策略并不能保证 BDD 建立过程的最优性以及结果的规模最小性.在今后的研究中,我们会继续深入挖掘特征模型验证问题的特点,并希望能够利用这些特点进一步提高特征模型的验证效率.

References:

[1] Kang KC, Cohen SG, Hess JA, Novak WE, Peterson AS. Feature-Oriented domain analysis feasibility study. Technical Reports, CMU/SEI-90-TR-21, ESD-90-TR-222, Software Engineering Institute, Carnegie Mellon University, 1990.

[2] Kang KC, Kim S, Lee J, Kim K, Shin E, Huh M. FORM: A feature-oriented reuse method with domain-specific reference architectures. Annals of Software Engineering, 2004,5(1):143−168.

[3] Griss ML, Favaro J, d'Alessandro M. Integrating feature modeling with the RSEB. In: Proc. of the 5th Int'l Conf. on Software Reuse. IEEE Computer Society, 1998. 76−85.

[4] Czarnecki K, Eisenecker U. Generative Programming: Methods, Tools, and Applications. Boston: Addison-Wesley, 2000.

[5] Batory D. Feature models, grammars, and propositional formulas. In: Proc. of the Software Product Line Conf. LNCS 3714, Berlin, Heidelberg: Springer-Verlag, 2005. 7−20.

[6] Mannion M. Using first-order logic for product line model validation. In: Proc. of the 2nd Software Product Line Conf. LNCS 2379, Berlin, Heidelberg: Springer-Verlag, 2002. 176−187.

[7] Zhang W, Zhao H, Mei H. A propositional logic-based method for verification of feature models. In: Proc. of the 6th Int'l Conf. on Formal Engineering Methods (ICFEM 2004). LNCS 3308, Berlin, Heidelberg: Springer-Verlag, 2004. 115−130.

[8] Tsang E. Foundations of Constraint Satisfaction. London/San Diego: Academic Press, 1993.

[9] Zhang W, Mei H, Zhao H. Feature-Driven requirements dependency analysis and high-level software design. Requirements Engineering, 2006,11(3):205−220.

[10] von der Maßen T, Lichter H. Deficiencies in feature models. In: Workshop on Software Variability Management for Product Derivation, in Conjunction with the 3rd Software Product Line Conf. 2004. http://www.swc.rwth-aachen.de/opencms/export/

[11] Hu AJ. Techniques for efficient formal verification using binary decision diagram [Ph.D. Thesis]. Stanford University, 1995.

[12] The JDD Project. http://javaddlib.sourceforge.net/jdd/

[13] The SMV System. Carnegie Mellon University, http://www.cs.cmu.edu/~modelcheck/smv.html

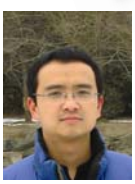
[14] Yang B. Optimizing model checking based on BDD characterization [Ph.D. Thesis]. Pittsburgh: CMU, 1999.

[15] Mei H, Zhang W, Zhao H. A metamodel for modeling system features and their refinement, constraint and interaction relationships. Software & System Modeling (SoSyM), 2006,5(2):172−186.

[16] Czarnecki K, Kim CHP. Cardinality-Based feature modeling and constraints: A progress report. In: Proc. of the OOPSLA 2005 Int'l Workshop on Software Factories. 2005. http://softwarefactories.com/workshops/OOPSLA-2005/

闫华(1984－),男,硕士生,主要研究领域为面向特征的领域分析.

赵海燕(1966－),女,博士,副教授,CCF 高级会员,主要研究领域为软件工程,程序设计语言.



张伟(1978－),男,博士,讲师,CCF 会员,主要研究领域为需求工程,领域工程.

梅宏(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程及软件开发环境,软件复用及软件构件技术,(分布)对象技术,软件工业化生产技术及支持系统,新型程序设计语言.