基于与或树的柔性BOM结构及其自动诊断方法

专业：计算机软件与理论
硕士生：刘裕
指导老师：姜云飞教授

摘 要

物料清单（BOM）是企业产品制造过程中不同部门、不同流程间传递数据的基本形式，是企业集成管理的信息纽带。因此，BOM的结构设计及其数据正确性的维护都是企业信息管理中十分重要的内容。

针对BOM的变型设计问题，本文首先分析了产品的组成成分，然后从产品结构的柔性特征出发，通过对传统树形产品结构的扩展，形式化定义了一种基于与或树的产品结构形式，并给出了相关的BOM数据表设计和面向任务的产品配置方法。事实表明，这种柔性BOM结构不仅可以实现动态的产品结构配置，体现了产品管理的一致性，还能显著地减少产品变型设计中的数据冗余，使统一的BOM数据管理得到保障。

针对BOM的数据维护问题，本文将基于模型的诊断技术与BOM的错误诊断相结合，实现了对产品结构中语义错误的自动诊断。为了使模型诊断技术得以应用，本文定义了BOM的一种诊断模型，给出了产品结构及其产品配置的逻辑描述形式，并引入产品配置样例集合作为系统的期望结果，通过相关的模型诊断算法，从而实现了对BOM的错误诊断。在此基础上，本文还实现了一个诊断系统原型，其测试结果表明，这是一种行之有效的诊断方法。

关键词：物料清单，变型设计，与或树，错误诊断，基于模型诊断

A Flexible BOM Structure Based on AND/OR Tree and Its Automatic Method of Diagnosis

Major: Computer Software and Theory
Name: Liu Yu
Supervisor: Professor Jiang Yun-fei

ABSTRACT

Bill of materials (BOM) is one of the basic forms for data exchange between different departments of enterprise, and between different manufacture processes, which also is the information bridge for integrating enterprises management. The design of BOM structure and the maintenance of BOM data correctness have been a growing interest for researchers.

To address the challenge of variant-design of BOM, this work investigates the composition of product with the required flexibility features, and defines product structures based on the extended AND/OR tree. The design of related BOM form and the correlated product-configuration algorithm are also presented. It is shown that, the flexibility design of BOM can provide dynamic product configuration, and suffice for maintaining the consistency of product management. The flexibility design can reduce data redundancy on variant design, ensuring consistent management of BOM data.

Due to the significant importance of correctness maintenance of BOM, this paper also employs modal-based diagnosis method in error diagnosis of BOM, aiming to identify semantic errors of product structure. To apply the model-based diagnosis technology, this paper presents a diagnosis modal of BOM as well as the logic description of product structure and product configuration, and then works out a solution of BOM diagnosis with relevant computing algorithms by importing product configuration samples as expected outcome. Finally, a prototype of diagnosis system has been introduced as well, whose testing ensures the effectiveness and efficiency of this method.

Key words: bill of materials, variant design, AND/OR tree, fault diagnosis, modal-based diagnosis

第一章　绪论

1.1 引言

在现代制造、加工业中，很多复杂的成套电子设备、机械设备都由数以万计的自制件、外协件、外购件及原材料等各种各样的零部件所组成，为了形成各种型号的产品，要对这些零部件进行合理的组合，这就是产品结构配置。基于这种思想，人们提出了物料清单 BOM（Bill of materials）这个概念。

随着以信息集成为核心的 CIM 技术和企业自动化程度的进一步提高，将产生大量与产品相关的重要数据和信息，如何有效地管理好这些产品信息，以及实现好信息共享和信息集成，成为企业界和学术界有待解决的问题。其中，采用数据格式来描述产品结构的物料清单文件即 BOM，更是集成管理系统的基础，几乎与企业中所有的职能部门都有关系，其主要用途有：

(1) 是计算机识别物料的基础依据；

(2) 是编制生产制造计划，配套和领料的依据；

(3) 可以根据它进行加工过程的跟踪；

(4) 是采购和外协的依据，且可进行物料追溯；

(5) 根据它进行成本的计算，作为报价参考；

(6) 使设计系列化,标准化,通用化。

BOM 不仅是物料资源计划（MRP）系统中重要的输入数据，还是 CIMS/MIS 与 CAD,CAPP 等子系统的主要接口，是财务部门核算成本,制造部门组织生产等的重要依据，其组织格式设计合理与否直接影响到系统的处理性能。因此，根据实际的使用环境，灵活地设计合理且有效的 BOM 是十分重要的。若给 BOM 一个柔性定义，可以更好地适应企业部门的不同需求，有效地减少 BOM 表的数量，从而能更好地消除产品数据的不一致性，促进企业信息集成，提高企业的快速反应能力。

与此同时，正因为在基础数据中 BOM 的影响面最大，对它的准确性要求也最高。一般来说，要使 MRP 系统顺利实施并提供有用信息，要求[1]：

(1) 产品物料清单 BOM 的准确性达到 98%；

(2) 产品物料清单 BOM 的合法性达到 100%；

(3) 库存状态数据的准确性达到95%。

以上三个指标不容易达到但又是必须努力达到的，否则MRPII便会失去意义，同时也会给企业的集成带来很多困难。而BOM通常是在关系数据模型的支持下，以数据表的形式存放，不仅数据量大，而且容易出错，靠人工检查或根据实际中出现的问题再定位错误，将导致很大的工作量且会带来许多不必要的损失。所以，对BOM的正确性检查和诊断也是一个非常有意义的课题。

本文主要针对上述领域，首先对BOM的柔性定义和产品配置问题进行了相关研究，然后把人工智能中的基于模型诊断技术应用到BOM数据的错误诊断中，最后介绍了其系统实现。本文的主要结构安排如下：第一章主要介绍BOM和模型诊断技术的基本概念和背景知识，以及相关的一些研究现状；第二章主要提出一种基于与或树的柔性BOM结构的设计及其产品配置方法；第三章主要阐述一种基于产品结构模型的，对BOM中产品结构错误进行自动的诊断方法；第四章则主要总结本文的工作。

1.2 物料清单（BOM）的介绍

BOM即物料清单，也被称为产品结构表或产品结构树。物料一词在企业中有着广泛的含义，它是所有产品、半成品、在制品、原材料、装配件、协作件、易耗品等与生产有关物料的统称[2]。从物理装配和组成关系的空间角度分析，BOM反映了产品与其构成的部件、部件与其子部件之间的层次关系，是一种代表各部件构成关系的树形结构数据文件，还可包括有关产品及其零部件的编码、规格、材料、质量等方面的信息。BOM可以直观的表示成一种根部在上的倒置树状图，即产品结构树（如图1-1）。

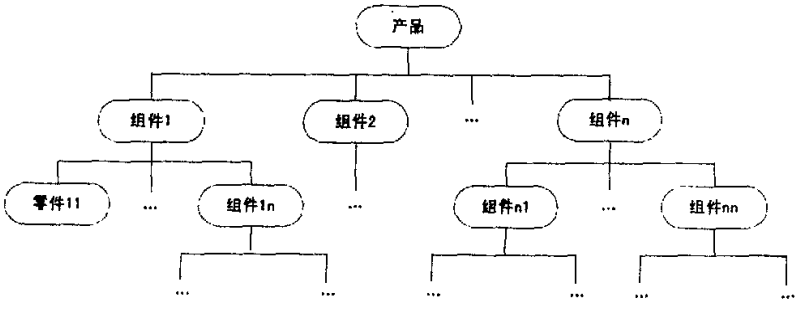

图1-1 产品结构树

这种产品结构树是一个包含一个元素以上的有限集合，其中：

1）有一个特殊的元素是根元素，该元素的代表用户感兴趣的对象，如一个产品、一个装配件等；

2）其余元素又被分成若干产品结构树，他们是其根元素的子树；

3）产品结构树中任一个元素也称节点都由其名称标志、属性及访问特性组成。名称标志是唯一的，用来识别该部件；属性内容是用户定制的，可以包括该部件的一切相关信息；访问特性是对节点的操作控制信息，如读、写权限等。

在企业管理中，产品 BOM 的应用可分为两步[3]：一是在开始实施 ERP 时，由设计部门将企业产品结构、数量信息输入到 PDM 系统或 ERP 的数据库中，形成了设计 BOM（BOM for Design，BOM4D），它是系统启动的基础数据之一。由于这项工作的量非常大，故在这一阶段需考虑系统启动的快速性和系统开始后的数据易维护性。二是系统启动后，根据 MRPII 的思想，将产品 BOM 用于企业的采购、投料、生产、仓库等生产部门。首先，工艺部门收到订单后，根据设计 BOM，制定符合客户要求以及本企业条件的生产加工前的准备工作文件，产生工艺 BOM；供应采购部门根据工艺 BOM 以及库存情况，确定需要购买的零部件清单；财务部门根据工艺要求、加工产品的复杂度、购买材料的费用以及产品管理所需的费用计算成本；销售部门以工艺 BOM 为基础，加上客户的需求，制定装箱清单。其在生产管理流程中的应用可如图 1-2 所示：

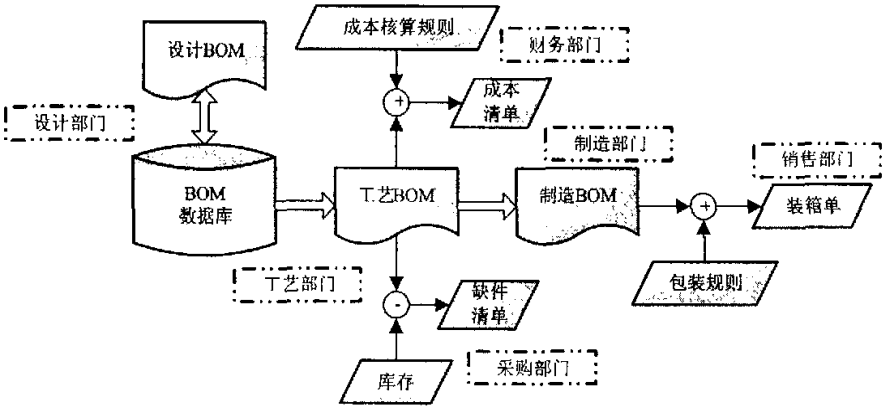

图 1-2　BOM 在生产管理流程中的应用

有关 BOM 的研究工作很多，目前的主要方向有：BOM 在 PDM 和 ERP 中的应用

及其集成管理[4, 5]，BOM 数据的存储和表示方法[6, 7]，基于产品 BOM 的生产制造过程规划和优化[8, 9]，基于 BOM 驱动的工作流配置方法及改进[10]，对 BOM 数据进行相关应用的智能数据挖掘[11, 12]，BOM 数据一致性和规则完备性的检查[13, 14]等等。

1.3 基于模型诊断技术的介绍

基于模型诊断(Model-based Diagnosis)是人工智能领域近年来发展起来的一个活跃的分支，其基本任务是从待诊断系统模型和观测到的现象出发进行推理，根据观测现象与期望结果之间的差异，从而决定出哪些部件产生故障，才能够解释出现的所有差异。

基于模型诊断的方法与传统诊断推理方法的主要区别在于[15]：一方面系统的行为可以被观测到；另一方面，需建立描述系统内部结构和行为方式的模型，并可预言其预期行为。如果系统的实际行为（观测）和期望行为（预言）出现了差异，就产生了诊断问题。一般来说，模型诊断中使用的模型可用标准 AI 技术构造，如谓词逻辑、框架、约束等。实现模型化技术的诊断算法也是基于标准 AI 技术的，如定理证明、启发式搜索、定性仿真和贝叶斯网络等。

总的来说，基于模型诊断的基本思想可如图 1-3 所示[16]：根据组成元件与元件之间的连接关系建起待诊断系统的模型（如结构、功能、行为），这种模型通常用一阶逻辑语句来描述，根据系统的逻辑模型以及系统的输入，可以通过逻辑的推理理论推导出系统在正常情况下的预期行为，如果观测到的系统实际行为与系统预期行为有差异，就说明系统存在故障，利用逻辑推理理论，便能够确定引发故障的元件集合。

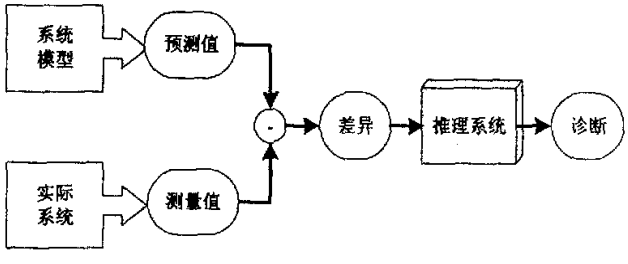

图 1-3 基于模型诊断的基本思想

在理论上，基于模型诊断过程已经形式化，根据不同的逻辑定义，基于一致性诊断[17]和溯因诊断[18]是其中两个具有代表性的研究流派。目前各种各样的基

于模型诊断的理论，对诊断知识的表示与描述方法也正在不断发展，如 Console 等将诊断看成是带一致性约束的溯因问题[19]，并提出了一个统一的定义，使得不同的逻辑定义都可在这一定义中进行表示，并能够进行比较；欧阳丹彤等提出的关于广义因果理论的基于模型诊断与中心诊断的方法[20]，以及对扩展的因果理论中的测试问题的研究[21]；Geffner 等[22]将系统行为的因果知识用信念网表示，并用 Bayes 推理来实现诊断。正是由于这些理论和方法的不断出现促进了基于模型诊断理论的发展，并使其在实践中得到越来越广泛的应用。

实际应用中，比较经典的诊断系统是 GDE 系统[23]和 DART 系统[24]，它们将测试和诊断很好地结合了起来。在应用领域方面，基于模型诊断系统最初主要应用于硬件电路故障诊断，用来诊断电子电路系统[17]等硬件设备中的错误行为，及发现 VHDL[25]等硬件设计描述语言中的逻辑错误等。后来类似的诊断框架同样地被应用到逻辑程序[26]，函数型程序[27]等软件方面的错误诊断，以及 Java[28]等高级语言程序的诊断。此外，基于模型诊断的系统目前已逐渐地被应用到各行各业。如在工程领域，Mostermen 等[29]成功地把动态系统的监视与诊断相结合的方法应用到反应堆辅助冷却循环系统的故障上；在医学领域，TrauAID[30]系统可以帮助医生诊断与修复外伤治疗；在经济领域，Hamscher[31]成功地开发出决策系统 CROSBY，把基于模型诊断理论应用于财会审计上等等。

总的来说，与传统的诊断方法相比，基于模型诊断的技术具有以下优点[32]：

(1)具有发现系统设计者所不能预见的故障的能力；

(2)模型可重用和易于维护，在结构调整时，以前的模型可以继承下来；

(3)知识获取相对容易，可利用系统的设计和技术规格信息；

(4)较强的解释能力，推理系统具有较强的系统独立性。

但是，由于基于模型诊断的方法需要对实际系统进行模型抽象，目前在模型的构建、选择、推理，应用的广度，处理复杂行为等方面还有许多待解决的问题[33]。其中，如何选择合适的模型是至关重要的，因为如果模型正确，那么观测和预言行为之间的所有差异都可以通过具有因果关系的路径往回追踪，一定能找到故障部件。但从理论上看，模型永远不是完备的，只是系统的抽象和近似，因此诊断系统必须在复杂性和完备性衡量。正因为如此，目前基于模型诊断系统还主要应用在电子电路方面，在其他如机械制造，水利设备等方面所做工作较少。

1.4 相关研究工作的介绍

由于本文主要针对基于模型诊断技术在BOM错误诊断中的应用，并对BOM的柔性定义和配置问题进行了一些探讨。下面主要介绍一些相关的研究工作和研究成果，以及本文从中得到的借鉴和参考：

为了使企业产品的设计能适应客户定制的要求，实现产品结构的动态配置，Wolfram[49]通过扩展传统关系数据库的操作规则，设计了一种由规则驱动的产品配置器，这种配置器能根据所定义的规则库，在查询时产生满足各种变型设计要求的产品结构。由于产品部件的变型组合是由外部规则约束的，这种方法可以显著地减少 BOM 中的数据记录量，并可以有效地减少数据冗余和数据不一致的情况。Alexander[50]也提出了一种实现产品结构的动态定制的配置器，该配置器主要根据一组描述客户需求的输入参数来实现产品结构的变型设计，具体方法是通过BOM数据与输入参数之间的计算与逻辑比较，便可构造出符合要求的产品零部件组合。文献[51]则把产品配置分解为模型库和产品定制库两个层次结构，并将相似产品的相关信息加以综合，提取其中的结构信息构建模型库，然后将具体产品与模型产品的转换信息集成在定制库中，使产品结构和组成零部件在一定程度得到分离，极大地降低了产品结构数据的记录量。由此可见，产品变型设计的关键在于降低变型设计中相似产品结构间的数据冗余，上述的方法实质就是把变型结构从产品数据中分离出来，或以规则，或以参数、模型等方式来实现动态定制。

BOM系统另一个重要的内容就是对BOM数据的维护和检查。文献[14]阐述了MRP系统运算中数据完备性和准确性的要求，指出了BOM合法性检查的主要内容及其错误的类型。文献[52]则给出了诸如BOM表死锁（循环引用）等相关BOM语法错误的检查算法。文献[13]针对BOM的一致性维护问题，形式化定义了部件关系、部件约束以及部件配置等基本概念，并用逻辑谓词表示的约束关系来检查BOM数据的一致性。总之，目前BOM检测的研究还是主要在数据完备性和一致性等语法错误的检查方面，在语义错误检查乃至诊断方面所做工作不多。

在模型诊断技术方面，文献[54]进一步刻画了基于模型的诊断问题的分解，为具有树型结构的系统诊断效率的提高提供了理论依据。文献[55]研究了对知识库的一致性诊断问题，采用了正例和反例来检测知识库中的配置冲突，从而实现对知识库中规则不一致性的诊断，给本文的研究工作提供了很多参考依据。

第二章　基于与或树的 BOM 设计

2.1　引言

物料清单 BOM 实质上是一种描述部件装配关系的结构化零件表，其中包括所有的子装配件、零件、原材料清单，以及制造一个装配件所需物料的数量等。要想用计算机来管理企业的这些繁多而又关联的物料，首先要使系统能够知道企业所制造产品的结构，为了便于计算机识别，还必须把产品结构图转换成规范的数据结构，而这种规范的数据结构就是所谓的 BOM 结构。

2.1.1 常见的 BOM 结构

一、多层 BOM 结构（MBOM）

多层 BOM 结构[5]（以下简称为 MBOM）采用“单父-多子”的数据结构，它详细记录了产品的结构信息，即便是同样的零部件结构，只要存在于不同的装配关系中，也要再记录一次。在关系数据库中，其结构可如表 2-1。

表 2-1　多表 BOM 结构

名称	类型	含义
Product	Integer	产品代码
Code	Integer	部件代码
Version	Char	部件版本
Sequence	Integer	顺序码
Hierarchy	Integer	层次码
Unit	Char	计量单位
Amount	Numeric	单位用量

BOM 表中的顺序码表示该部件出现在该产品的产品树中的行位置；层次码表示该部件出现在产品树的列位置。层次码从 1 开始，1 表示最终产品的层次，大于 1 的层次码表示该部件为其紧挨着的，层次码比它小 1 的部件的子部件。由此，在查询该产品结构时，只需以要查的产品代码为条件按顺序码排序即可一次查出产品的完整结构。

MBOM 结构的特点是产品间结构不互相影响，各个产品之间的数据记录没有交

叉。查询也简单快捷，产品分解时只需根据产品的 ID 查询出最底层零件（需采购的原材料）的 ID 和数目即可。

但 MBOM 也存在很多不足之处，主要有：

(1) 顺序号的维护非常困难。例如，在表中插入一个零件，其后的零件号也必须相应的变化；

(2) 数据冗余大，不同装配关系中相同的零部件结构需要重复定义；

(3) BOM 结构的设计必须按从部件到子部件再到零件，由大到小的顺序完成。

二、单层 BOM 结构（SBOM）

单层 BOM 结构[5]（简称 SBOM）采用“单父-单子”的数据结构，只是记录了各个父件和子件之间的对应关系，特点是相同的零部件装配关系只需记录一次，其中表 2-2 就描述了一种单层 BOM 的结构。

表 2-2 单层 BOM 结构

名称	类型	含义
Code	Char	部件代码
Version	Char	部件版本
Node	Char	节点代码
FatherNode	Char	父节点代码
Unit	Char	计量单位
Amount	Numeric	单位用量

SBOM 表中部件代码和节点代码为主关键字，节点代码把不同部件映射到产品结构树中的对应节点，通过父节点标识指向其所属的父部件。由于是单层 BOM，产品树的生成需要使用递归的查询方法[7]，即首先查找产品第一层子项，再查找所有以第一层子项为父项的第二层子项，如此递归查找直至其下再无子项为止。

在实际使用中，单层 BOM 具有以下优点：

(1) BOM 表修改简单，相同的装配关系只需修改一次；

(2) 数据库冗余小，对相同的层次结构只定义一次，大大节省了存储空间。

(3) 清晰定义了产品结构的树形关系。

但它也有以下缺点：

(1) 容易产生 BOM 表中的层次循环错误；

(2) BOM 生成速度比多层 BOM 要慢些。

三、复合式 BOM 结构（CBOM）

复合式 BOM 结构(CBOM)[3]综合了多层 BOM 和单层 BOM 的构造方法，用以平衡 BOM 数据维护和运行效率的问题。在结构上，CBOM 和 SBOM 类似，仍然采用“单父-单子”的数据结构，只是在每个子件的表示中又加上了所对应的最终产品。复合式 BOM 在构造时采用单层 BOM 的构造方法，而在 BOM 分解时，又与多层 BOM 的方法相同，其实质就是通过增加数据冗余项来综合 SBOM 和 MBOM 的构造方法，在形式上同时具有设计用 BOM 和生产用 BOM 的特点，满足了在一个系统中只使用一套 BOM 的基本思想。但是，由于复合式 BOM 存在着数据冗余，数据更新的一致性问题仍然存在。

由于计算机硬件技术的飞速发展和 CPU 运算速度的不断提高，BOM 的分解速度和运行效率在大多数情形下已不是瓶颈问题。因此，目前大多数 BOM 结构设计依然是以单层 BOM 结构为主。

2.1.2 BOM 的变型设计问题

激烈的市场竞争迫使企业的经营管理强调“以客户为中心”，提供“个性化服务”，以期实现满足客户个性化条件下的产品规模生产，实现产品定制，从而在降低成本前提下提高企业的竞争力，这就提出了 BOM 的变型设计问题。

根据设计活动中的创造性程度，设计行为可以分成原始的创新设计、适应性设计和变型设计，其中变型设计是指在产品的系统工作原理、功能和结构不变的情况下，在预定选定的范围内，对现有产品的局部组成部件的属性变更。如同样一种尺寸的自行车，除了颜色不同，其余设计都相同，对客户的这一需求，映射到自行车这种产品，体现在其颜色属性的变化[2]。在这类设计中，新产品的定义完全可以从已有的设计模板获取。随着大批量生产模式转向小批量生产模式甚至单间生产模式，此时，变型设计在设计活动中占有主导地位。

如果 BOM 管理的方式是为每种产品建立一个这 BOM 表，这种方式在面向客户需求的变型设计中，倘若要将类似产品的所有明细表的全部信息储存起来，数据量就会猛增。例如，一辆完整的自行车由 1000 多个零件组成，只要其中一个零件的尺寸、颜色、材料及规格等不一样，该自行车作为产品的明细表就与其他产品的明细表不同，如果有 5 种用户选择的参数，如颜色(10 种)、车轮大小(3 种)、

车身大小(3种)、车铃式样(3种)、车座式样(5种)、脚踏式样(2种)，就可能产生出10*3*3*3*5*2=2700种不同的选择方案，如果新增一种自行车的速度是否为变速的属性，那么将产生种不同的2700*2=5400种不同的选择方案。虽然在实际的选择中不是所有的组合都会被选中,但如果每使用一种变型产品就存储一种独立的BOM，只要多考虑几种变型方式就会使BOM文档出现数据爆炸现象[34]。

2.1.3 BOM的变型设计要求

既然客户个性化的大规模定制已经成为企业面临的一项重要课题，产品供应方与消费者之间也正在建立这么一种新型的关系,那么企业产品设计和生产组织都必须适应这种变化，这就对BOM的构造提出了更高的柔性要求，主要体现在如下几个方面：

(1) 客户定制的快速响应；

(2) 网上定制的可支持性；

(3) 较高的运作速度；

(4) 产品结构的直观化、形象化；

(5) 产品设计的客户需求化。

然而，目前在作BOM的柔性设计时，通常都采用规划性BOM（Planning BOM）的形式，即将特性件和可选件都建立在该产品的BOM中，并依虚拟件来计算各种特性件、选用件的供应计划[35]。这种方法在实现上首先利用虚拟件定义各主料和特性件（可替换件）、选用件的适当组合；然后按照定义的规则以及可选件表、可替换件表中的内容,系统自动在基本物料表中选择符合条件的零部件来替换产品结构上的节点，形成产品结构树。然而，利用这种规划性BOM作产品的变型设计时，存在以下问题：

(1)没有一种清晰、直观的数据结构反映产品结构中替换件或可选件的特性及其组合关系，具体表现为在产品结构树中难以反映替换件或可选件的组成；

(2)在规划BOM设计时，需要定义基本物料（零件）表、特性件表、可选件表等来区分不同的零部件属性[4]，产品的零部件缺少统一的表示形式和存储结构,造成了对BOM表中产品结构或零部件关系修改的困难,且容易导致不一致性；

(3)相关的产品变型特性难以用形式化的语言描述。

显然，利用规划性 BOM 表示变型设计的方法与上面提到的新型产品定制的柔性要求还有一定的距离。更重要的是，为了适应快速的市场变化和需求，产品结构在面向企业的任务定制方面的要求也愈加迫切。例如，一件产品或半成品可以由不同厂家或型号的零部件的不同组合构成，如何根据产品结构提取出成本最低的零部件组合。又譬如，不同零部件的采购周期、装配件的生产周期都各不相同，如何从中选择出限定工期内可完成的产品结构……类似的需求还有很多。

鉴于这种需求，针对 BOM 表示的产品结构柔性的提高，下一节本文将开始阐述一种基于与或树模型的 BOM 结构。

2.2 基于与或树的 BOM 结构

2.2.1 产品结构与或树

与或树[36]是人工智能中用于表示问题归约及其求解过程的一种方法，它把复杂的多阶问题分解成多个子问题。下面利用与或树的形式来表示产品的结构组成，使其可以表示产品部件的替换关系和可选关系。假设产品P可由部件P_1、P_2组成，那么他们之间存在下面两种基本关系：

(1)产品P由两个子部件P_1、P_2的组合制造而成，则称P_1和P_2之间存在“与”关系，如图2-1所示的“与”树的形式，用逻辑公式可表示为：$P\rightarrow P_1\wedge P_2$。

(2)产品P可由子部件P_1和P_2中的其中一个制造而成，则称P_1和P_2之间存在“或”关系（用圆弧线连接[1]），如图2-2所示的“或”树形式，用逻辑公式可表示为$P\rightarrow(P_1\wedge\neg P_2)\vee(\neg P_1\wedge P_2)$。

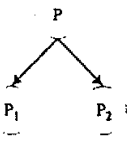

图 2-1 与树

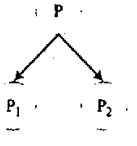

图 2-2 或树

定义2.1(产品与或树) 本文定义的产品结构与或树是一个六元组G=(O, A, T, N, E, r_0)。其中O是“或”节点的兄弟关系集合；A是“与”节点的兄弟关系集合；

[1] 与一般的与或树定义不同，这里“或”关系是 N 中选一的选择关系，为了突出这种部件间的替换选择关系，用圆弧线将相关的“或”结点连接。

T是终端叶节点集合，N是非终端叶节点集合；E是树的边集即N×(T∪N)的子集，其元素是由父节点指向子节点的有向弧；r_0是树G的根节点；若边含权值，则从节点n到节点m的边代价记为cost(n, m)。

显然，对一棵产品结构与或树而言，根节点r_0表示用户感兴趣的对象如一个产品等。“或”节点集合O代表可替换件集；“与”节点集合A代表必选件集；终端叶节点T代表原材料或零件，是产品的最终组成成分；非终端叶节点N表示概念件（或虚拟件），代表产品的半成品和装配件。边集E表示产品结构中的父项子项关系，即父项部件由哪些子项部件直接组成。显然，产品与或树本身是一种递归定义，通过定义不同的与或关系，可以表示出产品部件的不同组合结构。

例 2.1 产品 a 由装配件 b 和装配件 c 所组成，而装配件 c 既可由零件 d 和零件 e 组合而成，又可以由零件 d 和 f 组合而成，即 O={{e, f}}，A={{b, c}, {d}}，对应的逻辑公式为{a→b∧c, c→d∧((e∧¬f)∨(f∧¬e))}。这种产品结构就对应于图 2-3 所示的与或树。

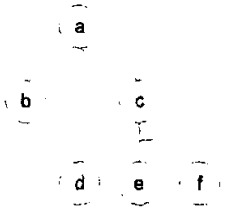

图 2-3 产品与或树示例

定义2.2（子与或树）若与或树$G'=(O', A', T', N', E', r_0')$是$G=(O,A,T,N,E,r_0)$的一个子树，可表示为 $G'\subseteq G$，则G′ 对以下条件均成立。

(1) $\forall o'\in O', \exists o\in O$有$o'\subseteq o$，且$\forall a'\in A', \exists a\in A$有$a'\supseteq a$；

(2) $T'\subseteq T$，$N'\subseteq N$，$E'\subseteq E$，且$r_0'=r_0$；

(3) $\forall n\in N'$，则$\forall m\in\{x\mid(n,x)\in E\}$即对n在G中的所有孩子节点m，同时满足下面两个条件：

(I) 若$\exists\alpha\in A$，使得$m\in\alpha$，则$\forall k\in\alpha$，有 $(n,k)\in E'$；

(II) 若$\exists\beta\in O$，使得$m\in\beta$，则$\exists k\in\beta$，有 $(n,k)\in E'$。

从以上定义可知，子树关系“$\subseteq$”是一种偏序关系。若G′是G的子与或树，则G′和G是同构的，且G′必须包含G中所有的“与”关系节点，而位于G中同一“或”关系集的节点，G′应至少包含一个。对产品结构与或树而言，它的一棵子与或树就对应于一个有效的产品结构，因为其包含了这个产品的所有必要成分。本质上，产品结构与或树在语义上扩展了传统上的产品结构树，增加了在孩子分支上的选择语义，若一棵产品结构与或树不包含任何“或”关系，其就相当于一棵传统意义上的产品结构树。

例 2.2 例2.1中的与或树(图2-3)就包含了如图2-4所示的两个子与或树。

图 2-4 例 2.1 的子与或树

定义2.3（极小子与或树）若与或树$G'\subseteq G$，且不存在$G''\subseteq G'$，则称G'为G的极小子树。

显然，极小子与或树对应的是产品的极小产品结构即最基本的组成成分。一棵与或树可能有多棵极小子树，类似地，一个产品也可能有多个基本组成结构。根据定义2.2和2.3，也容易有以下推论。

推论2.1 极小子与或树必定不包含“或”关系节点即$O=\phi$。

由定义可知，若存在“或”关系节点，“或”关系节点集合应至少包含两个节点，则删去其中一个节点所成的与或树也必定是原与或树的子树。因此，极小子与或树的“或”关系节点集必定为空。

定理2.1 一个产品的所有的变型设计结构都可以用一棵产品结构与或树所表示。(完备性)

证明：用数学归纳法，验证任意层次的产品结构均可由与或树表示。

(1)当产品结构只有 1 层时，显然对应于只有一个根节点的产品结构与或树。

(2)假设当产品的组成结构为 k(k>1)层时，均可由一棵产品结构与或树表示。

那么当产品组成结构为 k+1 层时，由假设可知，前 k 层的产品结构均可表示为一棵产品结构与或树。在此基础上，我们构造 k+1 层产品结构的与或树：

不失一般性地，我们假设 a 为其第 k 部件中的任意一个，若 a 不由任何一个 k+1 层的子部件所构成，则 a 作为与或树的叶子节点。若 a 的组成包含 k+1 层的子部件 b，而 b 是属于没有可替换件的必选件，则在与或树增加一个“与”节点 b，其父亲节点为 a；如果 b 有若干可替换件，则这些可替换件和 b 一起构成一个“或”节点集，其父亲节点同样为 a；如果 b 是可选件，则 b 可与“空”节点一起构成一个父亲节点为 a 的“或”节点集。

因此，k+1 层的产品结构就可以由构成的产品结构与或树所表示。

2.2.2 BOM表的数据设计

产品结构与或树可以看作由结构相似的产品结构树所耦合而成，其实质是将相同的装配结构抽取出来，用相应的与或关系来表示不同的变型成分。从数据库设计的角度，可以把基本(必选件)表、可替换件表、可选件表（可选件可通过部件与一个空节点存在“或”关系来表示）等统一到同一种BOM表结构，从而可以方便的对产品结构进行修改和定制，减少BOM表设计的复杂度和所需的存储空间。下面表2-3和表2-4就描述了一种基于与或树BOM的基本结构：

表2-3 与或树节点表

名称	类型	含义
TreeID	Integer	所属树标识
NodeID	Integer	节点标识
ParentID	Integer	父节点标识
RelationID	Integer	所属关系标识
Code	Char	对应部件代码

表2-4 部件属性表

名称	类型	含义
Code	Char	部件代码
Name	Char	部件名称
Version	Char	部件版本
Type	Char	部件类型
Unit	Char	计量单位
Amount	Numeric	单位用量

其中，与或树节点表反映的是产品与或树的结构，主键是TreeID和NodeID，树标识(TreeID)是节点所属与或树的唯一标识，节点标志(NodeID)是对应节点的唯一标识。父节点标识(ParentID)指向当前节点的父亲节点；关系标识(RelationID)表示节点间的“与/或”关系，属于同一“或”关系集合的节点属于同一关系（即RelationID的值相同），不同的“与”节点及不同的“或”关系节点集合以不同的关系（即不同的RelationID）相区别。例如图2-3所示的产品结构，节点e和f的关系标识RelationID应相同而e和d的关系标识必须不同；部件代码(Code)是外键，是该节点所对应的产品部件的编号。

当需要修改产品的配置结构时，只需要修改反映结构关系的与或树节点表，

更改部件属性则只需修改部件属性表及节点表与属性表间的对应关系。整个产品的结构可由通过部件代码关联节点表和属性表所得到的视图来反映，完整的产品结构关系可以通过递归的方法获得，具体可参考文献[7]。

与或树结构在程序设计语言中可有多种表示方法，下面就是其中一种：

```
struct AO_NODE
{
    int nRelation;          //节点所属的关系
    AO_NODE* pParent;       //父亲节点的指针
    AO_NODE* pFirstChild;   //第一个孩子节点的指针
    AO_NODE* pNextBrother;  //邻近的兄弟节点的指针
        … …
}
```

2.3 面向任务的 BOM 配置算法

2.3.1 基本定义和公式

产品与或树可看作一个超图，节点间用超弧线（或连接符）来连接，求与或子树的过程可等效为求对应与或图解图的过程。如果给与或树的每条边权值赋予一定的意义——如生产成本、生产周期等，那么类似于成本最低、工期最短等面向任务的产品结构配置可看作为寻找权值总和最小的子与或树（解图）。

在搜索子与或树的过程中，还要进行代价值的计算，设子树 G 中节点 n 的代价值为 k(n, G)，则可递归计算如下：

(1) 若 n∈T 即 n 为终端节点，则 k(n, G) = 0；

(2) 若 n∈N 即 n 有连接符指向其孩子节点 $\{n_1, n_2, \ldots, n_c\}$，则

$$k(n, G) = \sum_{i=1}^{c} cost(n, n_i) + \sum_{i=1}^{c} k(n_i, G)$$

其中，$cost(n, n_i)$为从 n 到 n_i的边的权值，实际意义为生产部件 n 所需耗费子部件 n_i的代价，即成本或工期等。

定义 2.4（最佳子与或树）根节点具有最小代价值的子与或树我们称为最佳子与或树，采用 G*表示。

易知，若所有权值为正的话，G*必定为极小子与或树，G*的代价值称为 G

的最小代价值，表示为 h*(G)。考虑权值为正的情况，由定义可知，最佳与或子树必定由全部“与”节点及“或”关系集合中各个代价值最小的“或”节点所组成。设以 n 为根节点的与或树的最小代价值为 h(n)，则有

(1) 若 n∈T 即 n 没孩子节点，则 h(n) = 0；

(2) 若 n∈N，设节点 n 的“与”关系的孩子节点集合为 $\{a_1, a_2, \ldots, a_p\} \in A$，所有“或”关系的孩子节点集合为 $\{o_{11}, o_{12}, \ldots, o_{m1}\} \in O$，$\{o_{21}, o_{22}, \ldots, o_{m2}\} \in O$，…，$\{o_{q1}, o_{q2}, \ldots, o_{mq}\} \in O$，则

$$h(n) = \sum_{i=1}^{p} (\text{cost}(n, a_i) + h(a_i)) + \sum_{i=1}^{q} \min(\text{cost}(n, o_{i1}) + h(o_{i1}), \ldots, \text{cost}(n, o_{mi}) + h(o_{mi}))$$

显然，$h^*(G) = h(r_0)$ （根节点 $r_0 \in G$）。

2.3.2 产品配置的递推算法

实现最佳产品配置就相当于求解对应的最佳子与或树，则产品配置算法可等效为搜索对应与或树解图的方法，求解与或图解图的经典方法[37]有深度优先搜索、广度优先搜索和 AO*算法等，都可以应用于求解产品与或树的最佳子树。

与或图搜索的空间和时间复杂度与与或图本身的结构有关，在文献[38]中有详细的讨论，这里限于篇幅，就不一一赘述了。但产品与或树作为与或图的一种特例，又有着自身的一些特点：①节点数有限，状态空间不大；②是一种树结构，不存在环；③不存在不能解节点。因此，针对这些特点，从简化算法实现复杂性的角度出发，下面提出一种求解最佳与或子树的递推算法：

算法 2.1 (求解最佳与或子树的递推算法)

FIND-BEST-SUBTREE (G，G*)

输入：产品与或树 G

输出：最佳子与或树 G*

BEGIN

G*=ϕ； 置所有 k(n)=0；

WHILE （G 不为空）DO

BEGIN

①找出 G 中所有的叶子节点；

②把这些叶子节点按其父亲节点(ParentID)和所在的关系集合标识(RelationID)排序，得序列 S；

③将 ParentID 值与 RelationID 值都相同（即同一或关系集合）的节点进行比较，保留使其父亲节点的代价值 cost(child ,parent)+k(child)最小的那个节点，把其余在同一或关系下的兄弟节点从 S 中删除；

④计算 S 中各节点的父节点的代价值 k(parent)；

⑤把 S 及其与父节点的边关系加入 G*；

⑥删除 G 中的所有叶子节点及其边关系；

END

END

显然，该算法是基于前面所推导的产品结构与或树的性质的，在实现上要比采用与或图搜索的方法简单，尽管不一定是最佳或最优的搜索策略，但在运行效率要求不高的应用中，仍具有一定的应用价值。分析可知，算法的时间复杂度为 O(nlogn)，空间复杂度为 O(n)，其中 n 为与或树中节点的数目。

2.3.3 应用实例

以上的产品配置方法在一个燃气生产企业的 ERP 项目得到实践。该企业是以从事燃气用具、厨房电器、家用电器的专业制造及经营的集团公司。其产品有燃气灶具、热水器、抽油烟机三大系列 100 多个品种，仅燃气炉具年产销量就超过 200 万台。针对该企业产品线长、品种繁多，新产品设计开发频繁，原料供应来源广泛等特点，我们制定了设计 BOM、试产 BOM、成本 BOM、订单 BOM、制造 BOM 等几种 BOM 表。并以与或树作为载体，把这几种功能各异但有机联系的 BOM 表纳入了统一的框架。例如：在设计阶段，研发部门会利用设计 BOM 的多种方案（即对应的子与或树）生成试产 BOM，进行小批量生产和进行测试。试产成功的 BOM 还会由财务部门和采购部门根据物料成本、供货情况生成相应最佳效应的工艺 BOM（即最佳子与或树），从而确定采购清单，最终生成供生产部门领料、生产用的制造 BOM。在实际应用中，可以根据试产结果、原料价格变动、供应商供货情况自动配置及修改各种 BOM 表，达到了缩短研发和生产周期、节省成本和提高企业竞争力的目的，体现出了柔性和敏捷性。

2.4 本章小结

在集成化的产品管理系统中，表达一致、又有不同形式的BOM结构，对产品数据的集成与共享具有重要意义。为了实现产品数据表达的同一性和可变性，使产品相关语义可以贯穿各个部门，本章提出了一种基于产品结构与或树的柔性BOM结构，既包含了传统的产品结构形式又赋予了其柔性特征，不仅可以实现动态的产品结构配置，还体现了产品管理的一致性，即产品的任何一个数据变化都可反映到产品结构与或树中。采用这种方法可以减少数据的冗余设置，保证了设计、制造、管理、销售各部门对产品数据语义解释的一致性，使统一的BOM管理得到保障。

基于产品与或树的BOM结构比传统的BOM结构具有更多的柔性特征，不仅可以满足产品定制的变型设计要求，还可实现面向任务的产品配置需求，本章内容在这方面的贡献有：

(1) 形式化定义了基于与或树的产品结构形式；

(2) 给出了基于产品结构与或树的BOM设计方法及其数据表实现；

(3) 在上述基础上实现了一个面向任务的产品结构配置算法；

理论上，产品结构与或树中的节点关系都可与一阶谓词公式相对应。在逻辑演算中，与或树常用于逻辑公式的归结和反演，利用与或树所对应的谓词公式来实现对BOM表逻辑的说明性表示，从而引入推理和演算等逻辑方法，实现对BOM表结构的正确性检测和错误诊断，将是本文下一章所要阐述的主要内容。

第三章　BOM（产品结构）的自动诊断

3.1 引言

由于 BOM 在企业集成管理中起着枢纽的作用，它的稳定性和可靠性将直接关系到企业产品生产的效率和质量，所以 BOM 数据必须精确、完整。然而，由于实际企业生产中涉及的数据众多，牵涉面较大，不一定能很好地对所有的数据进行详尽而准确的整理和输入，这就需要对 BOM 数据的正确性进行检验。

由于牵涉到的数据量非常大，传统人工核对与检查的方法既费时又费力，所以出现了 BOM 表数据维护与检验的研究课题。但目前的工作基本上还是集中在对 BOM 结构性错误的语法检查方面，如部件间循环引用[13]、合法性检查（数据输入是否符合设定的规则）[14]，完整性检查（是否存在空指向，存在多余部件）[14]等等，很少针对于 BOM 结构语义的检查。例如，若图 3-1 所示的产品结构在 BOM 中被错误设定成如图 3-2 的形式，但图 3-2 所示的结构仅从语法角度来看显然也是一个“正确”的产品结构。在理论上，仅仅对 BOM 表的数据进行一致性、完备性等语法性检查是不能验证类似 BOM 语义的正确性的。

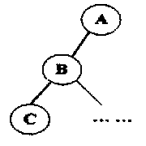

图 3-1 正确形式

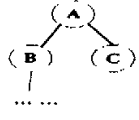

图 3-2 错误形式

此外，目前生成和维护 BOM 的主要方式还是通过二维表格以及装配关系的父子表格编辑器[39]，这种方式虽然可以很好地表达产品设计中静态的 BOM 信息，但要跟踪产品的动态生成过程就比较困难了，尤其是在产品结构错误的定位及修正中，需要从成千上万条数据记录中找出引起错误的相关记录。相对而言，语法性错误的定位还比较简单，因为引起错误的往往就是非法记录本身（如引起循环引用错误的记录就存在部件环路中，引起空指向错误的记录就是空指向部件本身的记录等）。而对于语义错误，就要复杂得多，首先是引起错误的可能因素很多，其次是产品结构及其部件间的关系又往往错综复杂，使得错误往往难于被发现。

因此，如何定义 BOM（产品结构）的语义描述，排除无关的部件，找出可能引起相关语义错误的部件集，从而诊断出引起错误的原因以及相关的部件装配关系，将是本章重点探讨的内容。

3.2 BOM 的诊断模型

3.2.1 基本思路及模型定义

参考程序测试的方法，为了实现对 BOM 表的检验及错误诊断，同样可以给产品 BOM 表引入配置样例（Sample）。通过样例，相当于给出了 BOM 系统的期望行为，如果分析到配置样例（期望）与实际 BOM 结构存在差异，就说明 BOM 表中存在错误，从而利用逻辑推理来确定可能引发错误的部件装配关系集合（对应于 BOM 表中的数据项记录集），这就是对 BOM 进行错误诊断的基本思路。

事实上，引入配置样例也具有一定的现实意义。人们在做产品 BOM 设计的时候，往往会有一些经典的产品配置方案，这些必须要与实际 BOM 结构相一致的配置方案可以称之为正例。另外，有时候也有一些不允许出现的配置方案或者零件组合，这些不允许出现的方案或者组合可以称之为反例。通过正例集合和反例集合，一方面可以测试实际 BOM 结构的正确性，另一方面也可以根据其与 BOM 结构的逻辑差异来判断可能引起错误的部件关系集合。下面就通过一个例子来说明正例、反例和与或树 BOM 结构的关系。

例 3.1 假设一个产品结构与或树如图 3-3 所示，若用“=”表示构成关系，“+”表示零件的组合关系，则根据定义 2.1，符合该 BOM 结构的正例有 A=D+E，A=C，A=F，C=G 等等；不符合该 BOM 结构的反例有 A=D，A=B+C，C=F+G，A=G+H 等等。

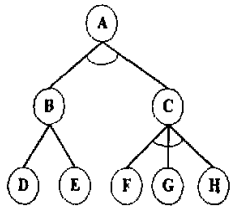

图 3-3 产品结构例

定义 3.1（BOM 的诊断问题）假设 BOM 的诊断问题是一个四元组（PSD, COMP, E^+, E^-），其中 PSD 是描述产品结构的一阶逻辑语句集，COMP 是有限的零部件集，E^+是产品配置的正例集，E^-是产品配置的反例集。

PSD (Product Structure Description) 是 BOM 的内容反映，代表的是产品结构模型及其部件转配关系的逻辑性描述。COMP (Components) 是产品结构中所有零部件（装配件、零件、原材料等）的集合，是一个有限的常量集。显然，在 PSD 中出现的逻辑文字项必须对应于集合 COMP 中的元素。E^+对应于合法的产品配置实例，E^-则对应于非法的产品配置实例，产品配置实例也都可以是用一阶逻辑语句来描述的。

类似于其它模型诊断问题中的观察集，在这里设立正例集 E^+和反例集 E^-的主要目的是提供辅助诊断的必要信息。E^+的目的在于检查已有的产品结构是否可以产生正确的产品配置且不引起冲突，若有产品配置实例 $e^+\in E^+$与产品结构描述 PSD 是不一致的，这就表明当前的产品结构有可能缺少一些必要的组成成分或者当中存在装配关系错误。相反，若有产品配置实例 $e^-\in E^-$与产品结构描述 PSD 是一致的，这就说明已有的产品结构存在着不合法的装配关系。显然，对一个正确的产品结构来说，$PSD\cup E^+\cup\neg E^-$应该是一致的。（这里$\neg E^-$表示集合$\{\neg e|\, e\in E^-\}$）

定义 3.2（配置的一致性）给定一个产品配置实例 e 以及一个产品结构 PSD，称 e 与 PSD 是一致的当且仅当 $e\cup PSD$ 不存在逻辑矛盾。

对于一个配置实例 e 而言，e 不一定必须包括产品结构中的最终产品，其组成成分也不一定是原材料或者零件，它也有可能仅仅包括产品结构中的局部组成成分，这种情况可称之为局部配置。比如在例 3.1 中，A=C（A 由 C 构成）或 B=D+E（B 由 D 和 E 构成），都是允许的产品配置实例。

定义 3.3（BOM 的诊断结果）对一个诊断问题 P（PSD, COMP, E^+, E^-），如果存在一个逻辑语句集 $DS\subseteq PSD$ 满足：

(1) $\forall e^+\in E^+$，　PSD-DS$\cup e^+$ 是一致的；

(2) $\forall e^-\in E^-$，　PSD-DS$\cup\neg e^-$是一致的；

且 DS 的所有真子集 $DS'\subset DS$ 均不满足上列条件，则称 DS 是 P 的一个诊断，DS 所对应的 BOM 表数据项或部件装配关系就是一个诊断结果。

显而易见，对一个 BOM 诊断问题而言，如果其正例 E^+和反例 E^-不存在自我矛盾的话，其诊断是必定存在的（在最坏情况下 DS=PSD）。

3.2.2 产品结构的逻辑描述

要对 BOM 进行诊断，首先就是要用逻辑语言来描述 BOM 即产品结构，以及定义部件间的装配关系和产品配置关系等。

定义 3.4（存在关系）对部件 $c\in COMP$，用逻辑常量 c 表示该部件在产品结构中存在，常量的非$\neg c$ 表示该部件在产品结构中不存在。

定义 3.5（装配关系）对部件集 COMP，装配关系 AR= { (c, c′) |c, c′∈COMP, 且 c 是 c′的上一层的部件}。

装配关系实质上就是产品结构中的父子组成关系，即产品结构树中的父亲节点和孩子节点之间的关系。因此，还可以有：

(1) 部件 c 的所有父部件集合 $PS(c) = \{ c' \mid c' \in COMP, (c', c) \in AR\}$；

(2) 部件 c 的所有子部件集合 $CS(c) = \{ c' \mid c' \in COMP, (c, c') \in AR\}$。

定义 **3.6** （必选子部件）对部件 $c \in COMP$，若其子部件 $c' \in CS(c)$对 c 来说是必须的，则称 c'是 c 的必选子部件。用 MCS(c)表示 c 的必选部件集，则

$MCS(c) = \{ c' \mid c' \in COMP, (c, c') \in AR$,且 c' 是 c 必须的$\}$

显然 $MCS(c) \subseteq CS(c)$，就产品结构与或树而言（定义 2.1），MCS 对应于其中的与节点集，MSC(c)表示的就是节点 c 的“与”关系的孩子节点集。

对 c 而言，其必选子部件的存在关系用逻辑语句可表示为：

$c \rightarrow c_1 \wedge \ldots c_i \wedge \ldots c_h$ 其中 $c_i \in MSC(c)$，$h \geq 0$ 表示 MCS(c)中元素的个数。

化为标准子句形式[36]后，其对应的逻辑子句集（用 MCSL 表示）为：

$MCSL(c) = \{\neg c \vee c_1, \ldots, \neg c \vee c_i, \ldots, \neg c \vee c_h\}$

定义 **3.7** （可替换子部件）对部件 $c \in COMP$，若其子部件 $c_1, c_2, \ldots, c_h \in CS(c) \cup \{\phi\}$对 c 来说是可替换的，即 c 可由其中之一构成，则称 c_i 是 c 的可替换子部件，用 $SCS(c) = \{c_1, c_2, \ldots, c_h\}$表示该可替换部件集，显然 $SCS(c) \subseteq CS(c)$。

同样对 c 而言，c 的一个可替换子部件集的存在关系可用逻辑语句表示为：

$$c \rightarrow (c_1 \wedge \neg c_2 \wedge \ldots \wedge \neg c_h) \vee (\neg c_1 \wedge c_2 \wedge \ldots \wedge \neg c_h) \vee \ldots \vee (\neg c_1 \wedge \neg c_2 \wedge \ldots \wedge c_h)$$

化为标准子句形式后，其对应的逻辑子句集 SCSL(c)为：

$$SCSL(c) = \{\neg c \vee c_1 \vee \ldots \vee c_h, \neg c \vee \neg c_1 \vee \neg c_2, \ldots, \neg c \vee \neg c_i \vee \neg c_j, \ldots\} \quad (i,j \leq h, i \neq j)$$

实际上，c 的可替换部件集可以有若干个，一个 c 的可替换部件集对应于产品结构与或树中节点 c 的一个“或”关系孩子节点集。用 ASCS(c)表示 c 的所有可替换子部件集合，则 $ASCS(c) = SCS_1(c) \cup SCS_2(c) \ldots \cup SCS_n(c)$, $n \geq 0$。

结论 **3.1** $CS(c) = MCS(c) \cup SCS_1(c) \cup SCS_2(c) \ldots \cup SCS_n(c)$. $(n \geq 0)$

为了方便，这里使用符号“| |”表示集合元素的个数，即用| CS(c) |表示集合 CS(c)中的元素个数...如此类推。那么也有

$$|CS(c)| = |MCS(c)| + |SCS_1(c)| + |SCS_2(c)| \ldots + |SCS_n(c)|. \ (n \geq 0)$$

结论 **3.2** 对部件 $c \in COMP$，假设 c 的必选子部件集为 MC，可替换子部件集分别为 $SC_1, SC_2, \ldots, SC_k$ $(k \geq 0)$，则 c 与子部件间装配关系的逻辑形式可表示为：

$c \rightarrow mc_1 \wedge \ldots mc_i \wedge \ldots \wedge mc_{|MC|}$

$\wedge((sc_{11} \wedge \neg sc_{12} \wedge \ldots \wedge \neg sc_{1i} \ldots) \vee \ldots \vee (\neg sc_{11} \wedge \neg sc_{12} \wedge \ldots \wedge sc_{1i} \ldots) \vee \ldots)$

$\wedge((sc_{21} \wedge \neg sc_{22} \wedge \ldots \wedge \neg sc_{2i} \ldots) \vee \ldots \vee (\neg sc_{21} \wedge \neg sc_{22} \wedge \ldots \wedge sc_{2i} \ldots) \vee \ldots)$

$\ldots \ldots$

$\wedge((sc_{k1} \wedge \neg sc_{k2} \wedge \ldots \wedge \neg sc_{ki} \ldots) \vee \ldots \vee (\neg sc_{k1} \wedge \neg sc_{k2} \wedge \ldots \wedge sc_{ki} \ldots) \vee \ldots)$

其中，$mc_i \in MC$, $sc_{1i} \in SC_1$, $sc_{2i} \in SC_2$, ... , $sc_{ki} \in SC_k$。将其化为标准子句形式后，描述部件 c 装配关系的逻辑子句集 ACL(c)就为

$ACL(c) = \{ \neg c \vee mc_1, \ldots, \neg c \vee mc_i, \ldots, \neg c \vee mc_{|MC|},$

$\neg c \vee sc_{11} \vee \ldots \vee sc_{1i} \vee \ldots, \neg c \vee \neg sc_{11} \vee \neg sc_{12}, \ldots, \neg c \vee \neg sc_{1i} \vee \neg sc_{1j}, \ldots,$

$\neg c \vee sc_{21} \vee \ldots \vee sc_{2i} \vee \ldots, \neg c \vee \neg sc_{21} \vee \neg sc_{22}, \ldots, \neg c \vee \neg sc_{2i} \vee \neg sc_{2j}, \ldots,$

$\ldots \ldots$

$\neg c \vee sc_{k1} \vee \ldots \vee sc_{ki} \vee \ldots, \neg c \vee \neg sc_{k1} \vee \neg sc_{k2}, \ldots, \neg c \vee \neg sc_{ki} \vee \neg sc_{kj}, \ldots \}$

其中，$1i,1j \leq |SC_1|$且 $1i \neq 1j$，...，$ki,kj \leq |SC_k|$且 $ki \neq kj$。根据逻辑推导的基本性质[36]可知，描述部件装配关系的逻辑子句集合实质上就是描述其必选子部件的子句集 MCSL 和各个可替换子部件集的子句集 SCSL 的并集，即

$$ACL(c) = MCSL(c) \cup SCSL_1(c) \cup SCSL_2(c) \cup \ldots \cup SCSL_k(c)$$

定义 **3.8** （产品结构描述）对一个诊断问题 P (PSD, COMP, E^+, E^-)而言，其产品结构描述 PSD 就是描述各个部件间装配关系的逻辑语句的集合，即

$$PSD = \cup_{c \in COMP} ACL(c)$$

例 3.2 以图 3-3 中的产品结构关系为例，其逻辑描述如下：

$COMP = \{A, B, C, D, E, F, G, H\}$

$PSD = \{A \rightarrow (B \wedge \neg C) \vee (\neg B \wedge C), B \rightarrow (D \wedge E),$

$C \rightarrow (F \wedge \neg G \wedge \neg H) \vee (\neg F \wedge G \wedge \neg H) \vee (\neg F \wedge \neg G \wedge H \}$

化为标准子句形式后，

$PSD = \{\neg A \vee B \vee C, \neg A \vee \neg B \vee \neg C, \neg B \vee D, \neg B \vee E,$

$\neg C \vee F \vee G \vee H, \neg C \vee \neg F \vee \neg G, \neg C \vee \neg F \vee \neg H, \neg C \vee \neg G \vee \neg H\}$

其中，

$ACL(A) = \{\neg A \vee B \vee C, \neg A \vee \neg B \vee \neg C\}$

$ACL(B) = \{\neg B \vee D, \neg B \vee E \}$

$ACL(C) = \{\neg C \vee F \vee G \vee H, \neg C \vee \neg F \vee \neg G, \neg C \vee \neg F \vee \neg H, \neg C \vee \neg G \vee \neg H\}$

3.2.3 产品配置的逻辑描述

产品配置指的是为制造某产品（或半成品）所选择的零部件的生产组合。显然，这种生产组合方案也可以用逻辑形式所描述。

从生产的角度来说，产品配置是由上到下的装配路径选择关系即是从决定需要生产的父部件开始到其零部件组合方式的选择，所以产品结构的逻辑描述也是基于从父部件存在关系到子部件存在关系的推导，而非由下至上地从零部件组合推导其可以生产的装配产品。因此，产品配置的问题就是给定一个产品，然后决定生产该产品的零部件组合。产品配置的验证问题本质上是判断该产品与这些零部件之间的存在关系是否与产品结构的逻辑描述发生矛盾，如果该产品配置是合理的，则不存在矛盾，反之，必然存在矛盾。

定义 **3.9** （产品配置方案）对于一个产品结构（PSD, COMP）而言，它的产品配置方案是一个二元组（p , CONF），其中 p∈COMP 是生产的目标产品或部件，CONF⊂COMP 是生产 p 的零部件组合。

此外，对给定的产品结构而言，一个产品配置方案不仅仅说明了配置方案中构成产品的零部件的存在关系，还说明了其余不参与构成产品的零部件的不存在关系，这就有了以下定义。

定义 **3.10** （终端部件集合）对部件 c∈COMP，若 c 在产品结构 PSD 中不存在任何子部件，则称 c 是产品结构 PSD 的终端部件，终端部件对应于其产品结构树中的叶子节点，用 LEAF(PSD)表示产品结构 PSD 的终端部件集合。

定义 **3.11** （产品配置方案描述）对于一个产品结构（PSD, COMP），它的一个产品配置方案 cfg（p , CONF）可用以下逻辑语句描述：

$$p \rightarrow (c_1 \wedge c_2 \wedge \ldots \wedge c_i \wedge \ldots) \wedge (\neg e_1 \wedge \neg e_2 \wedge \ldots \wedge \neg e_i \wedge \ldots)$$

其中，$c_i \in CONF$，$e_i \in LEAF(PSD)$且 e_i 本身及 e_i 在产品结构 PSD 中的祖先部件不在 CONF 中出现。若用 PCL(cfg)表示产品配置描述的逻辑子句集，则有

$$PCL(cfg) = \{\neg p \vee c_1, \ldots, \neg p \vee c_i, \ldots, \neg p \vee \neg e_1, \ldots, \neg p \vee \neg e_i, \ldots\}$$

定义 **3.12** (产品配置的一致性) 如果一个产品配置方案 cfg（p , CONF）的逻辑描述与产品结构（PSD, COMP）的逻辑描述在产品 p 存在为真的情况下不存在逻辑矛盾，则称该产品配置与产品结构是一致的。

由定义 3.8 和定义 3.12 可知，产品结构描述和产品配置描述表示的都是从父

部件到子部件存在关系的逻辑推导。如果两者是不一致的话，则从生产的产品部件出发，经过各自装配关系推导得出的相应子部件的存在关系必然是矛盾的。总而言之，如果产品配置符合产品结构关系，则它们的逻辑描述必然是一致的，否则，它们的逻辑描述就会存在逻辑矛盾。为了具体说明这种关系，下面以一个例子来加以阐述。

例 3.3 假设产品结构关系如图 3-4 所示。

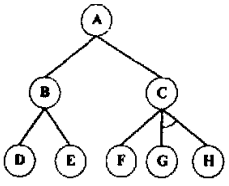

图 3-4 产品配置例

由定义 3.8，可知其产品结构的逻辑描述为

PSD = {¬A∨B，¬A∨C，

¬B∨D，¬B∨E，

¬C∨F，¬C∨G∨H，¬C∨¬G∨¬H }

①正确的产品配置方案Ⅰ：A=B+F+G 即 cfg(A,{B,F,G})

由定义 3.11, PCL(cfg)={¬A∨ B, ¬A∨F, ¬A∨G, ¬A∨¬H}

由归结原理知， PSD ∪PCL(cfg)∪{A}不存在逻辑矛盾，故一致。

②正确的产品配置方案Ⅱ：B=D+E 即 cfg(B,{D,E})

PCL(cfg)={¬B∨ D, ¬B∨E, ¬B∨¬F, ¬B∨¬G, ¬B∨¬H}

则 PSD ∪PCL(cfg)∪{B}不存在逻辑矛盾，故一致。

③错误的产品配置方案Ⅰ：A=D+C （缺少构成 B 的必选件 E）

PCL(cfg)={¬A∨ D, ¬A∨C, ¬A∨¬E}

归结后，PSD ∪PCL(cfg)∪{A}存在逻辑矛盾，故是不一致的。

④错误的产品配置方案Ⅱ：A=F+G （缺少构成 A 的必选件 B）

PCL(cfg)={¬A∨ F, ¬A∨G, ¬A∨¬D, ¬A∨¬E, ¬A∨¬H}

归结后，PSD ∪PCL(cfg)∪{A}存在逻辑矛盾，故是不一致的。

⑤错误的产品配置方案Ⅲ：C=F+G+H （替换件 G，H 同时存在）

PCL(cfg)={¬C∨ F, ¬C∨G, ¬C∨H, ¬C∨¬D, ¬C∨¬E}

归结后，PSD ∪PCL(cfg)∪{C}存在逻辑矛盾，故是不一致的。

⑤错误的产品配置方案Ⅳ：A=D+E+F （同时缺少 G 和 H）

PCL(cfg)={¬A∨ D, ¬A∨E, ¬A∨F, ¬A∨¬G, ¬A∨¬H}

归结后，PSD ∪PCL(cfg)∪{A}存在逻辑矛盾，故是不一致的。

3.2.4 诊断模型的逻辑描述

对一个 BOM 诊断问题 P（PSD, COMP, E^+, E^-）而言，如果其产品结构 PSD 与期望的产品配置样例 $E^+ \cup \neg E^-$不一致，就说明 BOM 所描述的产品结构与期望的形式存在差异，而诊断就是要找出引起这些差异的可能原因。

定义 3.13 （正例的逻辑描述）对于诊断问题 P（PSD, COMP, E^+, E^-），一个正产品配置样例 $e^+ \in E^+$是一个产品配置方案，是产品结构中允许出现的产品零部件组合。就诊断问题而言，e^+的逻辑描述形式与定义 3.11 相同：

$$p \rightarrow (c_1 \wedge c_2 \wedge \ldots \wedge c_i \wedge \ldots) \wedge (\neg e_1 \wedge \neg e_2 \wedge \ldots \wedge \neg e_i \wedge \ldots)$$

其中，$c_i \in CONF$，$e_i \in LEAF(PSD)$且 e_i 及 e_i 在 PSD 中的祖先部件不在 CONF 中出现，LEAF(PSD)是产品结构 PSD 的终端部件集合。

用 CCL(e^+)表示对应的逻辑子句集，则

$$CCL(e^+) = \{\neg p \vee c_1, \ldots, \neg p \vee c_i, \ldots, \neg p \vee \neg e_1, \ldots, \neg p \vee \neg e_i, \ldots\}$$

正产品配置样例的现实意义是需产品结构确保与之一致的产品配置方案。其逻辑实质是，从产品 p 的存在，可以在给定的产品结构中推导出相关零部件的装配关系，而这种装配关系应该包含配置样例中描述的零部件组合。

定义 3.14 （反例的逻辑描述）对于诊断问题 P（PSD, COMP, E^+, E^-），一个反产品配置样例 $e^- \in E^-$也是一个产品配置方案。就诊断问题而言，一般采用其否定描述，这里用$\neg e^-$表示，其否定描述的逻辑形式为：

$$p \rightarrow \neg ((c_1 \wedge c_2 \wedge \ldots \wedge c_i \wedge \ldots) \wedge (\neg e_1 \wedge \neg e_2 \wedge \ldots \wedge \neg e_i \wedge \ldots))$$

其中，$c_i \in CONF$，$e_i \in LEAF(PSD)$且 e_i 及 e_i 在 PSD 中的祖先部件不在 CONF 中出现，LEAF(PSD)是产品结构 PSD 的终端部件集合。

用 CCL($\neg e^-$)表示其对应的逻辑子句集，则

$$CCL(\neg e^-) = \{\neg p \vee \neg c_1 \vee \ldots \vee \neg c_i \vee \ldots \vee e_1 \ldots \vee e_i \vee \ldots\}$$

正产品配置样例具有现实意义是需要产品结构确保与之不一致的产品配置方案。其逻辑实质是，从产品 p 的存在，可以在给定的产品结构中推导出相关零部件的装配关系，而这种装配关系不应该包含配置样例中描述的零部件组合。

定义 3.15 （配置样例描述）为了统一叙述的方便，这里用 $E = E^+ \cup \neg E^-$表示所有的配置样例，其中$\neg E^- = \{ \neg e^- \mid e^- \in E^-\}$。这样，BOM 的诊断问题 P(PSD, COMP, E^+, E^-）也可同样定义为 P（PSD, COMP, E）。

至此，通过以上逻辑定义，可以引进基于模型诊断的技术（模型诊断的基本概念可参考附录 A）来对 BOM 的产品结构进行一致性的诊断。

定义 **3.16** （冲突集）对一个诊断问题 P（PSD, COMP, E），如果存在产品结构描述 PSD 的一个子集 CS⊆PSD，使得∃e∈E：CS∪e 是不一致的，则称 CS 是 P 的一个冲突集。由于 CS 是由 e 引起，也可以用 CS(e)来表示该集合。如果不存在 CS′⊂CS，使得 CS′也是 P 的一个冲突集，则称 CS 是 P 的一个极小冲突集。

冲突表明了实际 BOM 的产品结构中相关的若干部件装配关系不能同时正确，否则将与期望的产品配置样例不一致，这就是冲突的本质，在实际 BOM 系统中可能同时存在着多个冲突。

由于产品结构描述 PSD 是用逻辑子句集的形式描述的，若 PSD 与一个产品配置样例 e 冲突，可以采用删除无关子句法来求 e 引起的极小冲突集。具体做法是尝试逐个删除冲突集 CS 中的一个逻辑子句，如果冲突消失，表明该子句是与冲突是相关的，将其仍保留在冲突集中；如果仍然冲突，表明该子句与冲突是无关的，将其从冲突集中删除...如此继续，直至冲突集中的子句无法再被删除为止。

定义 **3.17**[17] 设 CH 是一个集合的集合，若对于集合 $H\subseteq\cup_{S\in CH}S$ 和 CH 中的任意一个集合 S′，都有 $H\cap S'\neq\phi$，则称 H 为 CH 的一个碰集。若 H 的任何一个子集都不是碰集，则 H 称为 CH 的极小碰集。

把碰集和冲突集两个概念结合起来，假设 C 为 PSD∪E 下所有冲突的集合，那么 C 的一个碰集 H 可理解为为解决这些冲突所作的假设，即若碰集中所有相关的部件装配关系都发生了错误，系统就不存在冲突了。实际上一个碰集就是一个诊断结果，Reiter 得出了所有冲突集的最小碰集就是一个诊断结果的结论[17]。

定理 **3.1** 对一个诊断问题 P（PSD, COMP, E）而言，DS⊆PSD 是诊断问题 P 的一个诊断当且仅当 DS 是 P 的冲突集集合的一个极小碰集。

Reiter 在文献[17]中证明了一个碰集就是一个候选诊断，根据 BOM 的诊断定义 3.3 及其冲突集的定义 3.16, 定理 3.1 显然也是成立的。因此，BOM 诊断的出发点就是从产品结构描述与期望的配置样例间的逻辑差异，找出冲突的装配关系集合，然后再从该集合中找出与所有冲突都相关的碰集 DS。

3.2.5 相关的进一步讨论

在模型诊断技术的应用中，比较重要的问题首先是建立实际系统的逻辑描述模型，然后就是在模型中构造实际与期望之间的冲突。在对 BOM 产品结构的描述中，本文的逻辑基础是从父部件的存在关系到相关子部件存在关系的推导。如果产品结构描述的推导关系与期望配置样例的推导关系不一致，这就构成了冲突，则相应的冲突集就是产品结构中与之产生矛盾的那一部分装配关系。

在建立逻辑描述的时候，我们也可以建立以子部件的存在作为条件，父部件的产生作为结果这样的推导方式。例如，图 2-1 结构的逻辑形式可为 $P_1 \wedge P_2 \rightarrow P$。但这样的话，产品配置问题就会类似于一个证明问题，即相关的零部件组合能否制造出给定的产品部件。由定理证明的方法，需将产品配置的逻辑描述取反后与产品结构的逻辑描述进行归结，如果产生矛盾（空子句）则表明该产品配置是正确的，否则，不会产生矛盾。然而，诊断问题的关键在于构造差异部分所造成的冲突，而证明问题是通过矛盾来证明结论，并不适用于求诊断冲突集。

3.3 BOM 的诊断算法

对 BOM（产品结构）的诊断过程可分为建立逻辑描述，计算冲突集合，产生诊断集合三个步骤。建立逻辑描述的主要任务是从 BOM 数据库中读出产品结构信息，然后把该产品结构信息以及产品配置样例转化成逻辑子句集的描述形式；计算冲突集合的主要任务是判断 BOM 的产品结构是否与各个产品配置样例相一致，若不一致，则找出产生这些冲突的部件装配关系；产生诊断集合的主要任务是计算所有冲突集合的最小碰集，并将其作为诊断结果输出。

3.3.1 建立逻辑子句集的算法

对一个产品结构而言，其逻辑描述就是由所有部件装配关系的逻辑子句集合组成的。定义 3.6 已经给出了描述父部件与必选子部件间装配关系的逻辑子句形式，定义 3.7 同样给出了描述父部件与可替换子部件间装配关系的逻辑子句形式。因此，根据定义 3.6 和定义 3.7，建立父部件与其所有子部件间装配关系的逻辑描述的过程可如算法 3.1。

算法 3.1 (建立部件装配关系的逻辑子句集)

输入：产品结构与或树 PTree，PTree 上的一个部件节点 c；

输出：描述部件 c 装配关系的逻辑子句集 ACL。

① 置 ACL={ϕ}；

② 找出 c 在 PTree 中所有的孩子部件，并置入集合 Child 中；

③ 如果 Child 为空，则结束，输出 ACL；否则，转④；

④ 从 Child 中取一个元素 d_0，如果 d_0 是 c 的必选子部件，则转⑤，如果 d_0 是 c 的可替换子部件，则转⑥；

⑤ 由定义 3.6,把逻辑子句$\neg c \vee d$ 加入 ACL,并从 Child 中删除 d_0,转③；

⑥ 找出 c 的所有可替换 d_0 的子部件 $d_1, d_2, \ldots, d_k$；

⑦ 由定义 3.7,把逻辑子句$\neg c \vee d_0 \vee d_1 \vee \ldots \vee d_k$ 加入 ACL，以及 $d_0, d_1, \ldots, d_k$ 的两两否定形式$\neg c \vee \neg d_i \vee \neg d_j (i \neq j)$也都加入 ACL；

⑧ 从 Child 中删除 $d_0, d_1, \ldots, d_k$,并转③；

将产品结构中各个部件装配关系的逻辑描述都分别求出，就可建立整个产品结构的逻辑描述，其过程如以下算法 3.2。

算法 3.2 (建立产品结构描述的逻辑子句集)

输入：产品结构与或树 PTree；

输出：描述该产品结构关系的逻辑子句集 PSD。

① 置 PSD={ϕ}；

② 把产品结构 PTree 上的所有部件置入集合 Comp 中；

③ 如果 Comp 为空，则结束，输出 PSD；否则，转④；

④ 从 Comp 中取一个元素 c，并按照算法 3.1 计算描述部件 c 装配关系的逻辑子句集合 ACL；

⑤ 把 ACL 中的所有逻辑子句加入 PSD,从 Comp 中删除 c，并转③；

对一个产品配置样例而言，其逻辑描述可如定义 3.13 和定义 3.14 给出的逻辑子句集形式，因此可有以下建立配置样例逻辑描述的算法 3.3。

算法 3.3 (建立配置样例描述的逻辑子句集)

输入：产品结构与或树 PTree，产品配置样例 e (p, Conf)；

输出：描述该产品配置样例的逻辑子句集 CCL。

① 置 CCL={ϕ}；

② 由定义 3.10，找出 PTree 中的所有终端（叶子）部件，置入集合 Leaf；

③ 把本身或其祖先部件在 CONF 中出现的终端部件从集合 Leaf 删除；

④ 如果 e 是正例，则转⑤，如果 e 是反例，则转⑥；

⑤ 由定义 3.13，把逻辑子句$\neg p \vee c_i$ ($c_i \in$ Conf)以及逻辑子句$\neg p \vee \neg e_i$ ($e_i \in$Leaf)加入 CCL，并转⑦；

⑥ 由定义 3.14，把逻辑子句$\neg p \vee \neg c_i$($c_i \in$Conf)及逻辑子句$\neg p \vee e_i$ ($e_i \in$Leaf)加入 CCL，并转⑦；

⑦ 结束，输出 CCL。

3.3.2 计算部件冲突集的算法

要计算由产品配置样例所引起的部件冲突集，首先就要判断该配置样例与产品结构的逻辑描述是否存在冲突。根据命题逻辑的归结原理（可参考附录 A），判断产品结构与产品配置样例是否存在逻辑冲突的方法可如算法 3.4。

算法 3.4 (判断配置样例与产品结构是否冲突)

输入：产品结构的逻辑描述 PSD，产品配置样例 e(p, Conf)的逻辑描述 CCL；

输出：PSD 与 CCL 是否产生冲突。

① 将子句集 PSD，CCL 以及表示产品 p 存在的子句{p}并入表 Clauses 中；

② 若有空子句 NIL 存在 Clauses 中，则结束，输出冲突；

③ 若 Clauses 存在可归结的子句对，则将其归结式并入 Clauses,并转②；若 Clauses 不再存在可归结的子句对，则转④；

④ 结束，输出不冲突；

若产品配置样例 e 与产品结构不一致，则可通过删除与冲突无关的部件装配关系来求得由该配置样例 e 引起的极小部件冲突集。根据这个思想，可有以下的算法 3. 5。

算法 3.5 (计算 BOM 诊断中的一个极小部件冲突集)

输入：产品结构的逻辑描述 PSD，产品配置样例 e(p, Conf)的逻辑描述 CCL；

输出：由 e 引起的一个极小部件冲突集 CS。

① 按照算法 3.4 判断 PSD 与 CCL 是否冲突，若没有，则结束，输出无冲突；

② 把在产品结构 PSD 中的所有部件置入集合 Comp，且置 CS={ϕ}；

③ 如果 Comp 为空，则转⑧

④ 取部件集 Comp 中的一个元素 c，并令 PSD′ = PSD；

⑤ 把 PSD′中描述部件 c 装配关系的逻辑子句 ACL(c)删除，然后按照算法 3.4 判断冲突是否仍然存在，若存在则令 PSD = PSD′，转⑦，否则转⑥；

⑥ 把部件 c 加入冲突集 CS 中；

⑦ 把部件 c 从 Comp 中删除，转③；

⑧ 结束，输出 CS。（在 CS 中部件的装配关系存在冲突）

3.3.3 计算诊断（碰集）的算法

正如 Reiter[17]指出，一个诊断就是冲突集集合的一个最小碰集。而从冲突集的集合求最小碰集的算法很多，从最早 Reiter 提出 HS 树[17]及其改进 HST-树[42]，到后来的递归计算[43]，BHS-树[44]，遗传算法[45]，布尔代数[46] 等方法都可以应用在 BOM 诊断求最小碰集。把一个部件对应于布尔代数中的一个文字，就可应用布尔代数法来求冲突部件集的最小碰集（诊断），其理论基础可参考文献[46]。

算法 3.6 (用布尔代数法[46]求冲突部件集集合的最小碰集)

输入：冲突部件集的集合 CSS = {CS_0, CS_1, CS_2, ...}；

输出：最小碰集集合 HSS。

① 如果CSS只由一个集合构成，即CSS= {{c_1,c_2,...,c_n}}，则算法终止，输出 HSS={{c_1},{c_2},...,{c_n}}，否则转②；

② 检查CSS中是否存在只有单个部件的集合，如有则转③，否则转⑥；

③ 设e是CSS中单部件集合中的那个部件元素，如果CSS中所有的部件集都含e，则算法终止，输出CS={{e}}，否则转④；

④ 把CSS分为含部件e的部件集集合CSS_1和不含部件e的部件集集合CSS_2两个部分，并递归调用本算法计算CSS_2的最小碰集HSS_2；

⑤ 把部件e作为元素插入HSS_2的各个部件集，算法终止，输出HSS= HSS_2；

⑥ 统计CSS中各部件元素出现的频度，设部件e在部件集中出现频度最高；

⑦ 把CSS分为含部件e的部件集集合CSS_1和不含部件e的部件集集合CSS_2两个部分，然后把部件e从CSS_1中删掉，并分别递归调用本算法计算CSS_1的最

小碰集HSS_1和CSS_2的最小碰集HSS_2。

⑧ 把部件e作为元素插入HSS_2的各个部件集；

⑨ 算法终止，输出$HSS= HSS_1 \cup HSS_2$。

用布尔代数法计算最小碰集有以下优点[47]：

(1)不需要建立树；(2)算法比较容易在计算机中实现；(3)具有很好的空间复杂度和时间复杂度。

3.3.4 BOM 诊断问题的求解算法

算法 3.7 (求解 BOM 的诊断问题的算法)

输入：BOM 的诊断问题 P（PSD, COMP, E^+, E^-）； 输出：诊断集 DSS。

① 从BOM表读入产品结构信息，由算法3.2建立其逻辑描述PSD；

② 由算法3.3为E^+和E^-中的各个产品配置样例建立逻辑描述；

③ 由算法3.5计算产品结构PSD与各个不一致样例间的极小部件冲突集；

④ 由算法3.6计算所有部件冲突集的最小碰集簇（诊断集）DSS；

⑤ 算法结束，输出所有可能的诊断方案ds∈DSS及其对应的装配关系。

3.4 BOM 的诊断系统

3.4.1 系统的基本结构

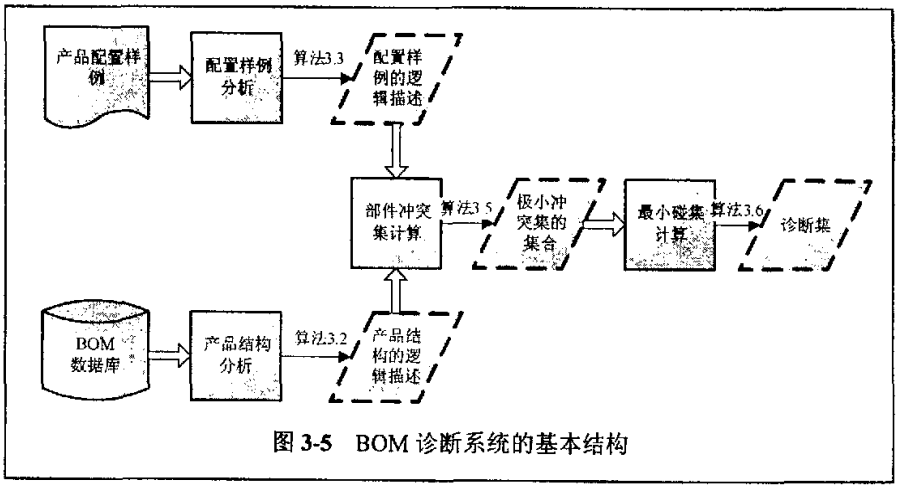

图 3-5 BOM 诊断系统的基本结构

3.4.2 测试结果及讨论

为了对BOM的诊断方法进行测试，本文实现了一个诊断系统原型。程序使用Visual C++.NET 2003开发，硬件环境是Intel Pentium III 800Mhz中央处理器，128M内存，40G硬盘的台式计算机，软件环境是Microsoft XP Professional SP2操作系统，Microsoft .NET Framework 1.1，数据库采用Access 2003。

测试一、诊断结果的正确性

为了测试诊断结果的正确性，这里采用若干产品 BOM 作为测试用例，并人为地植入一些部件装配关系的错误，并辅以相关的配置样例，然后测试诊断系统对这些植入错误的诊断正确率，以下表 4-1 是测试结果：

表 4-1　诊断的正确性测试

部件数	正例	反例	实际错误	诊断方案	命中	诊断错误总数	正确	诊断率
10	5	2	3	1	1	3	3	100%
31	7	3	5	3	1	7	5	100%
85	9	2	4	1	0	3	3	75%
160	28	25	12	1	0	6	6	50%

其中，“实际错误”是人为植入的错误总数，“诊断方案”为系统给出的诊断结果集（可能诊断）个数，“命中”指的是正确诊断出全部植入错误的方案数，诊断错误总数是指全部方案所诊断出的错误数总和。

由表 4-1 可知，随着产品结构中部件数的增多，诊断率有所下降，这是因为诊断结果与所输入的配置样例对错误的覆盖率密切相关。配制样例对错误的覆盖率越高，提供的辅助信息越多，诊断率就越高。在理论上，若输入的配置样例能把全部错误因素都覆盖在内，就可得到全部的诊断。但在实际中是难于保证这一点的，由于配置样例对错误的覆盖率会随着产品结构规模的增大而降低，所以提高诊断率的有效方法就是提供尽可能多的符合期望的产品配置样例。

另外一个测试是把植入的错误取消，检验系统对正确的产品结构会否出现误诊情况，结果如下：

表 4-2　对正确产品结构的诊断

部件数	正例数	反例数	实际错误	诊断错误	有否误诊
85	9	2	0	0	无
160	28	25	0	0	无

测试二、诊断算法的运行时间

表 4-3 测试算法的运行时间（单位：秒）

部件数	正例数	反例数	建立逻辑描述	计算冲突	计算诊断	总时间
10	5	2	0.67	0.07	0.01	0.75
31	7	3	0.91	0.19	0.02	1.12
85	9	2	1.72	0.34	0.03	2.09
160	28	25	0.72	0.48	0.03	1.23

从表 4-3 可以看出，诊断算法的总运行时间都在单位秒的数量级之内，具有不错的时间性能。其中，建立产品结构逻辑描述所耗费的时间最长，这是由于从 BOM 表读入产品结构信息时的数据库读写操作需占用较多的时间，而且系统在建立产品结构树时采用了递归的方式。因此，一般来说，产品结构关系越复杂，递归次数（与产品部件数不一定成比例）越多，耗费的时间也就越长。

在计算冲突集方面，因为测试时设置的产品结构错误随部件数增多而增多，故系统所产生的冲突集合也随之增多，耗费时间也随之增长。此外，系统在逻辑子句归结时采用了单元子句优先[48]的归结策略，时间性能获得了显著的改善。

在计算诊断集(最小碰集)时，系统采用的是时间性能很好的布尔代数法[46]，这里的测试结果与文献[16]中相关的测试结果在时间量级上也是一致的。

测试三、正例对诊断结果的影响

表 4-4 测试正例对诊断结果的影响

部件数	正例数	反例数	实际错误	诊断结果	正确诊断	多余诊断	诊断率
31	1	0	5	1	1	0	20%
31	3	0	5	4	3	1	60%
31	5	0	5	3	3	0	60%
31	7	0	5	6	4	2	80%

从表 4-4 结果可知，在产品结构不变的情况下，增加产品配置正例一方面可以提高错误的诊断率，另一方面在保持诊断率的前提下也可以减少多余的诊断数。从理论上分析，正例（即正确的产品配置样例）描述的是所期望的产品结构的特性，提供的所期望的产品结构的描述信息越详细，可供系统参考的逻辑推导依据就越多，系统的诊断率也就随之提高。由诊断结果来看，系统主要根据正例推导期望的产品结构特性，从而判断实际产品结构与之不一致的地方。

测试四、反例对诊断结果的影响

表 4-5 测试反例对诊断结果的影响

部件数	正例数	反例数	实际错误	诊断结果	正确诊断	多余诊断	诊断率
31	0	3	5	0	0	0	0%
31	0	9	5	1	1	0	20%
31	0	15	5	1	1	0	20%
31	0	20	5	2	2	0	40%

从表 4-5 可以看出，增加反例（即非法的产品配置样例）数量对诊断结果的影响不如正例明显。从理论上看，反例描述的是不期望出现的产品结构特性，显然，反例的外延范围要远比正例大。从测试结果来看，若给定的反例能给实际诊断提供帮助，需要符合两个条件：第一，该反例是与期望的产品结构不一致的；第二，该反例却又恰好符合实际中待诊断的产品结构。满足第一个条件的反例数目很多，但同时满足第二个条件的只是其中的一个少数部分。所以，反例对诊断结果只能起着辅助和补充的作用。

3.5 本章小结

如何把模型诊断技术结合到实际应用领域是基于模型的诊断技术研究的重要课题之一。本章通过建立 BOM（产品结构）的逻辑描述模型，并引入产品配置样例作为系统期望的结果，把模型诊断技术与 BOM（产品结构）的错误检查及诊断结合了起来，有效地解决了对产品结构语义错误的诊断问题，其主要工作可概括为以下几个方面：

(1) 给出了产品结构及产品配置的逻辑描述形式，结合模型诊断技术的特点，建立了一个 BOM 诊断问题的模型；

(2) 把模型诊断技术应用于 BOM 的错误诊断中，并根据产品结构的逻辑描述形式，给出了诊断问题求解及计算的相关算法；

(3) 实现了一个诊断系统原型，并对测试结果进行了定性分析和讨论。

需要特别指出的是，本章的诊断系统虽然是在第二章提出的与或树 BOM 结构基础上实现的，但就诊断方法而言，与实际的 BOM 结构形式关系不大。其他形式的 BOM 结构同样可以将其转换为对应的逻辑描述形式，实现对 BOM 的错误诊断。

第四章　总结

4.1 相关工作的比较

本文主要针对 BOM 的变型设计及数据维护问题，提出了一种基于与或树的柔性 BOM 结构和一种对产品结构错误的自动诊断方法。以下就本文所作的工作与相关领域的其他工作进行一些比较和讨论：

在 BOM 的产品结构定义方面，本文对传统的树形产品结构[1]进行了扩展，通过增加“与树”和“或树”语义，提出了产品结构与或树的形式化定义。与原来纯粹的树形产品结构相比，与或树形式的产品结构在与其保持结构兼容的基础上，还具有了更多的柔性特征，可以显著地改善在 BOM 变型设计中的数据冗余问题。在进行产品结构的动态配置时，与 Wolfram[49]提出的规则驱动和 Alexander[50]提出的定制参数等将变型特性从产品数据中分离的方法不同，产品结构与或树的实质是将变型规则蕴含在产品结构数据中。与规则驱动的配置方式相比，该方法虽然不允许用户自定义和扩展规则，但在产品结构设计时不需要建立额外的规则库和输入相关的规则集，可以更方便地实现对产品结构的管理。与输入配置参数的方法相比，该方法在产品配置和 BOM 数据传递时并不需要额外的配置参数信息，可以保持产品数据表达的一致性。

在 BOM 数据的正确性维护方面，目前的大部分工作[14, 52]都主要针对 BOM 数据的一致性、完备性等语法相关的错误检查，在产品结构语义相关的错误检测乃至诊断方面所作的工作不多。文献[14]采用逻辑谓词的形式描述了产品结构中部件关系、部件约束以及部件配置等基本概念，在 BOM 一致性维护方面做了一些研究和探讨，但同样未就其语义错误提出一个有效的诊断方法。在模型诊断技术的应用中，大部分的工作还是集中在硬件故障[17, 25]和程序错误[27, 28]诊断方面，在相关 BOM 领域的应用不多。文献[55]研究了知识库的一致性诊断问题，利用正例和反例来诊断知识库中的规则冲突。但由于其诊断是基于知识规则集合的，若具体到实际领域，在如何定义规则来描述领域知识方面还需要与具体应用结合。而本文在这方面的主要贡献就在于建立了一个 BOM 诊断模型，把基于模型诊断技术应用到产品结构的语义错误诊断方面，并且实现了一个 BOM 诊断系统的原型。

4.2 本文工作的总结

BOM 是企业生产管理系统集成的关键，是产品结构和产品配置管理的核心，是企业不同部门、不同流程间传递的基本数据。本文主要在 BOM 结构的柔性定义（第二章）和 BOM 错误的自动诊断（第三章）两个方面作了相关研究和阐述。

针对BOM的变形设计问题，本文扩展了传统产品结构树的语义，提出了一种基于产品结构与或树的柔性BOM结构，该产品结构不仅可以满足产品动态定制的变型要求，还可以实现面向任务的产品配置，其主要工作有：

(1) 提出了基于与或树的产品结构的形式化描述；

(2) 给出了相关的BOM设计方法和实现；

(3) 在上述基础上实现了一个面向任务的产品配置算法。

针对BOM的正确性维护问题，本文将基于模型的诊断技术应用于BOM的语义错误诊断上面，建立了相关的BOM诊断模型，其主要工作有：

(1) 给出了相关产品结构和产品配置的逻辑描述形式；

(2) 把基于模型的诊断方法与 BOM 的错误诊断相结合，给出了一个 BOM 语义错误的诊断方法及其相关算法；

(3) 实现了一个诊断系统原型，并对测试结果进行了相关分析和讨论。

当然，这里提出的诊断模型在理论上并不是完备的(理论上难以证明其完备性)，只是对实际系统的模拟抽象和近似，这是本文的不足之处。

4.3 下一步工作及展望

正如测试结果所反映的那样，在BOM的错误诊断中，产品配置样例对诊断结果具有直接的影响，不同的配置样例组合往往具有不同的诊断效果。因此，如何选择合适的产品配置样例，如何完善BOM诊断模型，进一步提高诊断率将是本文将来需要研究和完善的主要内容。另外，如何把诊断系统整合在实际ERP/PDM系统中，将其应用到实际的生产管理中去，也是下一步工作的重点。

与此同时，由于产品结构与产品配置间具有紧密的逻辑联系，如何进一步扩展诊断技术的外延，通过配置样例集合及诊断结果集推导出正确的产品结构，从而实现自动纠错，也将是一个十分有意义且具有广阔应用前景的研究方向。

参考文献

[1] 温咏棠. MRP-II 制造资源计划系统. 北京: 机械工业出版社, 1994.

[2] 胡敏. 企业集成环境下的 BOM 研究. 计算机工程, 2001, 27(6): P22-24.

[3] 刘艳凯, 于明, 张斌等. ERP 系统中 BOM 构造方法研究. 计算机集成制造系统-CIMS. 2003, 9(4): P309-313

[4] 程存有, 叶晓俊. BOM 的建立及在 PDM 与 ERP 集成系统中的应用. 计算机工程. 2003, 29(4): P143-145.

[5] 曾富东, 曾富鸿. PDM 中 BOM 实现技术研究. 机械设计与制造. 2001, 8(4): P94-95.

[6] Guo-li J, Da-xin G, Freddie T. A Tree Structure Storage of BOM Modal. Journal of. Systems Science and Systems Engineering. 2002, 11(1): P55-60.

[7] 梁书锋, 郭顺生, 熊志勇. 使用递归算法的 N 级结构的 BOM 的设计. 计算机应用. 2003, 23(12): P274-274.

[8] Ismail, N, Tai, S.S, Leman, Z. Improving productivity and efficiency of a vehicle seat assembly line in a manufacturing company. Student conference on Research and Development, SCOReD 2002.

[9] Cao, H, Xi.H, Smith, S.F. A reinforcement learning approach to production planning in the fabrication/fulfillment manufacturing process. Proceedings of the 2003 Winter Simulation Conference, 2003, Volume2: P1417 – 1423.

[10] van der Aalst W.M.P, Reijers H.A, Liman S. Product-driven workflow design. The Sixth International Conference on Computer Supported Cooperative Work in Design, 2001.

[11] Jayant K, Moninder S, et.al. Industry/government track posters: A system for automated mapping of bill-of-materials part numbers. The ACM international conference on Knowledge discovery and data mining, SIGKDD 2004.

[12] Sulaiman, A. Using Datalog data model and data mining to solve bill of materials personalization problems. Conference on Advance Issues of E-Commerce and

Web-Based Information Systems, WECWIS, 1999.

[13] 袁品鹏, 陈刚, 董金祥. BOM 一致性维护. 计算机辅助设计与图形学学报. 2002, 14(1): P83-86.

[14] 张玲, 潘晓弘, 董天阳等. MRP 运算的数据完备性保证策略. 计算机应用研究. 2004, 2: P23-25.

[15] 欧阳丹彤, 欧阳继红, 刘大有. 基于模型诊断的研究与新发展. 吉林大学自然科学学报. 2001,4: P38-45.

[16] 林笠. 基于模型诊断算法及应用. 中山大学博士学位学术论文, 2002 年 6 月.

[17] Reiter R. A Theory of diagnosis from first principles. Artificial Intelligence, 1987, 32(1): P57-96.

[18] Console L, DupreD T, Torasso P. A theory of diagnosis for incomplete casual models. Proceeding of 11th IJCAI, Detroit, Morgan Kaufmann Publishers, 1989.

[19] Console L, Torasso P. A Spectrum of logical definitions of Model-based diagnosis. Computational Intelligence, 1991, 7(3): P133-141.

[20] 欧阳丹彤, 姜云飞. 广义因果理论的基于模型的诊断. 计算机软件与发展. 1999, 36(1): P31-35.

[21] 欧阳丹彤, 姜云飞. 扩展的因果理论的鉴别诊断. 软件学报. 1999, 10(10). P719-723.

[22] Geffner H, Pearl J. Distributed Diagnosis of Systems with Multiple Faults. Proceeding of 3th IEEE Conference on AI application, Orlando, 1987, P156-162.

[23] De Kleer J, Williams B.C. Diagnosing Multiple Faults. Artificial Intelligence, 1987, 32(1): P97-130.

[24] Genesereth M.R. The Use of Design Descriptions in Automated Diagnosis. Artificial Intelligence. 1984, 24(1-3): P411-436.

[25] Friedrich G.E, Stumptner M, Wotawa F. Model-based diagnosis of hardware designs. Artificial Intelligence. 1999, 111(2): P3-39.

[26] Console L, Friedrich G.E, Dupre D.T. Model-based diagnosis méets error diagnosis in logic programs. Proceeding of IJCAI-93. San Mateo, CA, 1993: P1494-1499.

[27] Stumptner M, Wotowa F. Debugging functional programs. Proceeding of

IJCAI-99. San Mateo, CA, 1999: P1074-1079.

[28] Mateis C, Stumptner M, Wotowa F. Modeling Java Programs for diagnosis. Proceeding of 14th European Conference on Artificial Intelligence, Berlin, Germany, IOS Press, 2000: P171-175.

[29] Mosterman P.J, Biswas G. Monitoring, prediction, and fault isolation in dynamic physical system. Proceeding of AAAI-97. Rhode Island, 1997: P100-105.

[30] Rymon R. Goal-directed diagnosis --- a diagnostic reasoning framework for exploratory corrective domain. Artificial Intelligence, 1996, 84: P257-297.

[31] Hamscher W. CROSBY: financial data interpretation as model-based diagnosis. Annals of Mathematics and Artificial Intelligence, 1994, 11: P511-P524.

[32] Chittaro L, Guida G, Tasso C, et.al. Functional and Teleological. Knowledge in the Multimodeling Approach for Reasoning about Physical Systems. IEEE Transactions on Systems,Man,and Cybernetics, 1993,23(6).

[33] De Kleer J. Focusing on probable diagnosis. Proceeding of AAAI, Anaheim, CA, 1991: P842-848.

[34] Olsen K.A, Aetre P, Thorstenson A. A Procedure-oriented Generic Bill of Materials. Computers in Industry Engineering. 1997. 32(1): P29-45.

[35] 罗鸿, 王忠民. ERP 原理·设计·实施. 北京: 电子工业出版社, 2003.

[36] Nilsson N.J. Principles of Artificial Intelligence. North Holland: Tioga Publishing, 1980.

[37] 王永庆. 人工智能原理和方法. 西安: 西安交通大学出版社, 2001.

[38] Dechter R. AND/OR Search Spaces for Graphical Network. Technical Report. University of California, 2004.

[39] Nandakumar G. The Design of a Bill of Materials Processor Using a Relational Data Base. Computers in Industry, 1985(6): P15-21.

[40] Genesereth M.R. The use of design descriptions in automated diagnosis. Artificial Intelligence, 1984(24): P411-436.

[41] Haenni R. Generating Diagnoses from Conflict Sets. Proceedings of the FLAIRS-98 Conference, AAAI Press, 1998: P120-124.

[42] Franz Wotawa. A Variant of Reiter's Hitting2set Algorithm. Information

Processing Letters, 2001, (79) :P45-51.

[43] 林笠. 递归建立HS2树计算最小碰集. 微电子学与计算机, 2002 ,19(2): P7-10.

[44] 姜云飞,林笠.用对分 HS-树计算最小碰集. 软件学报. 2002,13(12):P2268-2274.

[45] Lin Li , Jiang Yunfei. Computing Minimal Hitting Sets with Genetic Algorithm. Austria. Proceeding of 13th International Workshop on Principles of Diagnosis. 2002: P77-80.

[46] 姜云飞,林笠. 用布尔代数方法计算最小碰集. 计算机学报. 2003, 26(8): P919-924.

[47] 林笠. 在基于模型诊断中计算最小碰集算法. 计算机应用研究. 2002, P36-39.

[48] 廉师友. 人工智能技术导论. 西安: 西安电子科技大学出版社, 2000.

[49] Wolfram W. A Rule-driven Generator for Variant Parts and Variant Bills of Material. Proceedings of Eighth International Workshop on Database and Expert Systems Applications, 1997.

[50] Alexander R. Fulfilling Customer's Needs by The Use of a Variant Configurator for Dynamic Product Definition. Proceedings of 7th IEEE International Conference on Emerging Technologies and Factory Automation, Volume, 1999.

[51] 王庆国, 蔡淑琴, 陈宏峰等. 面向客户定制的动态 BOM 模型及算法. 计算机工程与应用. 2002, 24: P94-95.

[52] 仲智刚, 潘晓鸿, 程耀东等. 物料清单正确性检查方法研究. 机械科学与技术. 2001, 20(1): P136-137.

[53] Blaha, M.R, Premerlani W.J, Bender, et.al. Bill-of-material configuration generation. Proceedings of Sixth International Conference on Data Engineering, 1990: P237 – 244.

[54] 李占山, 姜云飞, 王涛. 基于模型的诊断问题分解及其算法. 计算机学报. 2003, 26(9): P1171-1175.

[55] Felfernig F, Friedrich G, Jannach D, et.al. Consistency-based diagnosis of configuration knowledge bases. Artificial Intelligence. 2004, 152: P213-234.

附录 A 预备知识

一、模型诊断的基本概念

基于模型的故障诊断是重要的故障诊断方法之一[17]。该方法首先对一个系统进行描述，再根据不同的输入，观测输出结果；如果观察结果和系统应有的结果冲突，则需要确定出可能出故障的器件。

对于基于模型的诊断方法，Reiter[17]用如下定义描述数字系统：

定义 1.1[17] 一个系统是一个序对<SD，COMPONENTS>，这里 SD 是系统的描述，也就是一个一阶语句的集合，COMPONENTS 是系统中元件的集合，它是一个有限的常量集。下面用文献[40]中的全加器例子做说明。

例子 **1.1**[40] 图 1.1 是一个全加器，A_1 和 A_2 是与门，X_1 和 X_2 是与或门，O_1 是或门。

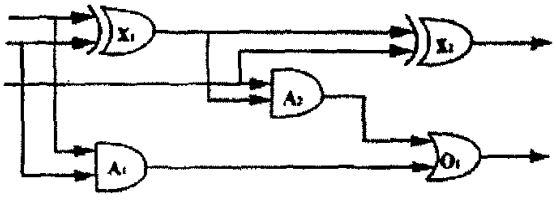

图 1.1 全加器例

该加法器可以由如下的系统来描述,元器件的集合 COMPONENTS 为 $\{A_1,A_2,X_1,X_2,O_1\}$，SD 由下列规则组成：

(1) ANDG(x)∧¬ AB(x)⊃out(x)=and(in1(x),in2(x));

(2) XORG(x)∧¬ AB(x)⊃out(x)=xor(in1(x),in2(x));

(3) ORG(x)∧¬ AB(x)⊃out(x)=or(in1(x),in2(x));

(4) ANDG(A_1); (5) ANDG(A_2); (6) XORG(X_1);

(7) XORG(X_2); (8) ORG(O_1); (9) out(X_1)=in2(A_2);

(10)out(X_1)=in1(X_2); (11)out(A_2)=in1(O_1); (12)in1(A_2)=in2(X_2);

(13)in1(X_1)=in1(A_1); (14)in2(X_1)=in2(A_1); (15)out(A_1)=in2(O_1).

除此之外,还包含如下说明输入是二进制数值的公理：

in1(X_1)=0∨in1(X_1)=1, in2(X_1)=0∨in2(X_1)=1, in1(A_1)=0∨in1(A_1)=1.

定义 1.2[17] 系统的一个观测(Observation)是一个有限的一阶语句集。对于系统<SD, COMPONENTS>，如果得到的观测为 OBS，则记为<SD, COMPONENTS, OBS>。

例子 1.2[17] 假设上面加法器的输入为 1、0、1，输出为 1 和 0，则该观测可以用如下的语句描述：

(16)in1(X_1)=1, (17)in2(X_1)=0, (18)in1(A_2)=1, (19)out(X_2)=1, (20)ou(O_1)=0.

定义 1.3[17] CC 称为<SD，COMPONENTS，OBS>的一个冲突集(Conflict Set)，如果 CC 是 COMPONENTS 的一个子集，并且 SD∪OBS∪ {¬ AB(c) | c∈CC} 是不一致的。如果不存在<SD, COMPONENTS, OBS>的任何一个冲突集 Δ′，使得 CC′ ⊂CC 成立，则称 CC 为极小冲突集。

例子 1.3 对于上面的加法器和例子 1.2 中的观察，其最小冲突集为 {X_1, X_2} 和 {X_1, A_2, O_1}。

定义 1.4[17] 设 CH 是一个集合的集合，若对于集合 H⊆∪$_{S\in CH}$S 和 CH 中的任意一个集合 S′，都有 H∩S′ ≠ ϕ，则称 H 为 CH 的一个碰集。若 H 的任何一个子集都不是碰集，则 H 称为 CH 的极小碰集。

定义 1.5[17] <SD，COMPONENTS，OBS>的一个诊断是 COMPONENTS 的一个极小子集 CC，CC 满足如下条件：SD∪OBS∪ {AB(c) | c∈CC} ∪ {¬ AB(c) | c∈COMPONENTS - CC}。对于前面的例子和观测结果，得到其诊断为：{X_1}、{X_2, A_2} 和 {X_2, O_1}。

事实上，一个极小击中集就是一个诊断，文献[17]证明了如下的定理 1.1。

定理 1.1[17] CC⊆COMPONENTS 是<SD，COMPONENTS，OBS>的一个诊断当且仅当 CC 是<SD，COMPONENTS，OBS>的冲突集的集合的一个极小碰集。

进一步，定理 1.1 中的冲突集的集合可以是极小冲突集的集合，也就是有如下的定理 1.2。

定理 1.2[17] CC⊆COMPONETNS 是<SD，COMPONENTS，OBS>的一个诊断当且仅当 CC 是<SD，COMPONENTS，OBS>的极小冲突集的集合的一个极小碰集。

通过上面的介绍可以看到，只要得到所有极小冲突集，就可以非常容易的得到所有诊断，Reiter[17] 也指出了这一点。但他在文献[17]中没有给出一种求所有极小冲突集的方法，只给出了一种间接的求得所有诊断的方法。该方法采用一般的定理证明器来得到冲突集，再根据冲突集求诊断。正如 Reiter 本人指出，这种方法存在很多不确定性：定理证明器每次求得的结果不一定是理想的结果，这样

使得求诊断更加复杂化。文献[17]中给出的例子也都是假设定理证明器返回的结果是某个冲突集，事实上这一次定理证明器给出的并不一定是该冲突集，而是别的冲突集。后来 Haenni [41] 在假设已经得到所有极小冲突集的情况下，给出了求所有诊断的算法。所以求所有极小冲突集成为了一个主要问题。

二、计算最小碰集的算法

一般来说，从极小冲突集计算最小碰集（诊断集）的方法与具体的系统模型及应用领域关系不大，所以一般具有较广的通用性。在基于模型的诊断技术的发展过程中，也出现了许多计算最小碰集的算法，如从最早 Reiter 提出 HS 树法[17]及后来改进的 HST-树[42]，到递归计算[43]，BHS-树[44]，遗传算法[45]，布尔代数[46]等等算法。

尽管从极小冲突集求最小碰集并不是本文研究的主要内容，但其仍然是模型诊断过程中一个必不可少的关键步骤。故下面简单介绍在上面所列举的方法中，时间和空间复杂度最好的——用布尔代数方法计算最小碰集的算法[47]。

设CS={C_1 ,C_2 , … ,C_m}是一个集合簇，其中的集合Ci = {e_{i1} ,e_{i2} , … ,e_{ini}} (i = 1 ,2 , …, m)，把负布尔表达式($\hat{e}_{11}\hat{e}_{12}...\hat{e}_{1n1} + \hat{e}_{21}\hat{e}_{22}...\hat{e}_{2n2} + ...+ \hat{e}_{m1}\hat{e}_{m2}...e_{mn}$)称为集合簇CS的布尔形式，记为CSF。其中$\hat{e}_{ij}$是与$e_{ij}$互补的文字。

设H= {h_1 ,h_2 , …, h_n}是一个集合，则我们把正布尔单项式$h_1h_2...h_n$称作该集合的布尔形式,记为HF。

对于一个集合簇和一个集合的布尔形式,我们有如下重要结论：

设集合簇HS是集合簇CS的所有最小碰集的集合簇，HS = {H_1 , H_2 , …, H_k}其中H_i = {h_{i1} , h_{i2} , ..., h_{ini}}，则($h_{11}h_{12}...h_{1n1} + h_{21}h_{22}...h_{2n2}+ ... + h_{k1}h_{k2}...h_{knk}$)称为碰集簇的布尔形式。显然，如果已经求出了碰集簇的布尔形式，就容易得到该碰集簇。

例2.1[17] 设CS={{2,4,5}，{1,2,3}, {1,3,5}, {2 ,4 ,6}, {2 ,4}, {2 ,3 ,5}, {1,6}}，则：

CSF = (245 + 123 + 135 + 246 + 24 + 235 + 16)，

HF = (12)。

首先,我们引进一个新的函数Γ(Π) 。

定义2.1[46] (Γ(Π) 函数) 该函数的定义域是负布尔表达式,值域是正布尔表达

式, ê与e是互补的文字。

Γ(Π)函数递归定义如下：

(1) Γ(0) = 1， Γ(1) = 0("0", "1"是布尔常量);

(2) Γ(ê) = e ；

(3) Γ(ê · Π) = e +Γ(Π) ；

(4) Γ(ê+Π) = e · Γ(Π) ；

若Π不满足以上(1) 、(2) 、(3) 、(4) 条，则用以下规则(5)计算：

(5) Γ(Π) = e · Γ(Π_1) +Γ(Π_2)。ê是Π中的任意一个文字,Π_1 是Π中不含ê的全部单项式,Π_2 是Π中所有项(包括含ê及不含ê的项) 删除ê后剩余的布尔形式。为叙述方便起见，规则(5)也称作"按ê分解"规则。

根据定义2.1，我们有如下结论：

定理2.1[46] 设CSF =Π是最小集合簇CS的布尔形式，则Γ(Π) 就是CS的最小碰集簇的布尔形式。

证明：见文献[46]。

算法2.1[46] (用布尔代数方法递归计算碰集簇算法BHS(Π)) 设CS表示冲突集合簇,Π是CS的布尔形式,算法BHS的输入为Π,输出为最小碰集的布尔形式Γ(Π)。

Step 1 如果Π由一个单项构成，即Π= ($\hat{e}_1\hat{e}_2...\hat{e}_n$)，则由定义2.1，输出结果为: $\hat{e}_1+\hat{e}_2+...+\hat{e}_n$，BHS 算法终止；否则执行Step 2。

Step 2 检查Π中是否有单文字单项式，如有，执行Step 3 ,否则,执行Step 4。

Step 3 设e是Π中的单文字单项式，根据定义2.1提取公因子e。如果Π中所有的单项式都含e，则输出BHS(Π) = e，算法终止；否则，BHS(Π) = e · BHS(Π1) ，并递归计算BHS(Π_1)。

Step 4 计算Π中各文字出现的频度，设e是各文字中出现频度最高的。

Step 5 根据定义2.1的规则(5)，执行按e分解，BHS(Π) = e · BHS(Π_1) + BHS(Π_2) ,并分别递归计算BHS(Π_1)和BHS(Π_2)。其中，Π_1是Π中不含e的全部单项式,Π_2是在Π的全部表达式中删除e之后剩余的布尔表达式。

用算法2.1计算碰集有以下优点: ①不需要建立树；②只要一次递归计算字符串即可以得到碰集；③空间复杂度和时间复杂度均是目前所有算法中最好的。

三、命题逻辑的归结原理

归结原理是由Robinson于1965年首先提出的，是谓词逻辑中一个相当有效的机械化推理方法。归结原理的出现，被认为是自动推理，特别是定理机器证明领域的重大突破。

定义3.1[48] 设L为一个文字，则称L与¬L为互补文字。

定义3.2[48] 设C_1，C_2是命题逻辑中的两个子句，C_1中有文字L_1，C_2中有文字L_2，从C_1，C_2中分别删除L_1，L_2，再将剩余部分析取起来，设构成的新子句为C_{12}，则称C_{12}为C_1和C_2的归结式(或消解式)，C_1和C_2称其为归结式的亲本子句，L_1和L_2称为消解基。

例3.1 设$C_1=\neg P\vee Q\vee\neg R$，$C_2=P\vee S$，则$C_1$和$C_2$的归结式为

$$C_{12}=Q\vee\neg R\vee S$$

定理3.1[48] 归结式是其亲本子句的逻辑结果。(证明略，可参考文献[48])

由归结原理可知，如果两个互否的单元子句进行归结，则归结式为空子句即恒假子句，这就提供了定理间接证明的方法。事实上，任何一个一阶命题逻辑都可转化成一种称为子句集的标准形式，具体步骤可参考文献[36]，对子句集使用归结方法就可以进行命题的反演推理。

推论3.1[48] 设C_1，C_2是子句集S的两个子句，C_{12}是它们的归结式，则

(1) 若用C_{12}代替C_1，C_2，得到新子句集S_1，则由S_1的不可满足可推出原子句集S的不可满足。即 S_1不可满足=>S不可满足。

(2) 若把C_{12}加入到S中，得到新子句集S_2，则由S_2与原S同不可满足。即

S_2不可满足<=>S不可满足。

算法3.1[48] (归结原理的一般性算法)

Step 1 将子句集S置入Clauses表中；

Step 2 若空子句NIL存在Clauses表，则归结成功(子句集S不一致)，结束。

Step 3 若Clauses表中存在可归结的子句对，则将其归结式并入Clauses表，转Step 2；若不存在可归结的子句对，则转Step 4；

Step 4 归结失败(子句集S一致)，结束。

此外，为提高归结算法的效率，人们提出了多种归结策略，如删除策略，支持集策略，线性归结策略，单元归结策略，祖先过滤策略等，具体可参考文献[36]。

附录 B 发表的相关论文

[1] 刘裕, 麦家键, 李磊等. 一种基于与或树结构的柔性 BOM 设计. 2004'全国软件与应用学术会议(NASAC)论文集. 北京：机械工业出版社, 2004. P522-526.

[2] 刘裕, 麦家键, 李磊. 基于与或树的柔性 BOM 结构及其产品配置算法. 计算机工程. 2005, 11.

[3] Yu Liu, Jia-jian Mai, Lei Li. A flexible BOM structure based on AND/OR tree with its product configuration and property verification. The 6th International Conference on Computer-Aided Industrial Design & Conceptual Design. Delft, the Netherlands, 2005.(Adopted)

致谢

首先感谢我的导师姜云飞教授，是他引导着我走完研究生三年的学习历程，从论文的选题到成稿，始终得到姜老师的悉心指导。姜老师知识渊博，治学严谨，同时为人正直、虚怀若谷，无论是做学问还是做人都给我留下了不可磨灭的印象，他的这些精神将是我学习的榜样，使我终生受益。

衷心感谢李磊教授，从我大二开始在软件所从事研发工作到现在研究生将近毕业，始终都得到李老师的指导和关怀。李老师渊博的知识，豁朗风趣的性格都给我刻下了深深的记忆烙印，本文的许多观点和思想都来自于李老师提供的实践项目，并且从李老师的亲身指导中也得到了不少启发。

感谢中山大学信息科学技术学院、计算机软件研究所、计算机科学系、电子通信与工程系的教师们，是他们的勤勤恳恳工作态度和孜孜不倦的教学精神，使我在求学期间掌握了许多终生受用的知识和技能。

感谢软件所的罗洁丽，学院办公室的田海云老师等信科院和软件研究所的工作人员们，是他们辛勤的工作，给我们创造了良好的学习和研究的环境。

感谢软件研究所的赵淦森、吴康恒、麦家健等博士硕士研究生们，在与他（她）们一起学习和工作的期间，得到了他们很多的帮助和照顾。

特别感谢我的父母，他们不仅在生活上给予我无微不至的关怀，而且在精神上也给了我很大的鼓励和安慰。他们对我事业上的理解和支持是我学习与生活的动力和保证。

最后，向所有关心和帮助过我的朋友表示最真挚的谢意！

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：刘格

日期：2005年 4月 15日