中图分类号：TP301　　论文编号：1028716 15-S038
学科分类号：081200

硕士学位论文

基于多值逻辑的软件产品线模型检测

研究生姓名　　石玉峰
学科、专业　　计算机科学与技术
研 究 方 向　　软件验证
指 导 教 师　　魏　欧 副教授

南京航空航天大学

研究生院 计算机科学与技术学院

二O一五年三月

Nanjing University of Aeronautics and Astronautics
The Graduate School
College of Computer Science and Technology

Multi-valued Model Checking of Software Product Lines

A Thesis in

Computer Science and Technology

by

Shi Yufeng

Advised by

Associate Prof. Wei Ou

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Engineering

March, 2015

承诺书

本人声明所呈交的硕士学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得南京航空航天大学或其他教育机构的学位或证书而使用过的材料。

本人授权南京航空航天大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本承诺书）

作者签名：_____

日　　期：_____

摘 要

软件产品线工程通过管理软件产品的可变性和共性特征，提高软件开发效率，节约开发成本。模型检测是一种自动形式化验证技术。随着软件产品线在安全关键领域的广泛应用，对软件产品线的模型检测已成为软件验证领域的一个重要研究方向。在软件产品线开发的早期阶段,由于特征的复杂性,系统设计中往往存在不确定信息。然而,现有的软件产品线模型检测对这种情况下的建模和验证支持不足。为此，本文提出一种基于多值逻辑对软件产品线的不完备设计进行建模和验证的方法，可以描述软件产品线设计初期存在的不确定信息，将模型检测提前到系统开发的早期阶段进行，及时发现错误，降低后期的修复成本。

本文的具体工作包括以下几个方面。首先，本文以世界双格作为理论基础，提出一种多值模型——基于双格的特征迁移系统，实现对包含不确定信息的软件产品线进行建模，并通过投影和精化关系定义特定软件产品的不完备模型和具体模型。然后，采用动作计算树逻辑描述系统的时序属性，定义动作计算树逻辑在基于双格的特征迁移系统上的语义，提出基于双格的多值模型检测。进一步，为实现基于双格的多值模型检测，一方面，提出从基于双格的特征迁移系统到多值 Kripke 结构的等价转换方法，开发了辅助工具 BPMCA，使得能够结合现有的多值模型检测工具 χ Chek 实现软件产品线的多值模型检测；另一方面，提出针对单个特征的分解方法，将多值模型检测问题分解为多个三值模型检测问题，降低模型检测的复杂度。最后，对文献中的软件产品线实例进行实验分析，验证了方法的有效性。

关键词：模型检测，世界双格，软件产品线，不完备模型

ABSTRACT

Software product line engeering (SPLE) manages commonalities and variabilities in software product lines to reduce cost and improve productivity, where commnalities and variabilites are expressed in terms of features. Model checking is an automatic verification technique. Nowadays, SPLE is widespread in industry, including critical areas like automotive and avionics. The emergence and the increasing popularity of SPLs have raised the need for model checking software product lines. Since the design decisions for a feature may be unkown in the early stage of software development, the systems may be with uncertainties. Currently, most attempts to address model checking problems of software product lines do not nicely support uncertain and inconsistent information and we consider model checking partial software product line designs, where incomplete information can be captured by multi-valued logics. This enables detecting design errors earlier, reducing the cost of later development of final products.

To this end, we first propose bilattice-based feature transition systems (BFTS) for modeling partial software product line designs, which support description of uncertainty and preserve features as a first class notion; the partial model and final model of product are defined via projection and simulation. We then express system behavioral properties using ACTL formulas and define its semantics over BFTSs. In this thesis, we investigates model checking software product lines basd on bilattice in two efficient ways: 1)we provide the procedures that translated BFTSs to multi-valued Kripke structure and develop a software model checker assistant BPMCA to leverage the power of exstiong model checking engine called χChek for verification,. 2)we decompose the multi-valued BFTS into 3-valued BFTS so that multi-valued model checking problems are decomposed into 3-valued model checking problems, which lower the complexity of model checking. Finally, we implement our approach and illustrate its effectiveness on a benchmark from literature.

Keywords：Model Checking, World-based Bilattice, Software Product Line, Partial Model

目 录

图表清单

注释表

L	格	D	德摩根代数
B	分配双格	$\bullet_t$	真值序关系
$\bullet_i$	信息序关系	$\hat{}$	真值序关系上的最大下界
_	真值序关系上的最小上界	$\otimes$	信息序关系上的最大下界
$\oplus$	信息序关系上的最小上界	$\|\|\varphi\|\|^M$	公式φ在模型M上的语义
B_W	世界双格	W	世界集合
$M_{\|P}$	产品P的不完备模型	$M(P)$	产品P的具体模型
χ Kripke	多值 Kripke 结构	H	基于世界双格多值模型精化关系
[	集合的并操作	\	集合的交操作
P	产品	¬	否定操作
;	空集	D	特征图
P(W)	集合W的幂集		

第一章 绪论

本章首先介绍软件产品线以及模型检测的研究背景和相关概念，从而引出对软件产品线的多值模型检测的探讨；其次就国内外有关软件产品线的研究情况作详细分析；最后给出本文研究的主要内容以及文章后续章节的组织结构。

1.1 研究背景

1.1.1 软件产品线

软件产品线是一组定义在公共核心资源的基础上，按照某种规定的方式开发的软件密集系统；这些系统共享一组公共的、可管理的、能够满足特定的市场或者任务需求的功能集合[1]。研发人员通过建立软件产品线生成不同种类的产品以满足不同用户的需求，并通过软件产品线分析产品间的共性以提高资源复用率。近年来，软件产品线逐渐成为软件工程的主要发展方向之一[1,2]，已广泛应用于通信、电力以及汽车等工业领域。来自实际应用中的数据显示，软件产品线提高了软件的质量以及缩短了产品上线的时间[3]。例如，柯达公司使用通用的组件和公共的生产流程重新组建了自己的产品线，使得它在富士公司推出第一款一次性快照相机之后很快夺回市场。基于这些成功案例，软件产品线被预测在未来很长一段时间内是软件工程发展的主导方向。

软件产品线上的产品之间既存在相同的部分，同时也存在着差异。描述产品之间共同的部分称为软件产品线的共性(commonality)，描述产品之间的差异称为软件产品线的可变性(variability)。在软件产品线中，可变性的管理成为了软件产品线工程中关键因素之一[2,4]。在软件产品线可变性的研究领域中，大部分研究都是基于特征的(feature-oriented)，即产品之间的差异采用特征描述；例如，P10 页中图 2.1 所示的特征图[5]是常用的一种特征建模方法，其中每个特征描述产品的一个功能或用户的某个需求[6]，特征以及特征之间的约束关系构成了一个树状结构。于是一个软件产品线可描述成一组特征的集合，产品可描述成软件产品线的一个子集。特征与系统行为之间存在密不可分的联系。一个软件系统所包含的所有行为来自于对其特征的设计，例如饮料机系统中的特征 *Tea*（茶）要求饮料机系统包含行为 *pour _tea*（倒茶），特征 *Freedrink*（免费饮水）要求饮料机系统不包含行为 *pay*（付款）。因此，当一个产品所包含的特征以及对于特征的设计已确定时，系统的所有行为已经确定。

目前，软件产品线的应用领域涉及航空、电子等关键系统，这就需要研究人员通过有效的检测方法确保软件产品线上的所有产品都满足既定的需求，即对于每一个产品，需要确保系统中的所有行为都是正确的。并且，当检测结果显示产品线不满足某个需求时，研究人员需要找出不满足需求的产品并且发现其中的错误，即与错误行为相关的特征或特征组合，从而完成系

统的修复。随着软件产品线工程的发展，软件产品线的规模不断增大，其验证问题成为了软件产品线工程的关键问题之一。

1.1.2 模型检测

模型检测[7]是一种自动形式化验证技术，它用于判断有限状态计算机系统的正确行为属性，其基本方法是用一个状态迁移图 M 来描述待检测系统，时序逻辑公式 φ 来描述系统的正确行为属性，然后通过对模型状态空间的穷举搜索来判断 φ 是否在 M 上被满足。如果 φ 在 M 上被满足，则系统的正确性得以证实(verified)；否则，表示系统中存在错误，系统的正确性被证伪(refuted)。与系统测试和习惯化推理等其他验证技术相比，模型检测的优点在于它能够检测系统模型所包含的所有行为，并能返回反例帮助找到错误产生的原因。模型检测技术如今已广泛应用于计算机硬件、控制系统、通信协议、安全认证协议等方面的系统分析与验证中，常用模型检测器包括 SPIN[8]、NuSMV[9]、Z3[10]等。

传统的模型检测基于传统布尔模型，在采用布尔模型建模的系统中，所有行为均是确定的。然而，在软件开发的初期，系统中包含不确定信息的情况不可避免：一方面，由于用户需求的不明确会出现信息不确定情况[11]，如用户对于手机是否需要“蓝牙”功能不确定；另一方面，由于系统涉及多个用户，他们对系统功能需求的不一致意见也会造成功能设计的不确定[12]，如在需求分析初期，A 类用户需要手机有“全键盘”，反对“触屏”，而 B 类用户要求手机是“触屏”，反对“全键盘”。这些不确定信息在布尔模型中无法描述。因此，传统的模型检测适用于软件开发的后期阶段，即系统中不存在不确定的信息。相比于软件开发的前期阶段，在后期阶段通过模型检测来修复系统错误的成本更高。

多值模型[13]是布尔模型的扩展。与布尔模型相比，多值模型更适合描述包含不确定信息的软件系统行为[14]。多值逻辑作为多值模型的理论基础，可以对不确定信息进行描述，其逻辑值作为多值模型检测结果，反映属性在系统上满足的真假程度。典型的多值逻辑可以用德摩根代数表示，德摩根代数通过定义逻辑值之间的真值序关系比较逻辑值之间的真实程度。例如三值逻辑[15](f,m,t)（如图 2.5 所示），f 、m 、t 代表三个逻辑值，在真值序关系上，三个逻辑值满足 $f < m < t$，即 t 和 f 分别表示真值上的“真”和“假”，m 则表示“可能”，用以描述系统中的不确定情况。

由于德摩根代数只包含真值序关系，而在实际应用中，有时需要描述系统中所包含信息量的多少，比如在需求分析阶段，对于软件的某个功能，除了要考虑到反对和支持的用户，还需要考虑到对此不确定或与此不相关的用户。为此，在对系统的多值建模中，需要定义逻辑值在信息上的精确程度，从而描述信息量的多少。为此，Fitting 等人[16]提出了分配双格(distributive bilattice)。在双格结构中，逻辑值之间存在两种序关系：真值序关系 $\leq_t$ 与信息序关系 $\leq_i$；其中，真值序关系可以反映信息的真假程度，信息序关系可用来描述信息的精确程度。世界双格[17]是

一种常用的分配双格，可以从实际角度帮助我们理解多值逻辑。例如：设命题 p 表示“需要系统中具有某个功能”，而对于每一个用户而言， p 的逻辑值可以为“真”、“假”或者“未知”。则当存在多个用户时，不同的用户可以持不同的观点。假如用 W 表示所有的用户群体，；表示空集，那么我们可以用有序对 $\langle U,V\rangle$ 来表示 p 的真值，其中 U 和 V 都是 W 的子集，且 U 对应要求 p 为真的用户集合而 V 对应要求 p 为假的用户集合。于是，在信息序关系上，$\langle ;,;\rangle$ 所包含的信息量最低，$\langle W,W\rangle$ 所包含的信息量最高，由此构造出基于世界双格的多值逻辑以描述信息量的多少。

1.1.3 软件产品线的模型检测

软件产品线的模型检测问题比单个系统的模型检测问题更为复杂。从模型检测的基本方法来看，实现软件产品线的模型检测需要判断软件产品线上每一个产品的正确行为属性，其中每一个产品可视为一个单个系统，这就意味着实现软件产品线的模型检测相当于实现多个单个系统的模型检测。然而软件产品线上的产品数量是随着特征的个数呈指数增长的，即若一个产品线包含特征个数为 n，则产品数量最多可达到 2^n [18]。这使得通过逐一检测单个产品系统的方式难以实施。

目前已有相关工作对软件产品线的模型检测进行了研究，实现了在软件产品线层面上对产品属性的检测。大部分研究是基于模态迁移模型(Modal Transition Systems，简称 MTS)[19]。在 MTS 中，系统的所有行为(behavior)采用迁移(transition)描述。迁移的种类分为必选型和可选型，其中必选型迁移表示产品系统中必须存在的迁移，用来描述软件产品线的共性。可选型迁移表示产品系统中允许存在的迁移，用来描述软件产品线中的可变性。这些方法实现了对软件产品线共性和可变性的描述，却忽略了软件产品线的特征对系统行为的影响，导致它们的模型检测结果缺乏产品属性与单个特征或多个特征的组合之间的联系，使得开发人员无法通过模型检测发现不满足属性的产品。基于此，Classen 等人[20-23]将系统行为与特征联系起来，提出特征迁移系统(Featured Transition System，简称 FTS)，使得当软件产品线不满足属性时，模型检测结果返回导致产品不满足属性的特征或特征的组合，从而找到相关产品。

然而以上的软件产品线模型检测方法更适用于软件产品线开发的后期，即软件产品线的所有信息都已明确。在软件产品线开发的初期，系统中存在不确定情况。例如，对特征设计的不确定，主要包括：1)特征与系统中某些行为之间的联系不确定，如手机系统中特征 *text*（发短信）不确定是否需要系统包含行为 *wifi _connected*（连接无线网）；2)特征之间就某个系统行为产生互斥，比如手机系统中特征 *fly _mode*（飞行模式）反对系统行为 *wifi*，特征 *explore*（浏览器）需要系统行为 *wifi _connected* 。这些在特征设计上的不确定影响最终产品的系统行为，进而影响产品的属性。若在建模中描述这些不确定信息，可以尽早地通过模型检测结果发现它们对系统属性的影响，并能尽早发现系统中的错误，降低修复成本。然而，在 FTS 中系统行为与特征之间的关系是二值的，即所有的行为和特征之间的关系都是明确的，这使得基于 FTS 的

模型检测需要在软件产品线开发的后期即特征设计已确定的情况下进行。同时，相比于软件产品线开发的初期，在后期对系统修复的成本更高。

多值模型作为布尔模型的扩展更适用于描述软件产品线中的不确定信息，其多值逻辑值可用来描述信息的确定程度，如世界双格中的信息序关系可以用来反映系统行为与特征之间的不确定情况。基于此，本文提出一种基于双格的多值模型用于描述软件产品线中的不确定信息，实现对软件产品线的不完备建模，将软件产品线的模型检测提前至软件产品线开发的初期，从而尽早发现系统中的错误，降低修复成本，并且其模型检测结果可以反映这些不确定信息对最终产品属性的影响。

1.2 研究现状

近年来，在软件产品线建模的研究中，对可变性建模的研究越来越广泛。文献[24]选取近20年发表的论文进行系统综述，对可变性建模方法的分类和相关技术等展开深入探讨。大部分对软件产品线可变性的研究主要基于特征建模，并且以 Kang 等人[5]提出的基于特征的领域分析(Feature Oriented Domain Analysis，简称 FODA)作为理论基础。在此基础上，已有相关研究提出了各种扩展方法用以弥补特征图描述能力的局限性[25]。特征是指系统或系统中用户可见的，显著或者与众不同的方面、品质或特点[5]。一方面，特征用以描述用户的某种需求，例如文献[12]利用特征模型对单个客户对产品的观点(view)进行建模，并考虑到不同客户对产品要求的不一致性，采取暂时保留不同意见的方法，以保证产品线中存在满足不同客户要求的产品；文献[26]提出针对给定用户需求对产品线特征图的裁剪方法，从而使得特征建模支持不同用户的需求配置。另一方面，特征可用于描述产品的功能，例如文献[19]基于可变性的管理策略，提出了以扩展的用例(use case)模型和特征模型为表现形式的变化性建模；文献[18]研究了由于功能的设计导致特征之间存在的相互作用。因此，特征成为了连接用户需求和产品功能的桥梁[27]。

另一部分研究是基于软件产品线的行为建模，大部分借助 Fischbein 等人[28]提出的模态迁移模型(Modal Transition Systems，简称 MTS)。例如文献[29]在 MTS 的基础上提出了广义扩展模态迁移模型(Generalized Extended Modal Transition Systems)，提高了行为建模的描述能力；文献[30]采用标签结构(structured labels)扩展了 MTS，提出标记模态迁移模型(label-structured modal transition system)。然而，这些建模方法忽略了特征与系统行为之间的联系，导致通过软件产品线模型得到产品模型与实际产品不相符。为此，Classen 等人[20-23]将特征与系统行为联系起来，提出 FTS，但由于在 FTS 中，特征与系统行为的关系是确定的，因此 FTS 不能很好地支持软件产品线设计初期所出现的信息不确定情况。此外，还有一部分研究是基于 UML 的建模方法，如文献[31]提出基于 UML 的建模方法，通过扩展已有的 UML 类型（状态机、时序图等）以实现软件产品线上的行为建模；文献[32]实现了通过产品线的模型得到单个产品的模型。然而他们并没有在软件产品线的模型上实现最终产品的属性验证。

如今，随着用户需求的增多，导致软件产品线规模的增大，这使得软件产品线的形式化验证逐渐引起了关注。在文献[33]中，Larsen 等人[33]基于 I/O 自动机，提出了一种行为建模方法以验证可结合性接口的错误。但此方法不涉及软件产品线的行为属性的验证。软件产品线的模型检测问题不同于单个软件系统的模型检测，由于软件产品线包含上百乃至上千个产品[18]，实现软件产品线层面上的模型检测是软件产品线模型检测问题的主要方法之一。为此，Gruler 等人[34]扩展了进程代数 CCS，提出了 PL-CCS 以支持软件产品线中两个进程之间的异或关系从而完成在软件产品线层面上对所有产品的属性检测，然而他们的模型检测方法并没有结合相关工具通过相关实验验证其有效性。Asirellis 等人[35,36]提出的模型检测方法采用 MTS 描述系统行为，采用 MHML 逻辑描述系统属性以及得到产品模型时需要添加的约束关系，从而实现了从软件产品线模型得到产品模型，为此他们开发了相应的模型检测工具 VMC 实现 MHML 公式在 MTS 上的模型检测。但他们并没有考虑系统行为与特征之间的联系，且并没有使用特定的建模语言。在 Classen 等人[20-23]提出的 FTS 建模方式中，软件产品线中的每一个行为与特征相关联，从而保证了得到有效的产品模型。并且，其模型检测结果可得到所有不满足属性的产品。基于此，他们开发了相应的验证工具 SNIP 和 ProveLines 以实现该模型检测。但该建模方式中，每个行为只能和一个特征相关联，导致不能很好地支持软件产品线中信息不确定和不一致的情况。

多值模型检测是传统的二值模型检测在多值逻辑上的扩展，适用于软件系统设计初期即软件系统中存在不确定信息的情况。目前已有相关工作对多值模型检测问题进行了研究，并开发了多值模型检测工具，如 STE[37]——主要应用于硬件的模型检测，以及 YASM[38]——主要应用于软件的模型检测。此外，Chechik 等人[38-40]提出了准布尔多值逻辑用于检测包含不确定和不一致信息的系统，并开发了多值模型检测工具 χChek。Salay 等人[11,41]采用不完备模型描述系统中不确定信息并且应用于软件系统中用户需求不确定的管理。但这些并没有直接应用到软件产品线领域中。为此，本文研究了软件产品线的多值模型检测，针对软件产品线初期出现的不确定情况，提出一种多值迁移系统以描述特征设计的不确定情况，并结合已有多值模型检测工具通过相关实验验证其有效性。

1.3 本文研究内容

本文主要研究基于双格的软件产品线多值模型检测，其研究内容主要包括以下两个方面：

1) 软件产品线的不完备建模；本文提出一种多值建模方法，描述软件产品线中存在的不确定信息，并定义产品线上产品的不完备模型和具体模型。
2) 软件产品线的多值模型检测；本文采用动作时序逻辑描述系统属性，定义了动作时序逻辑在多值逻辑上的语义。接着，本文提出两种方法以实现软件产品线的不完备多值模型检测：一种是通过模型转换，从而运用现有的多值模型检测工具完成多值模型检测；另一种是通过模型分解，从而将多值模型检测问题分解为多个三值模型检测问题。

下面，将对以上两个方面作详细分析。

1.3.1 软件产品线的不完备建模

在软件产品线的开发初期，特征的设计存在不确定情况，即特征与系统行为之间的关系不确定。传统的特征迁移系统并没有考虑到这一点，在该建模中特征与行为的关系是确定的，且一个行为只与一个特征相关。本文将传统的特征迁移系统扩展到多值逻辑上，提出一种多值模型——基于双格的特征迁移系统(Bilattice-based Featured Transition Systems，简称 BFTS)，实现对软件产品线的不完备建模。

在 BFTS 中，每一个迁移被标记(label)了一个世界双格 $\mathbb{B}_W$ 中的逻辑值；逻辑值是 $\langle U,V\rangle$ 这样的有序对形式，其中 U 表示需要该迁移的特征集合，V 表示排斥该迁移的特征集合，我们不考虑特征与系统行为关系不一致情况，即对于产品线的 BFTS 中任一迁移的逻辑值 $\langle U,V\rangle$，满足 $U \setminus V = ;$，则 $W / (U \lceil V)$ 表示与该迁移的关系不确定的迁移集合。例如，手机的软件产品线的特征集合 W 为 $\{explore(浏览器), text(短信), fly\_mode(飞行模式)\}$，迁移 $(s, wifi\_connected, s')$ 描述系统行为“连接无线网”；假设特征 *explore* 需要该行为，而特征 *fly_mode* 反对该行为，特征 *text* 与该行为的关系不确定，则迁移 $(s, wifi\_connected, s')$ 的逻辑值为 $\langle\{explore\},\{fly\_mode\}\rangle$。通过给迁移标记逻辑值的方式，特征与行为之间的不确定关系得以描述。此外，我们定义了投影和精化关系的概念，其中软件产品线模型可通过投影得到产品的不完备模型，产品的不完备模型通过精化关系得到产品的具体模型。

BFTS 主要解决了传统的 FTS 中的 2 个局限性：一方面，特征与迁移之间的关系不再局限于“一对多”，即一个特征可以与多个迁移相关而一个迁移只能与一个特征相关，基于多值逻辑，特征与迁移之间的关系可以是“多对多”，即一个特征可以与多个迁移相关且一个迁移可以与多个特征相关；另一方面，在传统的特征迁移系统中特征与迁移之间的关系只有“需要”或“不需要”，即标记在迁移上的特征视为需要该迁移，而其他特征视为不需要，而在基于双格的特征迁移系统中，特征迁移之间的关系可分为“需要”、“排斥”以及“不确定”。因此，BFTS 更好地支持了软件产品线中存在的不确定情况。

1.3.2 软件产品线的多值模型检测

在软件产品线的属性验证方面，本文选择一种基于动作的时序逻辑——动作计算树逻辑(Action Computation Tree Logic，简称 ACTL)[42]描述软件产品线的时序逻辑属性。由于传统的 ACTL 语义是定义在标记迁移系统(Labeled Transition System，简称 LTS)[43]上，并不适用于多值模型，于是本文给出了 ACTL 在多值模型上的语义。

在实现基于 BFTS 的多值模型检测方面，本文采用了两种不同方法：一种是采取直接进行多值模型检测的方法，提出从 BFTS 到多值 Kripke 结构——χ Kripke 结构的等价转换方法，以

实现从 BFTS 到χKripke 结构的自动转换，将基于 BFTS 的多值模型检测问题转换为基于χKripke 结构的多值模型检测问题，从而结合现有的多值模型检测工具χChek[40]实现软件产品线的多值模型检测；另一种是采取将多值模型检测问题转换为三值模型检测问题的方法，提出多值 BFTS 相对于单个特征的分解方法，将多值 BFTS 上的模型检测问题分解为多个三值 BFTS 上的模型检测问题。

为了验证方法的有效性，我们在文献[29]饮料机产品线例子的基础上设计了相关实验。首先开发辅助工具——基于双格的软件产品线的不完备模型检测工具(Bilattice-based Partial Model Checker Assistant of Software Product Line，简称 BPMCA)——以实现从 BFTS 到χKripke 结构的自动转换；其次，结合χChek 完成饮料机产品线的多值模型检测；接着，将饮料机产品线的BFTS相对于特征集合中每一个特征分解成多个三值BFTS，将ACTL公式在这些三值BFTS上进行验证，实验结果证实公式在原多值 BFTS 上验证的结果等于在这些三值 BFTS 验证结果的并(⊕)；最后，通过比较直接进行多值模型检测和将多值模型检测分解成多个三值模型检测这两种实现方法的时间、环境要求等，得出基于模型分解的多值模型检测降低世界双格的规模，从而降低模型的导入时间以及模型检测工具的运行环境要求等。

1.4 本文组织结构

本文其他部分的组织如下：第二章概述相关的预备知识，包括软件产品线的可变性建模中的特征建模和行为建模，以及多值模型检测中的世界双格，χKripke 结构和 CTL 时序逻辑等；第三章介绍软件产品线的不完备建模，包括基于双格的特征迁移系统的以及产品的不完备模型和具体模型；第四章介绍软件产品线的多值模型检测，首先给出 ACTL 逻辑在 BFTS 上的语义以及模型检测结果的含义，接着介绍从BFTS到χKripke结构的转换方法和从ACTL公式到CTL公式的转换方法，从而实现将 BFTS 上的多值模型检测问题转化为χKripke 结构上的多值模型检测问题，最后介绍 BFTS 相对于单个特征的分解方法，从而实现将多值 BFTS 上的模型检测问题分解成多个三值 BFTS 上的模型检测问题。第五章介绍实验设计与分析，包括现有的多值模型检测工具χChek 工作以及开发的辅助工具 BPMCA，最后通过实验验证方法的有效性。

第二章 预备知识

本章主要介绍软件产品线以及多值模型检测的基本知识。首先介绍软件产品线的可变性建模，包括特征图、迁移系统、特征迁移系统等；然后介绍多值模型检测，包括世界双格、χ Kripke 结构、计算树逻辑(Computation Tree Logic，简称 CTL)[44]等。

2.1 软件产品线的可变性建模

软件产品线上的产品之间既存在相同或相似的部分同时也存在着差异。共性用于描述产品之间的相同部分，可变性则用于描述产品之间的差异，这是软件产品线工程区别于其他软件开发过程的重要特征[24]。可变性建模方法主要包括两类：特征建模(Feature Modeling)和行为建模(Behavioral Modeling)。特征建模侧重于描述软件产品线中的特征以及特征之间的约束关系，行为建模侧重于描述软件产品线的系统行为。

2.1.1 特征与特征图

基于特征的可变性建模是最早用于可变性建模的方法[27]。通常情况下，产品的每个功能可以看作是一个特征。例如，手机里的“指纹解锁”的功能就是一个特征。产品线上产品之间的异同可以反映在所包含的特征的异同上。例如，手机产品 P_1 有特征 *blueteeth*（蓝牙），*fly _ mode*（飞行模式），*text*（短信），手机产品 P_2 有特征 *fly_mode*（飞行模式），*text*（短信），*camera*（照相），则产品 P_1 与产品 P_2 的共性是特征 *fly _ mode* 和 *text*，即都有“飞行模式”和“短信”功能，而可变性体现在特征 *blueteeth* 和 *camera*，即 P_1 有“蓝牙”功能而 P_2 没有，P_2 有“照相”功能但 P_1 没有。

软件产品线所包含的特征之间存在着各种约束关系，包括互斥、依赖等。从理论上来说，用户根据特征之间的约束关系来选择特征即可得到有效的产品。例如，手机系统产品线的特征 *fingerprint_recognition* (指纹解锁)与特征 *image_recognition*（图像解锁）互斥，即这两个特征不能同时出现在产品中；再比如若用户选择了手机产品包含特征 *text* (信息)，就必须要从特征 *keyboard* (键盘输入)，特征 *voice* (语音输入)中选择一个。

特征图由 Kang 等人[5]于 1990 年提出用于描述软件产品线的功能需求，目前大部分基于特征建模的研究都是以此为理论基础，其定义如下：

定义 2.1[27] 特征图(Feature Diagram，简称 FD)是一种以图的形式表示的基于特征集合 F 的树状特征模型 $D=(N,r,DE)$，其中：

1) N 是树的结点集合；
2) $r \in N$ 是树的根结点；

3) *DE* 是树的边集合；

N 中一个结点是 *F* 中的一个特征， *DE* 中每条边代表一个特征与子特征之间的约束关系。约束关系的类型分为 *madatory* (必选)、 *optional* (可选)、 *or* (或)、 *alternative* (多选一)4 种，这 4 种约束关系有如下定义：

$$\text{root}(f) \equiv f$$
$$\text{mandatory}(p, f) \equiv f \Leftrightarrow p$$
$$\text{optional}(p, f) \equiv f \Rightarrow p$$
$$\text{alternative}(p,\{f_1,\ldots f_n\}) \equiv ((f_1 \vee \ldots \vee f_n) \Leftrightarrow p) \wedge \bigwedge_{i<j} \neg(f_i \wedge f_j)$$
$$\text{or}(p,\{f_1,\ldots f_n\}) \equiv (f_1 \vee \ldots \vee f_n) \Leftrightarrow p$$

其中 p, f, f_i 为 *F* 中的元素。

特征图 *D* 可以等价一组命题公式，其中每一个命题公式代表一个约束关系。例如对于图 2.1 所示的特征图，其等价于以下命题公式：

$$\begin{cases} \text{root}(v) \\ \text{manadatory}(b,v) \\ \text{manadatory}(f,v) \\ \text{alternative}(f,\{d,r\}) \\ \text{or}(b,\{c,t\}) \end{cases}$$

例 1 饮料自动售货机(简称饮料机)是软件产品线研究领域中代表性例子之一，最早在文献[29]中用以介绍模态迁移系统(Modal Transition System，简称 MTS)对软件产品线的可变性建模。本文根据需要，对原有的饮料机例子进行修改，设计饮料机自动售货机的基本工作流程如下：饮料机初始状态为待机状态 $\rightarrow$ 顾客投进硬币 $\rightarrow$ 顾客选择饮料是否含糖 $\rightarrow$ 顾客选择饮料的种类 $\rightarrow$ 饮料机根据顾客的选择提供相应饮料 $\rightarrow$ 注入饮料完成 $\rightarrow$ 饮料拿走后回到初始状态。

以上是饮料机产品线的共性，即该产品线上的所有产品均有这些行为。饮料机产品线的可变性如下：

1) 饮料机至少提供一种饮料：咖啡或茶；

2) 饮料机提供提醒功能：显示“完成”或响铃。

饮料机产品线的特征图如图 2.1 所示。图中每一个特征均用一个小写字母作为名称缩写，如 *c* 代表特征“咖啡(*coffee*)”。特征 *v* 是特征图的根特征，即饮料机产品线上所有的产品均包含特征 *v* 。特征 *v* 有两个子特征：特征 *b* 和特征 f ，两者都是必选特征，即如果选择了特征 *v* ，那么必须选择特征 *b* 和特征 f 。特征 *b* 与子特征 *c* 、 *t* 的关系为 *or* (或)，即如果选择了特征 *b* ，那么至少选择其中一个子特征。特征 f 与子特征 *d* 、 *r* 的关系为 *alternative* (多选一)，即如果选择了特征 f ，那么必须且只能选择其中一个子特征。

特征图实现了通过特征以及特征之间的约束来描述软件产品线。在特征建模中，通过软件产品线的特征集合中的子集得到一个产品，这个子集叫作特征选择(feature selection)[27]。然而，并不是每一个特征选择都是有效的(valid)。以例 1 中饮料机产品线为例，特征 d 与特征 r 为互斥关系，即不能同时出现在一个产品里，于是包含这两个特征的特征选择如 $\{v,b,c,f,r,d\}$ 是无效的。于是，我们给出产品的定义如下：

定义 2.2[27] 设产品线的特征集合 F，若存在集合 P 满足 $P \subseteq F$ 且 P 中的特征满足 F 中的特征约束关系，称 P 是 F 的**产品**。

以例 1 中饮料机产品线为例，其特征图如图 2.1 所示。根据特征图的语法，饮料机的产品线的特征集合为 $\{v, f, b, r, c, t, o\}$，产品线上的所有产品为：

$$\{\{v,b,f,c,d\},\{v,b,f,c,r\},\{v,b,f,t,d\},\{v,b,f,t,r\},\{v,b,f,c,t,d\},\{v,b,f,c,t,r\}\}。$$

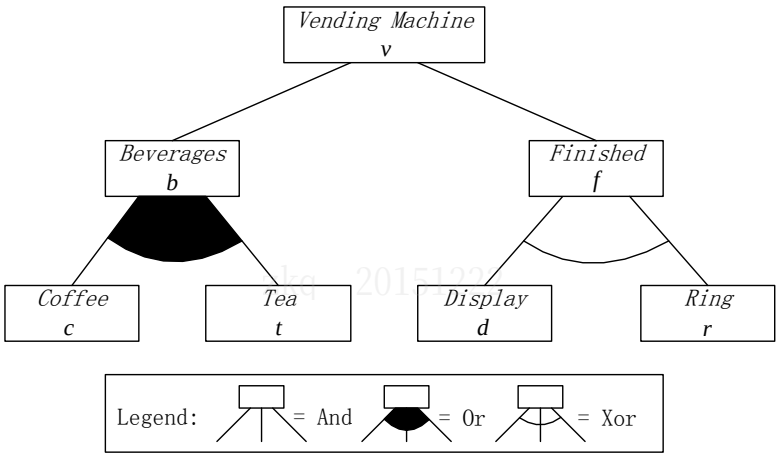


图 2.1 饮料机产品线的特征图

2.1.2 迁移系统与特征迁移系统

软件产品线的行为模型用于描述系统行为，即系统处于某个状态时对某个事件的反应。迁移系统(Transition System，简称 TS)[45]是常用的行为模型之一。该模型是一种有向图。在 TS 中，迁移采用动作(action)标记且状态采用原子命题标记，其定义如下：

定义 2.3[45] **一个迁移系统(Transition System，简称 TS)**是一个六元组：$M = (S, Act, trans, I, AP, \mathrm{L})$，其中：

1) S 是一个有穷状态集；
2) Act 是一个行为集；
3) $trans \subseteq S \times Act \times S$ 是一个迁移集；
4) $I \subseteq S$ 是一个初始集；

5) AP 是一个原子命题集；

6) $\mathrm{L}: S \to 2^{AP}$ 是一个标签函数。

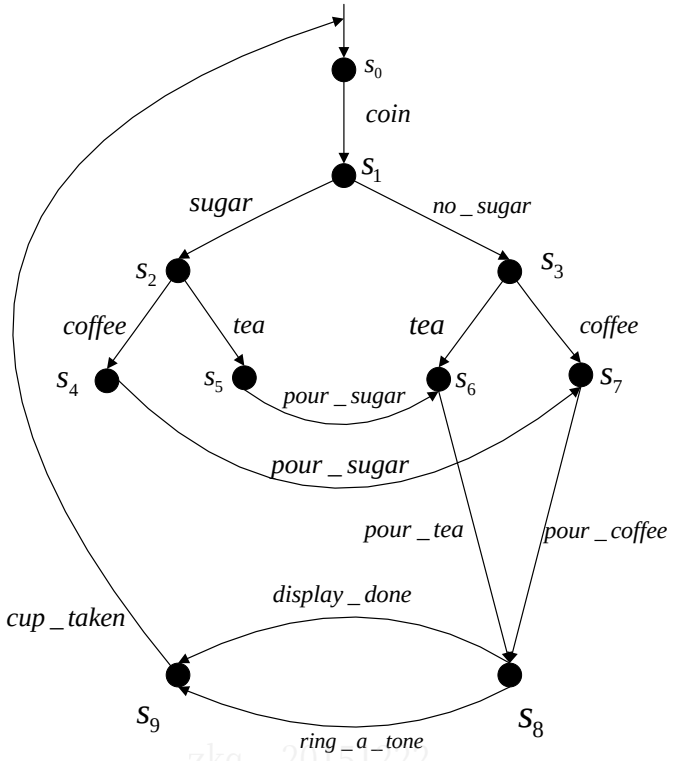


图 2.2 饮料机产品线的 TS（原子命题在此省略）

对于 TS M，迁移 (s,a,s') 表示系统处于状态 s 时，通过动作 a 到达状态 s。在饮料机产品线的 TS 中(如图 2.2 所示)，迁移 $(s_0,coin,s_1)$ 描述当系统处于初始状态 s_0 时，通过动作 $coin$（“顾客投进硬币”）到达状态 s_1。然而，迁移系统忽略了软件产品线的特征与系统行为之间的联系，于是采用迁移系统对软件产品线建模，无法通过所选特征得到相应的产品的行为模型，并且在产品线的 TS 上得到的模型检测结果无法返回不满足该属性的产品，不能实现在软件产品线层面上对所有产品完成模型检测。为此，Classen 等人[20-23]考虑到特征与行为之间的联系，在迁移系统基础上，提出特征迁移系统(Featured Transition System，简称 FTS)。在 FTS 中，每条迁移采用特征标记(label)，使得系统行为与特征相联系。其定义如下：

定义 2.4[20] 一个**特征迁移系统(Featured Transition System，简称 FTS)**是一个八元组：$M=(S,Act,trans,I,AP,\mathrm{L},D,\gamma,>)$，其中：

1) $(S,Act,trans,I,AP,\mathrm{L})$ 是一个迁移系统；

2) D 是一个特征图；

3) γ 是标签函数，是 $trans \to N$ 上的映像；

4) $> \subseteq trans \times trans$ 是一个偏序关系，定义迁移之间的优先级。

对于 FTS M，若迁移 (s,a,s') 的标签特征为 v，即 $s \xrightarrow{a/v} s'$，则表示该迁移被特征 v 需要。如饮料机产品线中迁移 $(s_0,coin,s_1)$ 被特征 v 需要，则 $s_0 \xrightarrow{coin/v} s_1$ (如图 2.3 所示)。此外，FTS 还描述了迁移之间存在的优先级关系，若迁移 (s_1,a_1,s_1') 与迁移 (s_2,a_2,s_2') 满足优先级关系 $s_1 \xrightarrow{a_1/v_1} s_1' > s_2 \xrightarrow{a_2/v_2} s_2'$，则表示若产品包含特征 v_1 与特征 v_2，则在该产品中不会出现迁移 (s_2,a_2,s_2') 所描述的系统行为。

最终产品的行为模型可通过产品线的行为模型 FTS 在产品的特征集合上投影得到，投影通过如下步骤实现：1)去掉与产品中特征无关的迁移；2)去掉由于优先级原因被产品中特征有关的迁移覆盖的迁移。其定义如下：

定义 2.5[20] 设产品线的行为模型 FTS $M=(S,Act,trans,I,AP,L,d,\gamma,>)$，$P$ 是一个产品，则产品 P 的行为模型 TS $M'=(S,Act,trans',I,AP,L)$ 由 M 在 P 上**投影**得到，记作 $M_{|P}$，其中：

$$trans'=\{(s_1,a,s_2)\mid (s_1,a,s_2)\in trans \ \hat{} \ \gamma(s_1,a,s_2)\in P \ \hat{} \ \nexists(s_1,a',s_2')\in trans\cdot\gamma(s_1,a',s_2')\in P \ \hat{} \ (s_1,a',s_2')>(s_1,a,s_2)\}.$$

以例 1 中饮料机软件产品线为例，饮料机产品线的行为模型如图 2.3 所示。设产品 $P=\{v,b,f,c,r\}$，根据定义 2.5，由于迁移 (s_2,tea,s_5)，$(s_5,pour\_sugar,s_6)$，$(s_6,pour\_tea,s_8)$ 以及 $(s_8,display\_done,s_9)$ 与产品所选的特征无关，它们在产品的行为模型中被删去。最终我们得到产品 P 的行为模型如图 2.4 所示。

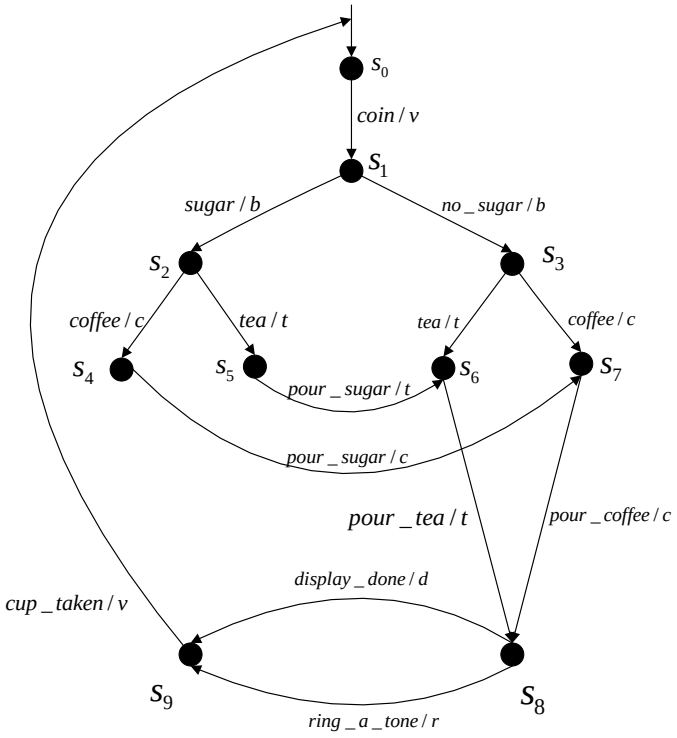


图 2.3 饮料机产品线的 FTS（原子命题在此省略）

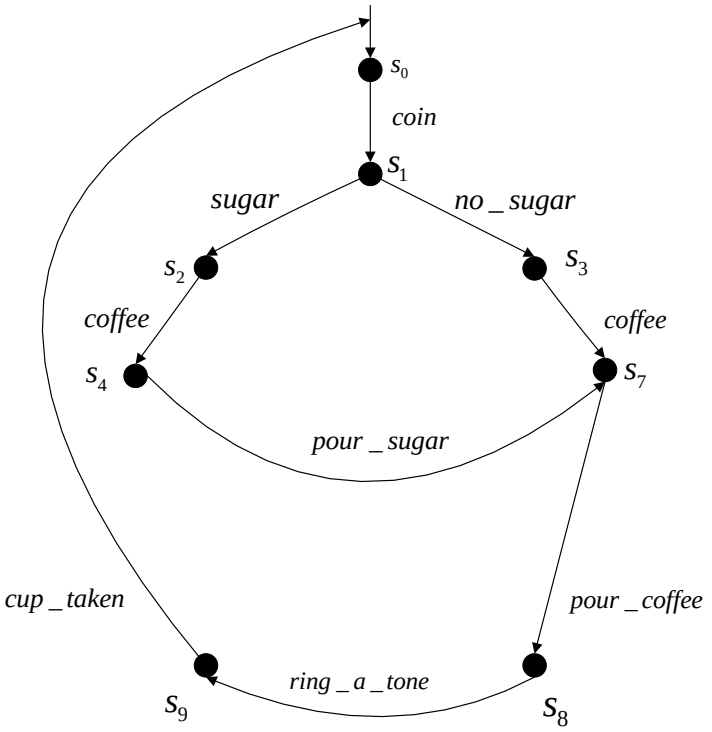


图 2.4 产品 $P=\{v,b,f,c,r\}$ 的行为模型（原子命题在此省略）

2.2 多值模型检测

本节主要介绍多值模型检测的基本知识。多值模型检测是传统模型检测的扩展，典型的多值逻辑采用德摩根代数 L 表示，多值模型采用多值 Kripke 结构表示。与传统的模型检测相比，多值模型更适合对包含不确定和不一致信息的软件系统进行建模；多值逻辑作为多值模型的理论基础，提供了除了 true（真），false（假）之外其他逻辑值用来表示信息的真假程度。在多值模型检测中，通过给定一个定义在德摩根代数 L 上的模型 M 和一个时序属性 φ，结合多值模型检测工具可实现验证 φ 在 M 上的真假程度。

2.2.1 双格与世界双格

本节主要介绍多值逻辑的相关背景知识，首先介绍格、分配格以及德摩根代数，然后介绍双格与世界双格，其中世界双格是我们提出的软件产品线建模方法的理论基础。

设 $L=(L,\leq)$ 为偏序集，若对任意 $C\subseteq L$，存在最小上界(join)与最大下界(meet)，分别表示为 $\_C$ 与 $\hat{}C$，则称 L 为**格**。若格中运算 _ 与 $\hat{}$ 操作满足分配律，即 $\forall a,b,c\in L$，有 $a\hat{}(b\_c)=(a\hat{}b)\_(a\hat{}c)$ 且 $a\_(b\hat{}c)=(a\_b)\hat{}(a\_c)$，则称 L 为**分配格**。

定义 2.6[46] 一个**德摩根代数**是一个三元组：$L=\langle L,\hat{o},\neg\rangle$，其中 $\langle L,\hat{o}\rangle$ 是一个有限分配格，$\neg$ 是取反操作并满足 $\neg\neg a=a$ 和德摩根定律：$\neg(a\hat{}b)=\neg a\_\neg b$ 与 $\neg(a\_b)=\neg a\hat{}\neg b$。

比如，基于三值德摩根代数的格（如图 2.5 所示）包含逻辑值 t，f 和 m，其取反操作满足 $\neg t = f$，$\neg f = t$ 和 $\neg m = m$。

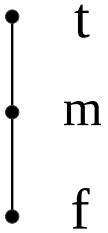


图 2.5 三值逻辑

德摩根代数表示的多值逻辑只包含一种序关系，我们称其为真值序关系。为了不仅可以表示多值的真假程度，还可以表示多值逻辑包含信息量的多少，Fitting 等人[16]提出分配双格的概念，其中逻辑值之间不仅存在真值上的序关系还有信息上的序关系，即逻辑值本身不仅存在真值方向上的含义也存在信息方向上的含义。其定义如下：

定义 2.7[47] 一个分配双格是一个结构 $\mathrm{B} = \langle B_W, \leq_i, \leq_t, \neg \rangle$，其中：

1) $\mathrm{B}_i = \langle B, \leq_i \rangle$ 是格且 $\mathrm{B}_t = \langle B, \leq_t, \neg \rangle$ 是德摩根代数；
2) B_i 中的 $\otimes$ 与 $\oplus$，B_t 中的 $\wedge$ 与 $\vee$ 都相对于 $\leq_i$ 与 $\leq_t$ 单调；
3) $\otimes$ 与 $\oplus$，$\wedge$ 与 $\vee$ 相互分配；
4) $\neg$ 相对于 $\leq_i$ 单调。

$\leq_t$ 和 $\leq_i$ 分别代表真值序关系和信息序关系。通常情况下，真值序关系 $\leq_t$ 上最大元与最小元分别表示为 true（真）与 false（假）；信息序关系 $\leq_i$ 上最大元与最小元分别表示为〉（信息不一致）与？（信息不确定）。

我们提出的软件产品线行为建模方法是基于世界双格的。世界双格是定义在一个已有世界集合 W 上的双格，其中一个世界可以用以描述一个事物。如在软件产品线中，一个特征可以看作一个世界。于是 W 可以看作产品线的特征集合。

定义 2.8[17] 设 W 为世界的集合，$\mathrm{P}(W)$ 为 W 的幂集。**世界双格(World-based Bilattice)**为 $\mathrm{B}_W = \langle \mathrm{P}(W) \times \mathrm{P}(W), \leq_t, \leq_i, \neg \rangle$，对于任意的 $\langle U_1, V_1 \rangle, \langle U_2, V_2 \rangle \in \mathrm{P}(W) \times \mathrm{P}(W)$，有以下条件成立：

$$\langle U_1, V_1 \rangle \leq_t \langle U_2, V_2 \rangle \Leftrightarrow U_1 \subseteq U_2 \wedge V_2 \subseteq V_1$$
$$\langle U_1, V_1 \rangle \leq_i \langle U_2, V_2 \rangle \Leftrightarrow U_1 \subseteq U_2 \wedge V_1 \subseteq V_2$$
$$\neg \langle U_1, V_1 \rangle \Leftrightarrow \langle V_1, U_1 \rangle$$

其中，$\leq_t$ 和 $\leq_i$ 分别表示真值序关系与信息序关系。

图 2.6 所描述的是逻辑值个数最小的世界双格 $\mathrm{B}_{\{w\}}$，即定义在只包含一个元素 w 的世界集合上的世界双格。$\mathrm{B}_{\{w\}}$ 共包含 4 个元素：$\langle \{w\}, \{w\} \rangle$，$\langle ;, \{w\} \rangle$，$\langle \{w\}, ; \rangle$ 以及 $\langle ;, ; \rangle$，其中 $\langle \{w\}, ; \rangle$

代表真值序关系方向上的最大元——true（真），$\langle ;,\{w\}\rangle$ 代表真值序关系方向上的最小元——false（假）；$\langle \{w\},\{w\}\rangle$ 代表信息序关系方向上的最大元——inconsistency（不一致），$\langle ;,;\rangle$ 代表信息序关系方向上的最小元——unknown（未知）。此外，对于 $\langle U,V\rangle$，我们定义操作：π_t 和 π_f。记 $\pi_t(\langle U,V\rangle)=U$，$\pi_f(\langle U,V\rangle)=V$。

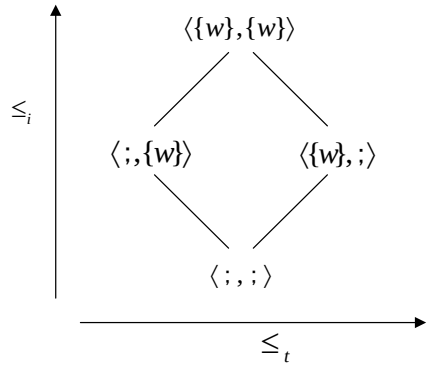


图 2.6 定义在包含一个世界的集合 $\{w\}$ 上的 $\mathrm{B}_{\{w\}}$

下面分别给出真值序关系和信息序关系上的交、并操作，其中 ˆ 、_ 与 ⊗、⊕ 分别对应于真值序关系与信息序关系。

定理 1[17] 设 $\mathrm{B}_W=\langle \mathrm{P}(W)\times\mathrm{P}(W),\leq_t,\leq_i,\neg\rangle$，对任意的 $\langle U_1,V_1\rangle,\langle U_2,V_2\rangle\in\mathrm{P}(W)\times\mathrm{P}(W)$，满足以下条件：

$$\langle U_1,V_1\rangle \hat{} \langle U_2,V_2\rangle = \langle U_1 \setminus U_2, V_1 \cup V_1\rangle$$
$$\langle U_1,V_1\rangle \_ \langle U_2,V_2\rangle = \langle U_1 \cup U_2, V_1 \setminus V_1\rangle$$
$$\langle U_1,V_1\rangle \otimes \langle U_2,V_2\rangle = \langle U_1 \setminus U_2, V_1 \setminus V_1\rangle$$
$$\langle U_1,V_1\rangle \oplus \langle U_2,V_2\rangle = \langle U_1 \cup U_2, V_1 \cup V_1\rangle$$

2.2.2 多值 Kripke 结构与 CTL 逻辑

Kripke 结构是由 Saul Kripke 等人[48]针对模型检测提出的一种用于描述系统行为的迁移系统，其基本结构包含一组状态的集合，原子命题集合，状态之间的迁移关系以及标签函数。传统的 Kripke 结构定义在布尔逻辑上，而 χ Kripke 结构是定义在多值逻辑上，其定义如下：

定义 2.9[49] 一个 **χ Kripke 结构**是一个七元组：$M=\langle S,AP,\rightarrow,S_0,\mathrm{B}_W,\mathrm{R},\mathrm{L}\rangle$，其中：

1) S 是一个有限状态集；
2) AP 是一个原子命题集；
3) $\rightarrow\subseteq S\times S$ 是一个迁移集；
4) $S_0\subseteq S$ 是初始状态集；
5) B_W 是一个世界双格；

6) R 是转换函数， $S\times S\to \mathrm{B}_W$ 上的映像；

7) $\mathrm{L}: S\times AP\to \mathrm{B}_W$ 为标签函数。

例如，图 2.7 所示的就是定义在四值逻辑 $\mathrm{B}_{\{w\}}$ 上的 χ Kripke 结构 M 。对于四值模型 M ，原子命题集合 AP 为 $\{p,q\}$，状态集合 S 为 $\{s_0,s_1,s_2\}$，初始状态为 s_0 。根据定义，原子命题在某个状态上的逻辑值以及迁移关系的逻辑值来自世界双格 $\mathrm{B}_{\{w\}}$ ，如 $\mathrm{L}(p)(s_0)=\langle ;,;\rangle$ ，$\mathrm{R}(s_0,s_1)=\langle\{w\},;\rangle$ 。

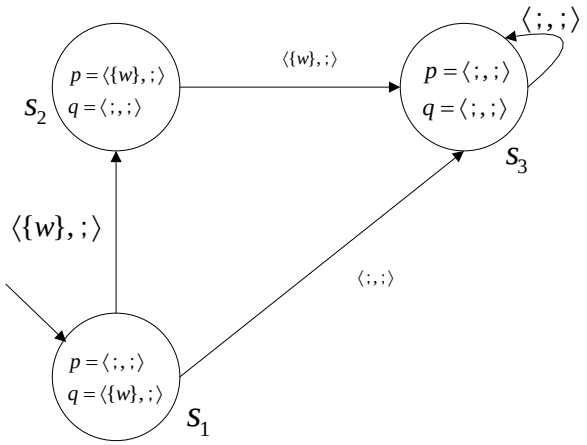


图 2.7 四值 χ Kripke 结构 M

CTL 是由 Emerson 和 Clark 等人[44]针对有限状态迁移系统提出的一种分支时间逻辑。在 CTL 中，系统状态变化的可能性被描述成树状结构，即从任意一个系统状态出发，存在多个可能的后续状态。

定义 2.10[44] 计算树逻辑(Computation Tree Logic，简称 CTL)公式语法由下列规则进行归纳：

$$\varphi ::= \mathrm{true}\,|\,\mathrm{false}\,|\,\neg\varphi\,|\,\varphi\ \hat{}\ \varphi'\,|\,\varphi\_\ \varphi'\,|\,(\varphi\to\varphi')\,|\,\mathrm{AF}\varphi\,|\,\mathrm{EF}\varphi\,|\,\mathrm{EG}\varphi\,|\,\mathrm{AG}\varphi\,|\,\mathrm{A}[\varphi\,\mathrm{U}\,\varphi']$$
$$|\,\mathrm{E}[\varphi\,\mathrm{U}\,\varphi']\,|\,\mathrm{AX}\varphi\,|\,\mathrm{EX}\varphi$$

其中，

1) true， false：逻辑常量，分别表示“真”，“假”；

2) ^ ，_ ，¬， →：基本逻辑连接词，分别表示“且”，“或”，“非”，“蕴含”；

3) A：表示“所有路径”；

4) E：表示“存在一条路径”；

5) F：表示“将来某个状态”；

6) G：表示“所有状态”；

7) U：表示“直到”；

8) X：表示“下一状态”。

CTL 中每个时态连接词都是一对符号：第一个符号均为 A 或 E，其中 A 表示“所有路径”，E 表示“存在一条路径”；第二个符号为 X、F、G 或 U，分别表示“下一状态”，“未来某个状态”，“所有状态”和“直到某个状态”。前者和后者必须成对出现，比如 AF、EG。

接下来，我们给出 CTL 公式在 χ Kripke 结构上的语义说明。

定义 2.11[49] 设 $M=(S,AP,S_0,\rightarrow,B_W,\mathrm{R},\mathrm{L})$ 是一个 χ Kripke 结构，$s\in S$，CTL 公式 φ 的**语义** $\|\varphi\|$ 是一个映射 $S\rightarrow B_W$，根据公式的结构，归纳定义如下：

$$\|\mathrm{true}\|^M=\lambda s.\langle W,;\rangle$$
$$\|\mathrm{false}\|^M=\lambda s.\langle ;,W\rangle$$
$$\|\neg\varphi\|^M=\lambda s.\neg\|\varphi\|^M(s)$$
$$\|\varphi\_\varphi'\|^M=\lambda s.\|\varphi\|^M(s)\_\|\varphi'\|^M(s)$$
$$\|\varphi\hat{}\varphi'\|^M=\lambda s.\|\varphi\|^M(s)\hat{}\|\varphi'\|^M(s)$$
$$\|\mathrm{EX}\varphi\|^M=\mathrm{pre}[\mathrm{R}](\|\varphi\|^M)$$
$$\|\mathrm{AX}\varphi\|^M=\tilde{\mathrm{pre}}[\mathrm{R}](\|\varphi\|^M)$$
$$\|\mathrm{AF}\varphi\|^M=\mu Q.\|\varphi\|^M\_(\tilde{\mathrm{pre}}[\mathrm{R}](Q))$$
$$\|\mathrm{EF}\varphi\|^M=\mu Q.\|\varphi\|^M\_(\mathrm{pre}[\mathrm{R}](Q))$$
$$\|\mathrm{AG}\varphi\|^M=\nu Q.\|\varphi\|^M\_(\tilde{\mathrm{pre}}[\mathrm{R}](Q))$$
$$\|\mathrm{EG}\varphi\|^M=\nu Q.\|\varphi\|^M\_(\mathrm{pre}[\mathrm{R}](Q))$$
$$\|\mathrm{A}[\varphi\mathrm{U}\varphi']\|^M=\mu Q.\|\varphi'\|^M\_(\|\varphi\|^M\hat{}(\tilde{\mathrm{pre}}[\mathrm{R}](Q))$$
$$\|\mathrm{E}[\varphi\mathrm{U}\varphi']\|^M=\mu Q.\|\varphi'\|^M\_(\|\varphi\|^M\hat{}(\mathrm{pre}[\mathrm{R}](Q))$$

其中，符号 μ 和 ν 分别为最小不动点和最大不动点算子。对于迁移关系 $\mathrm{R}:S\times S\rightarrow \mathrm{B}_W$，状态集 $Q:S\rightarrow \mathrm{B}_W$ 相对于 R 的**前像**(pre-image)是一个映射 $\mathrm{pre}[\mathrm{R}]:[S\rightarrow \mathrm{B}_W]\rightarrow[S\rightarrow \mathrm{B}_W]$，定义为：

$$\mathrm{pre}[\mathrm{R}](Q)\triangleq\lambda s.\bigvee_{s'\in S}(\mathrm{R}(s,s')\hat{}Q(s'))。$$

根据定义可以看出，$\mathrm{pre}[\mathrm{R}](Q)$ 是通过 R 可以到达 Q 的状态集。前像的对偶操作 $\tilde{\mathrm{pre}}$ 定义为：

$$\tilde{\mathrm{pre}}[\mathrm{R}](Q)\triangleq\neg\mathrm{pre}[\mathrm{R}](Q)。$$

也就是说，$\tilde{\mathrm{pre}}[\mathrm{R}](Q)$ 是通过 R 必然到达 Q 的状态集，即这些状态的所有后继状态是 Q 的子集。

2.3 本章小结

本章我们首先介绍了软件产品线的两种建模方式：一种是基于特征建模，属于软件产品线的静态建模(static modeling)，另一种是基于系统行为建模，属于软件产品线的动态建模(dynamic modeling)。特征图是基于特征建模的代表性方法之一，我们介绍了特征以及特征图的定义，并给出了基于特征的产品的定义。在软件产品线的行为建模方面，我们介绍了迁移系统和特征迁移系统，以及在特征迁移系统中产品的行为模型的定义。另一方面，我们也介绍了多值模型检

测的相关知识。首先介绍了格、德摩根代数的定义，从而引出双格和世界双格的概念。其次介绍了 χ Kripke 结构以及 CTL 时序逻辑，并给出了 CTL 公式在 χ Kripke 结构上的语义。

第三章 软件产品线的不完备建模

在第二章中，我们介绍了将软件产品线行为与特征相联系的 FTS 建模方法。然而，FTS 中存在的局限性使之更适用于在软件产品线的开发后期。在软件产品线的开发初期，由于对特征的设计不确定，系统中不可避免包含不确定信息，多值逻辑模型适用于描述系统中存在的不确定信息。基于此，我们选择世界双格描述特征设计中的不确定信息，提出基于双格的特征迁移系统以描述含有不确定信息的产品线系统行为。本章首先介绍基于双格的特征迁移系统，然后介绍软件产品线的行为模型在特定产品上的投影从而得到产品的不完备模型，最后介绍通过定义精化关系得到产品的具体模型。

3.1 基于双格的特征迁移系统

在传统的 FTS 中，软件产品线中的行为被描述成带有动作(action)的迁移，且每条迁移用标记(label)的形式将特征与系统行为联系在一起。例如 $s \xrightarrow{a/f} s'$，表示迁移从状态 s 出发，做 a 这个动作，到达状态 s'，并且该迁移是特征 f 要求的，对于其他特征，则定义为与该迁移无关。然而，传统的 FTS 采用这种标记方式无法描述软件产品线的设计初期存在的不确定信息。例如，在饮料机产品线的设计初期，特征 *Tea* 与系统行为 *sugar*（用户选择加糖）之间的关系不明确，即特征 *Tea*（茶）需要还是排斥系统行为 *sugar* 在设计初期并不确定，因此，无论是否采用特征 *Tea* 标记系统行为 *sugar*，都不能准确描述两者之间的不确定关系。此外，软件产品线中存在某两个或两个以上特征对某个系统行为关系不一致(inconsistent)的情况，依旧以饮料机产品线为例，设特征 *Ring*（响铃提醒）需要系统行为 *ring_a_tone*（响铃），而特征 *Display*（显示提醒）排斥系统行为 *ring_a_tone*，则在传统的 FTS 中，系统行为 *ring_a_tone* 采用特征 *Ring* 标记，于是特征 *Display* 被描述为与系统行为 *ring_a_tone* 无关，然而这并没有准确描述特征对系统行为的排斥。传统的 FTS 的局限性来自于 FTS 是布尔模型，在布尔模型中所有的信息都是确定的，因此不适用于描述包含不确定信息的系统。

为此，我们采用世界双格中的逻辑值描述软件产品线中特征设计的不确定情况。考虑到在软件产品线的开发初期，特征与系统行为之间的关系可以分为三种：需要(require)、排斥(forbid)、不确定(unknown)，我们令世界双格中的 W 为软件产品线的特征集合，即每一个世界看作是一个特征；逻辑值 $\langle U,V\rangle$ 中 U 和 V 分表代表依赖和排斥该迁移的特征集合，则 $W\setminus(U\cup V)$ 代表与该迁移的关系暂时不确定的特征集合。例如，在饮料机产品线中，若迁移 $s_1 \xrightarrow{sugar} s_2$ 有逻辑值 $\langle\{b\},;\rangle$，则表示该迁移所描述的系统行为 *sugar* 被特征 *Beverage* 需要，而其他特征如特征 *Tea*，与该迁移的关系暂时不确定；若迁移 $s_8 \xrightarrow{ring\_a\_tone} s_9$ 有逻辑值 $\langle\{r\},\{d\}\rangle$，则表示该迁移所描述的系统行为 *ring_a_tone* 被特征 *Ring* 需要，被特征 *Display* 排斥。我们假设任意一个特征与系

统行为的关系是一致(consistent)的，即任何迁移的逻辑值$\langle U,V\rangle$满足$U \setminus V = ;$。基于此，我们提出基于双格的特征迁移系统(Bilattice-based Featured Transition System，简称 BFTS)，其定义如下：

定义 3.1 一个**基于双格的特征迁移系统(Bilattice-based Featured Transition System，简称 BFTS)**是一个六元组：$M = (S, Act, s_0, \rightarrow, \mathrm{B}_W, \mathrm{R})$，其中

1) S 是一个有穷状态集；

2) Act 是一个有穷行为集；

3) $s_0 \in S$，是唯一的一个初始状态；

4) $\rightarrow \subseteq S \times Act \times S$，是一个有穷迁移集；

5) $\mathrm{B}_W = \langle P(W) \times P(W), \leq_t, \leq_i, \neg \rangle$是一个世界双格；

6) R 是转换函数，$S \times Act \times S \rightarrow \mathrm{B}_W$上的映像。

对于M，若$(s,a,s') \in \rightarrow$，则表示系统存在迁移从状态s，通过动作a，到达状态s'，我们亦可写作$s \xrightarrow{a} s'$，并把s'称作s的后继状态。设W为特征集合，若$\mathrm{R}(s,a,s') = \langle U,V\rangle$，则表示迁移$(s,a,s')$的逻辑值为$\langle U,V\rangle$，即该迁移被$U$中的特征需要，被$V$中的特征反对，并且$W \setminus (U \lceil V)$与该迁移的关系在行为建模过程中尚未确定。特别地，若$\mathrm{R}(s,a,s') = \langle W, ;\rangle$，表示迁移$(s,a,s')$被所有的特征需要；若$(s,a,s') \notin \rightarrow$，则表示系统不存在一条迁移从状态$s$发出$a$这个动作后到达下一个状态$s'$，因此迁移$(s,a,s'$在该产品线中被所有特征排斥，即$\mathrm{R}(s,a,s') = \langle ;, W\rangle$。

以例 1 中的饮料机产品线为例，我们采用 BFTS 描述其所有的系统行为，如图 3.1 所示，其中s_0为系统的初始状态，系统中每个迁移的逻辑值已在迁移上标明。以“投币”行为为例，该行为在行为模型中被描述成迁移$(s_0, coin, s_1)$，由于它被特征v需要并且没有产品线所包含的任何特征排斥，于是在 BFTS 中，迁移$(s_0, coin, s_1)$的逻辑值为$\langle \{v\}, ;\rangle$，即$\mathrm{R}(s_0, coin, s_1) = \langle \{v\}, ;\rangle$。同理，特征$c$要求系统包含行为$coffee$（提供咖啡）、$pour\_coffee$（倒咖啡）以及$pour_sugar$（加糖），且这些行为没有被其他特征要求或排斥，因此它们对应的迁移在 BFTS 中的逻辑值均为$\langle \{c\}, ;\rangle$。

此外，迁移的逻辑值采用有序对$\langle U,V\rangle$的形式能直观反映特征之间的互斥关系。互斥关系存在于两个或多个特征之间，若两个特征是互斥的，则表示它们共存于一个产品中。造成特征之间的互斥是由于它们与系统的某些行为的关系是不一致的，即若特征f_1与特征f_2互斥，则一定存在某迁移(s,a,s')，其逻辑值$\langle U,V\rangle$满足$f_1 \subseteq U \wedge f_2 \subseteq V$或$f_1 \subseteq V \wedge f_2 \subseteq U$。例如，饮料机产品线中的特征$r$与特征$d$互斥（如图 2.1 所示），在对应的 BFTS 中，特征r与特征d的互斥体现在与迁移$(s_8, display_done, s_9)$和$(s_8, ring_a_tone, s_9)$的关系不一致，即$\mathrm{R}(s_8, display_done, s_9) = \langle \{d\}, \{r\}\rangle$，$\mathrm{R}(s_8, ring\_a\_tone, s_9) = \langle \{r\}, \{d\}\rangle$。

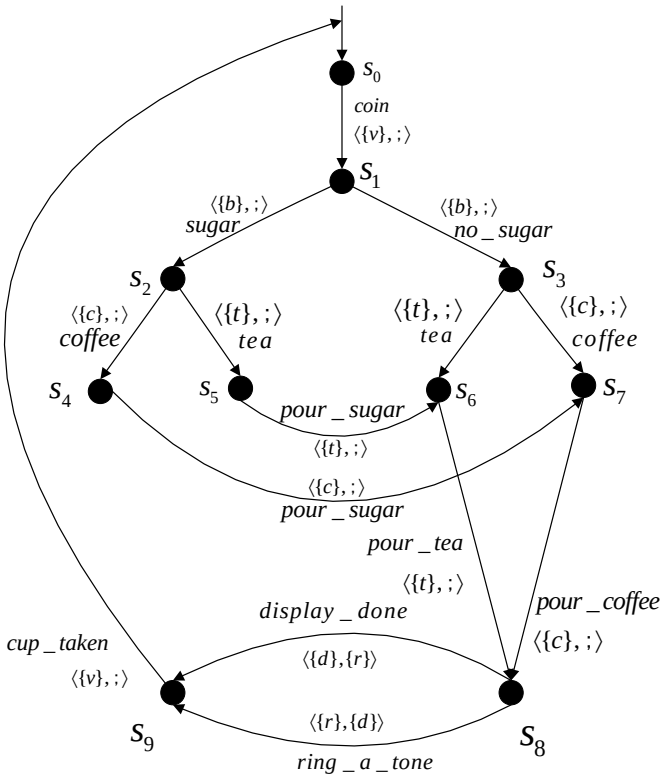


图 3.1 饮料机产品线的 BFTS

3.2 产品的不完备模型

本节我们主要介绍 BFTS 在特征集合上的投影。在 2.1.1 节中，我们介绍了在特征建模中产品的定义，即原产品线的特征集合中符合特征约束关系的子集。于是，根据产品中的特征，通过保留产品线的行为模型中这些特征需要的迁移即可得到产品的行为模型。传统的 FTS 建模中把这一过程称为投影。若已知产品线的 FTS M 和特定产品 P，通过投影就能得到产品的行为模型 TS $M_{|P}$。然而，在软件产品线的 BFTS 建模中，由于系统中可能包含不确定信息，即存在特征与系统行为之间关系不确定的情况，而这些特征可能存在于特征产品 P 中。若采用传统 FTS 中定义的投影方法,则得到的产品模型可能并没有包含 P 中最终所有的行为。这是因为当 P 中存在某个特征 f，且 f 与某些行为的关系在软件产品线的开发后期定义为“需要”，但在软件产品线的开发初期定义为“不确定”时，通过传统 FTS 中所定义的投影方法，这些行为将被去除，即产品模型中不包含这些行为，使得得到的产品模型与实际不符。例如，设特征 *Text* 与系统行为 *Internet*（上网）关系在软件产品线开发的初期不确定，而在后期特征 *Text* 需要系统行为 *Internet*，产品 P 包含特征 *Text* 且 P 中其他特征与行为 *Internet* 关系都不确定，则通过传统 FTS 定义的投影， P 的行为模型中不包含系统行为 *Internet*。

为此，我们定义软件产品线的不完备建模中投影的概念，使得软件产品线的 BFTS 模型通过在产品 P 上的投影得到的模型不仅包含了特征需要的迁移，也包含了与所选特征关系不确定

的迁移，即仅删去了产品线模型中 P 中特征排斥的迁移。由于投影中包含 P 中的所有系统行为以及可能出现的行为，因此我们又称之为产品的不完备模型(partial model)。假设产品线中每一个特征与系统行为之间不存在不一致情况，即产品线的 BFTS 中任一迁移的逻辑值 $\langle U,V\rangle$ 满足 $U \setminus V = ;$，我们提出的投影的定义如下：

定义 3.2 设一个产品线 F 的行为模型 BFTS $M = (S, Act, s_0, \rightarrow, \mathrm{B}_W, \mathrm{R})$，$P$ 是 F 的一个产品。则我们称 BFTS $M' = (S', Act', s_0, \rightarrow', \mathrm{B}_P, \mathrm{R}')$ 是 M 在 P 上的**投影**，记为 $M_{|P}$，其中：

1) $\rightarrow' = \{(s,a,s') \mid (s,a,s') \in \rightarrow \wedge \exists f \in \pi_t(\mathrm{R}(s,a,s')) \cdot f \in P\}$；

2) $S' = \{s \mid \exists s' \cdot \exists a \cdot ((s,a,s') \in \rightarrow' \_ (s',a,s) \in \rightarrow')\}$；

3) $Act' = \{a \mid \exists s \cdot \exists s' \cdot (s,a,s') \in \rightarrow'\}$；

4) $$\mathrm{R}'(s,a,s') = \begin{cases} \langle P, ; \rangle, & \text{当} \pi_t(\mathrm{R}(s,a,s')) \setminus P \neq ;, \\ \langle ; , P \rangle, & \text{当} \pi_f(\mathrm{R}(s,a,s')) \setminus P \neq ;, \\ \langle ; , ; \rangle, & \text{其他.} \end{cases}$$

对于 $M_{|P}$，通常情况下根据逻辑值的不同，该模型包含三种迁移：1)被所选特征需要的迁移——逻辑值为 $\langle P, ; \rangle$；2)被所选特征排斥的迁移——逻辑值为 $\langle ; , P \rangle$；3)与所选特征关系不确定的迁移——逻辑值为 $\langle ; , ; \rangle$。因此，通常情况下 $M_{|P}$ 是一个三值模型。特别地，若特征选择 P 中所有特征在软件产品线的模型中与迁移之间的关系都是确定的，即针对产品线的行为模型中每一个迁移，P 中所有特征明确“需要”或者“排斥”，则得到的投影 $M_{|P}$ 只包含逻辑值为 $\langle P, ; \rangle$ 或 $\langle ; , P \rangle$ 的迁移，那么 $M_{|P}$ 是一个二值模型。

以例 1 中的饮料机产品线为例，设产品 $P = \{v, b, f, c, r\}$，我们按照投影的定义，根据饮料机产品线的行为模型 BFTS M 中迁移 $(s_0, coin, s_1)$ 、迁移 (s_2, tea, s_5) 、迁移 $(s_8, display\_done, s_9)$ 的逻辑值来分别计算它们在投影 $M_{|P}$ 中的逻辑值 $\mathrm{R}'(s_0, coin, s_1)$ 、$\mathrm{R}'(s_2, tea, s_5)$ 、$\mathrm{R}'(s_8, display\_done, s_9)$。其计算过程如下：

(1) 计算 $\mathrm{R}'(s_0, coin, s_1)$ 的值；

因为有 $\mathrm{R}(s_0, coin, s_1) = \langle \{v\}, ; \rangle$，且 $\pi_t(\mathrm{R}(s_0, coin, s_1)) \wedge P \neq ;$，

则 $\mathrm{R}'(s_0, coin, s_1) = \langle \{v, b, f, c, r\}, ; \rangle$，迁移 $(s_0, coin, s_1)$ 被产品 P 的特征需要，一定存在于 P 的行为模型中。

(2) 计算 $\mathrm{R}'(s_2, tea, s_5)$ 的值；

因为有 $R(s_2, tea, s_5) = \langle \{t\}, ; \rangle$，且 $\pi_t(\mathrm{R}(s_2, tea, s_5)) \wedge P = ;$， $\pi_f(\mathrm{R}(s_2, tea, s_5)) \wedge P = ;$，

则 $\mathrm{R}'(s_2, tea, s_5) = \langle ; , ; \rangle$，迁移 (s_2, tea, s_5) 与产品 P 的特征关系不确定，暂不确定该迁移是否存在于 P 的行为模型中。

(3) 计算 $\mathrm{R}'(s_8, display\_done, s_9)$ 的值；

因为有 $\mathrm{R}(s_8, display\_done, s_9) = \langle \{d\}, \{r\} \rangle$，且 $\pi_f(\mathrm{R}(s_1, \mathrm{coffee}, s_2)) \wedge P \neq ;$，

则 $\mathrm{R}'(s_1, coffee, s_2) = \langle ;, \{v, b, f, c, r\} \rangle$，迁移 $(s_8, display\_done, s_9)$ 被产品 P 的特征排斥，一定不存在于 P 的行为模型中。

同理，我们可计算出饮料机产品线 BFTS M 中所有的迁移在投影 $M_{|P}$ 中的逻辑值，得到 $M_{|P}$ 如图 3.2 所示。由于 M 中包含不确定信息，$M_{|P}$ 是一个三值 BFTS，其中虚线表示逻辑值为 $\langle ;, ; \rangle$ 的迁移，实线表示逻辑值 $\langle P, ; \rangle$ 的迁移。

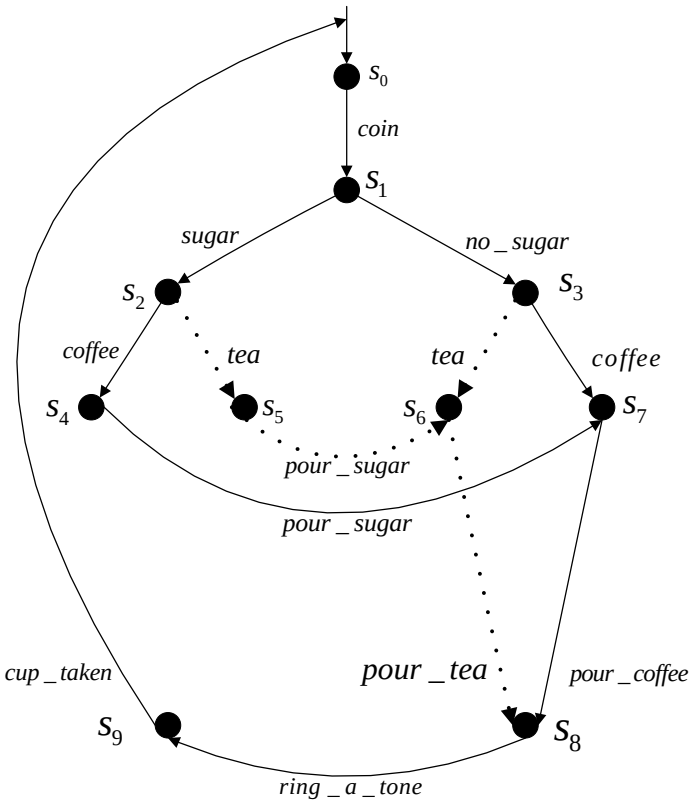


图 3.2 产品线的 BFTS M 在产品 $P = \{v, b, f, c, r\}$ 上的投影 $M_{|P}$

3.3 产品的具体模型

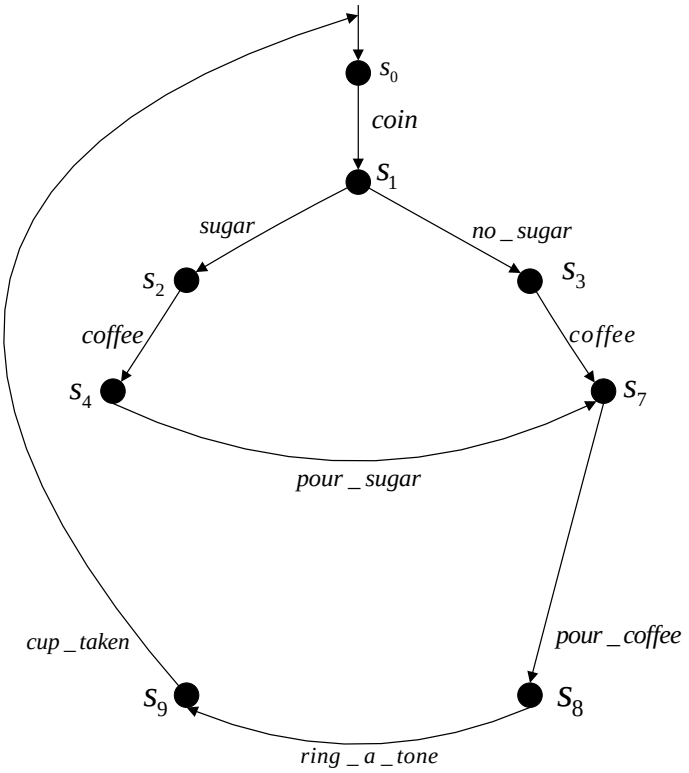
本节中，我们主要介绍产品的最终模型。在 3.2 节中我们介绍了软件产品线的不完备模型在已选特征上的投影，即产品的不完备模型(partial model)。对于产品的最终模型，其所有信息是明确的，即对于投影中所有不确定的迁移，他们在产品的最终模型中与产品特征的关系是明确的。为此，我们采用定义精化关系以得到产品的最终模型，即对于产品的不完备模型中的迁移 (s, a, s')，若其逻辑值为 $\langle P, ; \rangle$，则产品的最终模型包含迁移 (s, a, s')；若其逻辑值为 $\langle ;, P \rangle$，则产品的最终模型不包含迁移 (s, a, s')；若其逻辑值为 $\langle ;, ; \rangle$，则产品的最终模型包含或不包含迁移 (s, a, s')。由于产品的最终模型中去掉了不完备模型中的所有不确定迁移，我们又称之为产品的具体模型(concrete model)，其定义如下：

定义 3.3[29] 设 $M_{|P}=(S_1, Act_1, \rightarrow_1, s_{0_1}, \mathrm{B}_P, R_1)$ 为 BFTS M 在 P 上的投影。产品 P 的**具体模型**是一个二值模型 BFTS $M(P)=(S_2, Act_2, \rightarrow_2, s_{0_2}, \mathrm{B}_P, R_2)$。关系 $H \subseteq S_2 \times S_1$ 是 $M_{|P}$ 与 $M(P)$ 之间的**精化关系**，当且仅当 $(s_{0_2}, s_{0_1}) \in H$，其中，如果 $(s_2, s_1) \in H$，则以下条件成立：

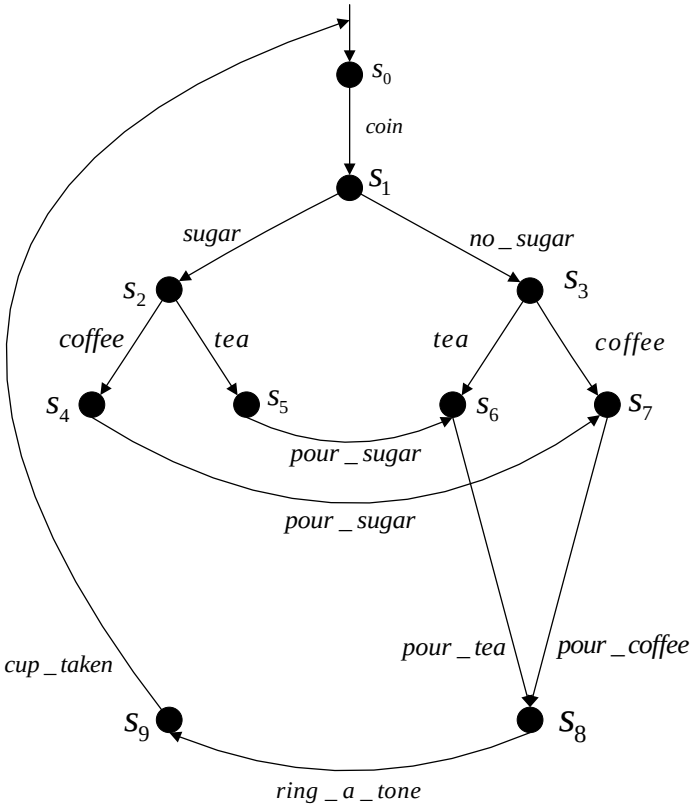
1) $\forall s_1^{'} \in S_1 \cdot \forall a \in Act_1 \cdot P \subseteq \pi_t(R_1(s_1, a, s_1^{'})) \Rightarrow \exists s_2^{'} \in S_2 \cdot P \subseteq \pi_t(R_2(s_2, a, s_2^{'})) \wedge (s_2^{'}, s_1^{'}) \in H$；

2) $\forall s_2^{'} \in S_2 \cdot \forall a \in Act_2 \cdot P \not\subseteq \pi_f(R_2(s_2, a, s_2^{'})) \Rightarrow \exists s_1^{'} \in S_1 \cdot P \not\subseteq \pi_f(R_1(s_1, a, s_1^{'})) \wedge (s_2^{'}, s_1^{'}) \in H$。

与 $M_{|P}$ 满足精化关系的 $M(P)$ 的个数可能不止一个。若 $M_{|P}$ 中逻辑值为$\langle ;, ; \rangle$ 的迁移个数为 n，则 $M(P)$ 的个数最多可达 2^n。因此，通过 BFTS 的建模方法，使得软件产品线初期的不确定信息得以描述，从而考虑到更多开发过程中的不确定因素。以例 1 中的饮料机产品线为例，已知产品 $P=\{v, b, f, c, r\}$ 和产品线的行为模型 BFTS M（如图 3.1 所示），我们首先得到 M 在 P 上的投影 $M_{|P}$（如图 3.2 所示），通过精化关系，我们最终得到的 P 的两个行为模型 $M(P)_1, M(P)_2$。迁移 (s_2, tea, s_5)、$(s_5, pour\_sugar, s_6)$、(s_3, tea, s_6) 和 $(s_6, pour\_tea, s_8)$，它们在 $M_{|P}$ 中的逻辑值为$\langle ;, ; \rangle$。然而在这两个行为模型中，它们的逻辑值是不同的：(1)在 $M(P)_1$ 中，这些迁移的逻辑值为$\langle ;, P \rangle$，即它们最终确定为特征不需要的（如图 3.3(a)所示）；(2)在 $M(P)_2$ 中，这些迁移的逻辑值为$\langle P, ; \rangle$，即它们最终确定为特征需要的（如图 3.3(b)所示）。若采用传统的 FTS 对饮料机产品线建模，根据其投影定义，则生成的产品模型只为 $M(P)_1$。



(a)



(b)

图 3.3 $P=\{v,b,f,c,r\}$ 的具体模型：(a) $M(P)_1$；(b) $M(P)_2$

至此，我们实现通过产品线的不完备模型得到产品的具体模型。图 3.4 总结了通过软件产品线的不完备模型得到特定产品的具体模型的流程：首先将软件产品线的行为模型 M 投影在特定产品 P 上从而得到产品的不完备模型 $M_{|P}$，其次通过精化关系得到产品的具体模型 $M(P)$。由于 M 中允许包含不确定信息，通过投影得到的 $M_{|P}$ 中可能存在不确定信息，而通过精化关系得到的 $M(P)$ 中不包含任何不确定信息。

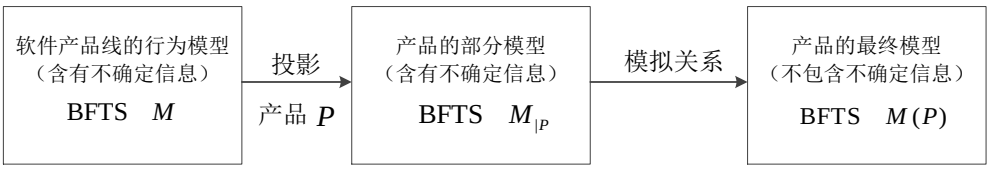


图 3.4 产品 P 的具体模型生成流程

3.4 本章小结

本章首先介绍了针对软件产品线的不完备建模所提出的多值迁移系统——BFTS。不同于传统的 FTS，在 BFTS 中，每一个迁移都有一个有序对 $\langle U,V\rangle$ 形式的逻辑值。通过逻辑值描述特征与迁移之间的联系，从而支持特征与迁移之间存在的不确定关系的情况。相比于传统的 FTS，

BFTS 更好地支持了软件产品线设计初期所出现的不确定情况。然后介绍得到产品模型的方法，由于产品线的 BFTS 中可能包含不确定信息，通过选取特定产品中特征需要的迁移不能保证得到正确的产品模型，于是，我们首先通过定义产品线的 BFTS 在特定产品上的投影得到产品的不完备模型；接着定义精化关系得到产品的具体模型。

第四章 软件产品线的多值模型检测

本章主要介绍软件产品线的多值模型检测。首先，基于本文提出的多值模型 BFTS，介绍描述系统属性的 ACTL 时序逻辑，并给出定义在 BFTS 上的语义，从而实现基于 BFTS 的多值模型检测；其次，提出从 BFTS 到 χ Kripke 结构上的等价转换方法，实现将基于 BFTS 的多值模型检测问题转化为基于 χ Kripke 结构的多值模型检测问题；最后提出多值 BFTS 相对于单个特征的模型分解方法，实现将基于 BFTS 的多值模型检测问题分解为多个基于三值 BFTS 的多值模型检测问题。

4.1 基于 BFTS 的多值模型检测

在模型检测中，时序逻辑公式用以描述系统属性。常见的时序逻辑有计算树逻辑(Computation Tree Logic，简称 CTL)[44]和线性时序逻辑(Linear-time Temporal Logic，简称 LTL)[50]。当系统属性采用 CTL 和 LTL 描述时，系统行为则采用基于状态的模型(state-based model)描述，如 Kripke 结构等。这些模型通过定义状态上的原子命题描述系统的所有状态，例如电梯系统的 Kripke 结构中，初始状态 s_0 上的原子命题 *button* 的值为 *false* 表示在初始状态电梯按钮没有被人按。除了基于状态的建模方式，系统的行为还可以采用基于动作的模型(action-based model)描述，如 LTS 等，这些模型通过定义状态的迁移和标记迁移上的动作来描述系统的动作和状态的变化，例如电梯系统的 LTS 模型中，迁移 $s_0 \xrightarrow{button} s_1$ 表示通过用户按下按钮 *button*，系统从初始状态 s_0 到达状态 s_1。对于这些采用基于动作的模型描述的系统，我们需要采用基于动作的时序逻辑描述其时序属性。为此，De Nicola 等人[42]针对 LTS 模型提出一种基于动作的 CTL 逻辑——ACTL，其语法如下：

定义 4.1 ACTL 公式由下列规则进行归纳：

$$\varphi ::= \text{true} \mid \text{false} \mid \neg\varphi \mid \varphi \hat{} \varphi' \mid \varphi \_ \varphi' \mid (\varphi \to \varphi') \mid \text{AF}\varphi \mid \text{EF}\varphi \mid \text{EG}\varphi \mid \text{AG}\varphi \mid \text{A}[\varphi \text{U} \varphi'] \mid \text{E}[\varphi \text{U} \varphi'] \mid ([a]\varphi) \mid \langle a \rangle \varphi$$

其中，

1) true，false：逻辑常量，分别表示“真”，“假”；
2) $\hat{}$，_，$\neg$，$\to$：基本逻辑连接词，分别表示“且”，“或”，“非”，“蕴含”；
3) A：表示“所有路径”；
4) E：表示“存在一条路径”；
5) F：表示“将来某个状态”；
6) G：表示“所有状态”；
7) U：表示“直到”；

8) a：表示“动作(action)”；

9) $\langle a\rangle$：表示“存在某条迁移做 a 动作到达下一状态”；

10) $[a]$：表示“所有做 a 这个动作的迁移到达下一状态”。

除了$\langle\rangle$和[]，每个 ACTL 时态连接词都是一对符号。符号对中的第一个是 A 或 E，符号对中的第二个符号是 F，G 或 U。以例 1 中的饮料机产品线为例，ACTL 公式 $AG[sugar]AF\,pour\_sugar$ 描述属性“若顾客选择饮料加糖，则饮料机将来会在饮料里加糖”。

本文选择 ACTL 描述软件产品线的系统属性以实现基于 BFTS 的多值模型检测。由于传统的 ACTL 语义是定义在二值上的，而 BFTS 是多值模型，传统的 ACTL 语义并不适用。于是，我们将 ACTL 的语义扩展到 BFTS 上，其定义如下：

定义 4.2 设 $M=(S,Act,s_0,\rightarrow,\mathrm{B}_W,\mathrm{R})$ 是一个 BFTS，$s\in S$，ACTL 公式 φ 的语义 $\|\varphi\|$ 是一个映射 $S\rightarrow\mathrm{B}_W$，根据公式的结构，归纳定义如下：

$$\|\mathrm{true}\|^M=\lambda s.\langle W,;\rangle$$
$$\|\mathrm{false}\|^M=\lambda s.\langle;,W\rangle$$
$$\|\neg\varphi\|^M=\lambda s.\neg\|\varphi\|^M(s)$$
$$\|\varphi\_\varphi'\|^M=\lambda s.\|\varphi\|^M(s)\_\|\varphi'\|^M(s)$$
$$\|\varphi\hat{}\varphi'\|^M=\lambda s.\|\varphi\|^M(s)\_\|\varphi'\|^M(s)$$
$$\|\langle a\rangle\varphi\|^M=\mathrm{pre}_a[\mathrm{R}](\|\varphi\|^M)$$
$$\|[a]\varphi\|^M=\tilde{\mathrm{pre}}_a[\mathrm{R}](\|\varphi\|^M)$$
$$\|\mathrm{AF}\varphi\|^M=\mu Q.\|\varphi\|^M\_(\hat{\bigcup_{a\in Act}}\tilde{\mathrm{pre}}_a[\mathrm{R}](Q))$$
$$\|\mathrm{EF}\varphi\|^M=\mu Q.\|\varphi\|^M\_(\bigcup_{a\in Act}\mathrm{pre}_a[\mathrm{R}](Q))$$
$$\|\mathrm{AG}\varphi\|^M=\nu Q.\|\varphi\|^M\_(\hat{\bigcup_{a\in Act}}\tilde{\mathrm{pre}}_a[\mathrm{R}](Q))$$
$$\|\mathrm{EG}\varphi\|^M=\nu Q.\|\varphi\|^M\_(\bigcup_{a\in Act}\mathrm{pre}_a[\mathrm{R}](Q))$$
$$\|\mathrm{A}[\varphi\mathrm{U}\varphi']\|^M=\mu Q.\|\varphi'\|^M\_(\|\varphi\|^M\hat{}(\hat{\bigcup_{a\in Act}}\tilde{\mathrm{pre}}_a[\mathrm{R}](Q))$$
$$\|\mathrm{E}[\varphi\mathrm{U}\varphi']\|^M=\mu Q.\|\varphi'\|^M\_(\|\varphi\|^M\hat{}(\hat{\bigcup_{a\in Act}}\mathrm{pre}_a[\mathrm{R}](Q))$$

其中，符号 μ 和 ν 分别为最小不动点和最大不动点算子。对于迁移关系 $\mathrm{R}:S\times Act\times S\rightarrow\mathrm{B}_W$，状态集 $Q:S\rightarrow\mathrm{B}_W$ 相对于 R，a 的**前像**(pre-image)是一个映射 $\mathrm{pre}_a[\mathrm{R}]:[S\rightarrow\mathrm{B}_W]\rightarrow[S\rightarrow\mathrm{B}_W]$，定义为：

$$\mathrm{pre}_a[\mathrm{R}](Q)=\lambda s.\bigcup_{s'\in S}(\mathrm{R}(s,a,s')\hat{}\,Q(s'))。$$

根据定义可以看出，$\mathrm{pre}_a[\mathrm{R}](Q)$ 是通过 R，a 可以到达 Q 的状态集。前像的对偶操作 $\tilde{pre}$ 定义为：

$$\tilde{\text{pre}}_a[\text{R}](Q) = \neg \text{pre}_a[\text{R}](Q)。$$

也就是说，$\tilde{\text{pre}}_a[\text{R}](Q)$ 是通过 R，a 必然到达 Q 的状态集，即这些状态的所有后继状态是 Q 的子集。

以饮料机产品线为例，若想描述属性“在显示‘结束’后，饮料会被拿走”，采用 ACTL 描述这一属性为 $\langle display\_done\rangle\langle cup\_taken\rangle \text{true}$，我们设该公式为 φ。根据饮料机产品线的 BFTS M，我们需要在状态 s_9 上验证公式 φ，即求 $\|\langle display\_done\rangle\langle cup\_taken\rangle \text{true}\|^M(s_9)$ 的计算结果。根据定义 4.2，我们有以下计算过程：

$$\begin{aligned}
&\|\langle display_done\rangle\langle cup_taken\rangle \text{true}\|^M(s_9)\\
&= \text{R}(s_8, ring_a_tone, s_9)\,\hat{}\,\|\langle cup_taken\rangle \text{true}\|^M(s_9)\\
&= \text{R}(s_8, ring_a_tone, s_9)\,\hat{}\,\text{R}(s_9, cup_taken, s_0)\\
&= \langle\{r\},\{d\}\rangle\,\hat{}\,\langle\{v\},;\rangle\\
&= \langle;,\{d\}\rangle
\end{aligned}$$

即公式 φ 在饮料机产品线模型 M，状态 s_9 的验证结果是 $\langle;,\{d\}\rangle$。

根据定义 2.2，我们得知产品的特征集合是软件产品线的特征集合的子集。再由定义 3.2 和定义 3.3，可得软件产品线 F 的行为模型 M 中的迁移集合 $\rightarrow$ 和所有产品的行为模型 $\{M(P_1), M(P_2)...M(P_i)...M(P_m)\}$（$m$ 为 F 的产品个数）中的迁移集合 $\{\rightarrow_1, \rightarrow_2 ... \rightarrow_i ...\}$ 满足关系：$\rightarrow = \bigsqcup_{i=1}^{i=m} \rightarrow_i$，即软件产品线中的迁移是所有产品中的迁移的总和。

由此我们得出结论：在不考虑产品中出现的不一致情况的情况下，即系统模型中任意迁移的逻辑值 $\langle X,Y\rangle$ 满足 $X \setminus Y = ;$，设公式 φ 在产品线模型上的检测的结果为 $\langle U,V\rangle$，若产品 P 满足 $P \setminus U \neq ;$，则公式 φ 在 P 上成立。若 $P \setminus V \neq ;$，则公式 φ 在 P 上不成立。例如，公式 $\langle display\_done\rangle\langle cup\_taken\rangle \text{true}$ 在产品线模型 M 的状态 s_9 验证结果为 $\langle;,\{d\}\rangle$，则表示包含特征 d 的产品不满足属性“在显示‘结束’后，饮料会被拿走”。

4.2 多值模型检测问题的实现

本节主要介绍基于模型转换的多值模型检测，首先介绍从 BFTS 到 χ Kripke 结构的等价转换方法，其次介绍从 ACTL 公式到 CTL 公式的等价转换，从而实现将基于 BFTS 的多值模型检测问题转换为基于 χ Kripke 结构的模型检测问题。

4.2.1 基于模型转换的多值模型检测

在形式化建模中，基于动作的模型(action-based model)和基于状态的模型(state-based model)是两类常用模型。基于状态的模型如 Kripke 结构，它们通过定义状态上的原子命题来描述状态之间的变化；基于动作的模型如 LTS，MTS 等，它们通过标记迁移上的动作来描述引起状态变化的原因。这两类模型之间的等价性在学术界已有相关研究，在二值模型方面，De Nicola 等人

[42]提出了 LTS 与 Kripke 结构之间的等价转换方法；在多值模型方面，三值模型作为一种特殊的多值模型，其相关研究已日趋成熟，其具体形式包括 MTS、KMTS、不完备 Kripke 系统(partial Kripke structure)[51]以及三值 Kripke 结构(3-valued Kripke structure)[52]。Godeforid 等人[51,52]中证明了这些三值模型虽然有不同的结构特征，但是都具有相同的表达能力，并给出了它们之间的等价转换方法。基于此，我们提出从多值模型 BFTS 到 χ Kripke 结构等价转换的方法，其定义如下：

定义 4.3 设 M 为 BFTS $M=(S,Act,s_0,\rightarrow,\mathrm{B}_W,\mathrm{R})$，则 χ Kripke 结构 $M'=(S',\rightarrow',P,s_0',\mathrm{B}_W,S')$，由 M **等价转换**得到，其中：

1) $S'=S\times Act$，
2) $AP=Act$，
3) $S_0'=s_0\times Act$，
4) $\rightarrow'=\{((s,a),(s',a'))\mid(s,a,s')\in\rightarrow\}$，
5) $\forall(s,a),(s',a')\in S':\mathrm{R}'((s,a),(s',a'))=\mathrm{R}(s,a,s')$，
6) $\forall(s,a)\in S':\forall p\in AP:$

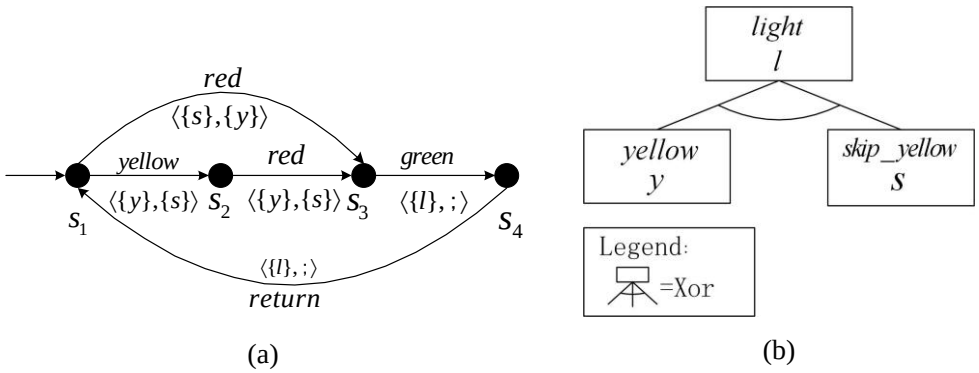
$$L((s,a),p)=\begin{cases}\langle W,;\rangle, & \text{if } p=a,\\ \langle;,W\rangle, & \text{else.}\end{cases}$$

由转换关系可知，M'的状态数最多能达到$|S|\cdot|Act|$，即 M 中的每一个状态在转换过程中最多复制$|Act|$次。因此，当 M 中 Act 的元素个数固定时，M'中的状态（迁移）数与 M 中的状态（迁移）数呈线性关系。特别地，当 M 的特征集合 W 元素个数为 1 时，转换方法看作是从三值 BFTS 到三值 χ Kripke 结构上的转换。

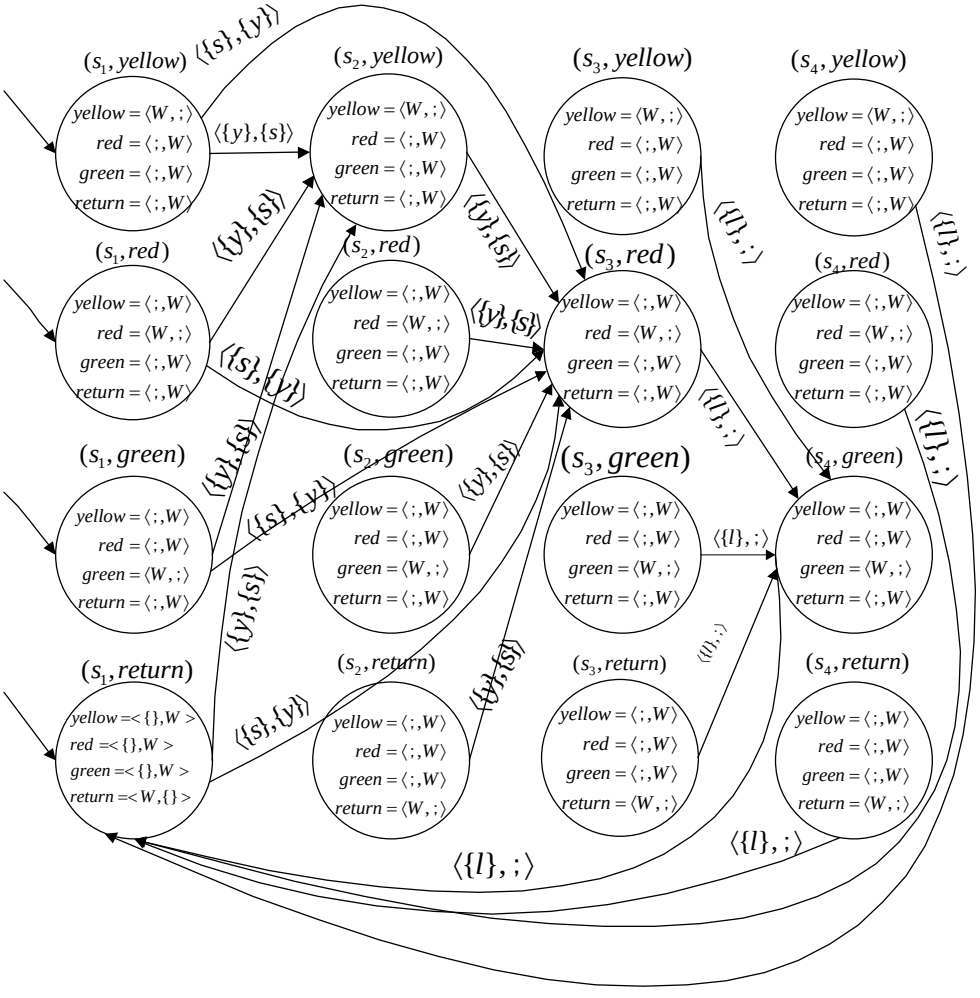
例 2 图 4.1(a)是信号灯软件产品线的特征图 D；其中 *light*（亮灯）为 D 的根特征，并包含了 2 个互斥的子特征：特征 *yellow*（亮黄灯）和 *skip_yellow*（不亮黄灯）。因此，根据特征图语义，信号灯软件产品线上的产品为 $\{l,y\}$ 或 $\{l,s\}$（l、y、s 分别为特征 *light*、*yellow*、*skip_yellow* 的缩写，已在图 4.1(a)中标明）。信号灯软件产品线的 BFTS M 如图 4.1(b)所示，它包含 4 个状态、4 个动作、5 个迁移其中特征 *light* 要求信号灯可以从红灯变成绿灯并且可以回到初始状态，即需要迁移 $(s_3,green,s_4)$ 和 $(s_4,return,s_1)$；特征 *yellow* 要求信号灯在亮红灯之前先亮黄灯，即需要迁移 $(s_1,yellow,s_2)$ 和 (s_2,red,s_3)，反对迁移 (s_1,red,s_3)，而与特征 *yellow* 互斥的特征 *skip_yellow* 要求系统不亮黄灯，即需要迁移 (s_1,red,s_3)，反对迁移 $(s_1,yellow,s_2)$ 和 (s_2,red,s_3)。

我们得到转换后的 χ Kripke 结构 M' 如图 4.1(c)所示，它包含 16 个状态和 20 个迁移。以 M_l 中的状态 s_1、状态 s_2 和迁移 $(s_1,yellow,s_2)$ 为例，状态 s_1 对应 M' 的状态 $(s_1,yellow)$，(s_1,red)，$(s_1,green)$ 和 $(s_1,return)$；状态 s_2 对应 M' 的状态 $(s_2,yellow)$，(s_2,red)，$(s_2,green)$ 和 $(s_2,return)$；

迁移 $(s_1, yellow, s_2)$ 对应 M' 的迁移 $((s_1, yellow),(s_2, yellow))$，$((s_1, red),(s_2, yellow))$，$((s_1, green),(s_2, yellow))$ 和 $((s_1, return),(s_2, yellow))$。



(a) (b)



(c)

图 4.1 信号灯软件产品线建模：(a) 信号灯软件产品线的 BFTS M；(b) 信号灯软件产品线的特征图 D；(c) 等价转换后的信号灯产品线的 χ Kripke 结构 M'

ACTL 是 De Nicola 等人[42]基于 CTL 提出的时序逻辑，它与 CTL 之间可以等价转换。以下为从 ACTL 到 CTL 的等价转换函数：

定义 4.4[42] 转换函数 $ks':\mathrm{ACTL}\to\mathrm{CTL}$ 有以下定义：

$$
\begin{aligned}
ks'(\mathrm{true}) &= \mathrm{true},\\
ks'(\mathrm{false}) &= \mathrm{false},\\
ks'(\neg\varphi) &= \neg ks'(\varphi),\\
ks'(\varphi \wedge \varphi') &= ks'(\varphi) \wedge ks'(\varphi'),\\
ks'(\varphi \vee \varphi') &= ks'(\varphi) \vee ks'(\varphi'),\\
ks'([a]\varphi) &= \mathrm{AX}(a \to ks'(\varphi)),\\
ks'(\langle a\rangle\varphi) &= \mathrm{EX}(a \wedge ks'(\varphi)),\\
ks'(\mathrm{AF}\varphi) &= \mu Q.ks'(\varphi) \vee (\bigwedge_{a\in Act} ks'([a]Q)),\\
ks'(\mathrm{EF}\varphi) &= \mu Q.ks'(\varphi) \vee (\bigvee_{a\in Act} ks'(\langle a\rangle Q)),\\
ks'(\mathrm{AG}\varphi) &= \mu Q.ks'(\varphi) \wedge (\bigwedge_{a\in Act} ks'([a]Q)),\\
ks'(\mathrm{EG}\varphi) &= \mu Q.ks'(\varphi) \wedge (\bigvee_{a\in Act} ks'(\langle a\rangle Q)),\\
ks'(\mathrm{A}(\varphi \mathrm{U} \varphi')) &= \mu Q.ks'(\varphi) \wedge (ks'(\varphi) \wedge \bigwedge_{a\in Act} ks'([a]Q)),\\
ks'(\mathrm{E}(\varphi \mathrm{U} \varphi')) &= \mu Q.ks'(\varphi) \wedge (ks'(\varphi) \wedge \bigwedge_{a\in Act} ks'(\langle a\rangle Q)).
\end{aligned}
$$

下面我们以 ACTL 公式 $\mathrm{AF}\langle ring\_a\_tone\rangle\,\mathrm{true}$ 为例，根据定义 4.4，将其转换成等价的 CTL 公式。计算过程如下：

$$
\begin{aligned}
& ks'(\mathrm{AF}\langle ring\_a\_tone\rangle\,\mathrm{true})\\
&= \mu Q.ks'(\langle ring\_a\_tone\rangle\,\mathrm{true}) \vee (\bigwedge_{a\in Act} ks'([a]Q))\\
&= \mu Q.\mathrm{EX}\, ring\_a\_tone \vee (\bigwedge_{a\in Act} ks'([a]Q))\\
&= \mu Q.\mathrm{EX}\, ring\_a\_tone \vee (\bigwedge_{a\in Act} \mathrm{AX}(a \to Q))\\
&= \mu Q.\mathrm{EX}\, ring\_a\_tone \vee (\mathrm{AX}\, Q)\\
&= \mathrm{AFEX}\, ring\_a\_tone
\end{aligned}
$$

因此，ACTL 公式 $\mathrm{AF}\langle ring\_a\_tone\rangle\,\mathrm{true}$ 与 CTL 公式 $\mathrm{AFEX}\, ring\_a\_tone$ 等价。

下面证明，BFTS M 转换为 χ Kripke 结构 M'，任意 ACTL 公式 φ 在 M 上的检测结果与和 φ 等价的公式 φ' 在 M' 上的检测结果相同。

定理 4.1 设 M 为 BFTS $M=(S,Act,s_0,\to,\mathrm{B}_W,\mathrm{R})$，$M$ 通过等价转换得到 χ Kripke 结构 $M'=(S',AP,\dot{S}_0,\to',\mathrm{B}_W,\mathrm{R}',\mathrm{L}$，则对 M 中的任意状态 s，任意 ACTL 公式 φ，有：$||\varphi||_M(s)=||\varphi||_{M'}(s')$，其中 x 为 Act 集合中任一元素。

证明： 下面我们对公式 φ 进行归纳来证明上面的定理。

归纳基础：

(1) $\varphi=\mathrm{true}$：

根据 $\|\text{true}\|^{M}(s)=\langle W,;\rangle$， $\|\text{true}\|^{M'}((s,x))=\langle W,;\rangle$，

得 $\|\text{true}\|^{M}(s)=\|\text{ks}'(\text{true})\|^{M'}((s,x))$。

(2) $\varphi=\text{false}$：

根据 $\|\text{false}\|^{M}(s)=\langle ;,W\rangle$， $\|\text{false}\|^{M'}((s,x))=\langle ;,W\rangle$，

得 $\|\text{false}\|^{M}(s)=\|\text{ks}'(\text{false})\|^{M'}((s,x))$。

归纳步骤：

(3) $\varphi=\neg\varphi_1$：

由定义可知：$\|\neg\varphi_1\|^{M}(s)=\neg\|\varphi_1\|^{M}(s)$，

又由归纳假设可知：$\|\varphi_1\|^{M}(s)=\|\text{ks}'(\varphi_1)\|^{M'}((s,x))$，

所以有：$\neg\|\varphi_1\|^{M}(s)=\neg\|\text{ks}'(\varphi_1)\|^{M'}((s,x))$；

又由定义可知：

$$\begin{aligned}&\neg\|\text{ks}'(\varphi_1)\|^{M'}((s,x))\\=&\|\neg\text{ks}'(\varphi_1)\|^{M'}((s,x))\\=&\|\text{ks}'(\neg\varphi_1)\|^{M'}((s,x)),\end{aligned}$$

由此得证：$\|\neg\varphi_1\|^{M}(s)=\|\text{ks}'(\neg\varphi_1)\|^{M'}((s,x))$。

(4) $\varphi=\varphi_1\wedge\varphi_2$：

由定义可知：$\|\varphi_1\wedge\varphi_2\|^{M}(s)=\|\varphi_1\|^{M}(s)\wedge\|\varphi_2\|^{M}(s)$；又因为 φ_1，φ_2 为公式 $\varphi_1\wedge\varphi_2$ 的子公式，所以由归纳假设得：$\|\varphi_1\wedge\varphi_2\|^{M}(s)=\|\text{ks}'(\varphi_1)\|^{M'}((s,x))\wedge\|\text{ks}'(\varphi_2)\|^{M'}((s,x))$；

又由定义可知：

$$\begin{aligned}&\|\text{ks}'(\varphi_1)\|^{M'}((s,x))\wedge\|\text{ks}'(\varphi_2)\|^{M'}((s,x))\\=&\|\text{ks}'(\varphi_1)\wedge\text{ks}'(\varphi_2)\|^{M'}((s,x))\\=&\|\text{ks}'(\varphi_1\wedge\varphi_2)\|^{M'}((s,x));\end{aligned}$$

由此得证：$\|\varphi_1\wedge\varphi_2\|^{M}(s)=\|\text{ks}'(\varphi_1\wedge\varphi_2)\|^{M'}((s,x))$。

(5) $\varphi=\varphi_1\vee\varphi_2$：根据对偶性，可证明 $\|\varphi_1\vee\varphi_2\|^{M}(s)=\|\text{ks}'(\varphi_1\vee\varphi_2)\|^{M'}((s,x))$。

(6) $\varphi=\langle a\rangle\varphi_1$（$a$ 为任一动作）：

由 ACTL 语义可知：$\|\langle a\rangle\varphi_1\|^{M}(s)=\bigvee_{s'\in S}(R(s,a,s')\wedge\|\varphi_1\|^{M}(s'))$；

根据 R s' (x (s, a)，，且由归纳假设可知 $\|\varphi_1\|^{M}(s')=\|\text{ks}'(\varphi_1)\|^{M'}((s',a))$，

所以有：$R(s,a,s')\wedge\|\varphi_1\|^{M}(s')=R'((s,x),(s',a))\wedge\|\text{ks}'(\varphi_1)\|^{M'}((s',a))$；

则 $\bigvee_{s'\in S}(R(s,a,s')\wedge\|\varphi_1\|^{M}(s'))=\bigvee_{s'\in S}(R'((s,x),(s',a))\wedge\|\text{ks}'(\varphi_1)\|^{M'}(s',a))$；

由定义可知：$\|\text{ks}'\langle a\rangle\varphi_1\|^{M'}((s,x))=\|\text{EX}(a\wedge\text{ks}'(\varphi_1))\|^{M'}((s,x))$，所以有：

$\| \mathrm{ks}'(\langle a\rangle\varphi_1) \|^{M'}((s,x))$

$=\| \mathrm{EX}(a \hat{} \mathrm{ks}'(\varphi_1)) \|^{M'}((s,x))$

$=_{(s',\overline{y})\in S'}(\mathrm{R}'((s,x),(s',y)) \hat{} \| a \hat{} \mathrm{ks}'(\varphi_1) \|^{M'}((s',y)))$

$=_{(s',\overline{y})\in S'}(\mathrm{R}'((s,x),(s',y)) \hat{} \| a \|^{M'}((s',y)) \hat{} \| \mathrm{ks}'(\varphi_1) \|^{M'}((s',y)))$

又因为当 $y \neq a$ 时，$\| a \|^{M'}(s',y)=\langle ;,W\rangle$，

所以，

$_{(s',\overline{y})\in S'}(\mathrm{R}'((s,x),(s',y)) \hat{} \| a \|^{M'}((s',y)) \hat{} \| \mathrm{ks}'(\varphi_1) \|^{M'}((s',y)))$

$=_{(s',\overline{a})\in S'}(\mathrm{R}'((s,x),(s',a)) \hat{} \| a \|^{M'}((s',a)) \hat{} \| \mathrm{ks}'(\varphi_1) \|^{M'}((s',a)))$

$=_{(s',\overline{a})\in S'}(\mathrm{R}'((s,x),(s',a)) \hat{} \| \mathrm{ks}'(\varphi_1) \|^{M'}((s',a)));$

所以 $\| \mathrm{ks}'(\langle a\rangle\varphi_1) \|^{M'}((s,x))=_{(s',\overline{a})\in S'}(\mathrm{R}'((s,x),(s',a)) \hat{} \| \mathrm{ks}'(\varphi_1) \|^{M'}((s',a)))$；

因此，$\| \langle a\rangle\varphi_1 \|^{M}(s)=\| \mathrm{ks}'(\langle a\rangle\varphi_1) \|^{M'}((s,x))$;

(7) $\varphi=[a]\varphi_1$ (a 为任一动作)：根据对偶性可证明。

(8) $\varphi=\mathrm{AF}\varphi_1,\mathrm{AG}\varphi_1,\mathrm{EF}\varphi_1,\mathrm{EG}\varphi_1$：此类公式的不动点计算可以由上述公式 1)~7)有限迭代计算得到结果，于是根据归纳假设证明结果成立。

综上所述：$\| \varphi \|^{M}(s)=\| \mathrm{ks}'(\varphi) \|^{M'}((s,x))$。

以信号灯软件产品线为例，设 ACTL 公式 φ 为 $\langle yellow\rangle\mathrm{true}$，则 φ 在产品线 BFTS 模型 M，状态 s_1 上的验证结果为：$\| \varphi \|^{M}(s_1)=\mathrm{R}(s_1,yellow,s_2) \hat{} \| \mathrm{true} \|^{M}(s_2)=\mathrm{R}(s_1,yellow,s_2)=\langle\{y\},\{s\}\rangle$ 接着，我们通过函数 ks' 将 φ 等价转换成 CTL 公式 φ'，φ' 为 $\mathrm{EX}\,yellow$。然后分别计算 φ' 在 χ Kripke 结构 M' 中的状态 (s_1,red)、$(s_1,yellow)$、$(s_1,green)$ 以及 $(s_1,return)$ 上的验证结果：

$\| \varphi' \|^{M'}((s_1,red))=_{s'\overline{\in}S'}(\mathrm{R}((s_1,red),s') \hat{} \| yellow \|^{M'}(s'))=\mathrm{R}((s_1,red),(s_2,yellow))=\langle\{y\},\{s\}\rangle$

$\| \varphi' \|^{M'}((s_1,yellow))=_{s'\overline{\in}S'}(\mathrm{R}((s_1,yellow),s') \hat{} \| yellow \|^{M'}(s'))=\mathrm{R}((s_1,yellow),(s_2,yellow))=\langle\{y\},\{s\}\rangle$

$\| \varphi' \|^{M'}((s_1,green))=_{s'\overline{\in}S'}(\mathrm{R}((s_1,green),s') \hat{} \| yellow \|^{M'}(s'))=\mathrm{R}((s_1,green),(s_2,yellow))=\langle\{y\},\{s\}\rangle$

$\| \varphi' \|^{M'}((s_1,return))=_{s'\overline{\in}S'}(\mathrm{R}((s_1,return),s') \hat{} \| yellow \|^{M'}(s'))=\mathrm{R}((s_1,return),(s_2,yellow))=\langle\{y\},\{s\}\rangle$

于是，φ 在 M 中状态 s_1 上的验证结果与 φ' 在 M' 中的状态 (s_1,red)，$(s_1,yellow)$，$(s_1,green)$ 以及 $(s_1,return)$ 上的验证结果相等，即：

$\| \varphi \|^{M}(s_1)=\| \varphi' \|^{M'}((s_1,red))=\| \varphi' \|^{M'}((s_1,yellow))=\| \varphi' \|^{M'}((s_1,green))=\| \varphi' \|^{M'}((s_1,return))$。

由定理 4.1 得知，任意 ACTL 公式在 BFTS 上的检测结果等于转换后的 CTL 公式在 χ Kripke 结构上的模型检测结果，于是通过 BFTS 以及 ACTL 的等价转换，基于 BFTS 的多值模型检测问题就可以转换为基于 χ Kripke 结构的多值模型检测问题。

4.2.2 基于模型分解的多值模型检测

多值模型检测就是给定一个多值模型 M 和一个时序逻辑公式 φ，通过模型检测工具计算出 φ 在 M 上的检测结果。该结果是一个多值逻辑值，反映了 φ 在 M 上的满足程度。实现多值模型检测的方法主要包含两类：一类为将多值模型问题转化为传统的二值模型检测问题进行研究，例如 Gurfindkel 等人[51,52]提出将多值模型检测问题分解为多个传统的二值模型检测问题，从而利用传统模型检测工具来实现多值模型检测；另一类是直接运用多值模型检测工具实现多值模型检测，常用的多值模型工具包括 χ Chek，STE[37]以及 YASM[38]等。

对于 BFTS 的多值模型检测问题，由于 BFTS 是基于世界双格的多值模型，其世界双格所包含的逻辑值个数随着特征的个数呈指数增长，即若特征集合 W 的元素个数为 n，其世界双格的逻辑值个数为 $2^n \times 2^n = 2^{2n}$。以例 1 中的饮料机产品线为例，根据饮料机产品线的特征图（如图 2.1 所示），产品线共包含 7 个特征，则产品线模型基于的世界双格由 2^{14} 个逻辑值构成，在实际中，一个软件产品线的特征个数可以达到上百个，这使得世界双格的规模过大，造成直接运用多值模型检测工具进行多值模型检测难以实施。为此，我们基于多值模型检测问题分解的思路，提出多值 BFTS 的分解方法，将多值 BFTS 上的模型检测问题分解为多个三值 BFTS 的模型检测问题。

三值模型作为一种特殊的多值模型也可以定义在 B_W 上，其中世界集合只包含一个世界，即 $W=\{w\}$。与传统的三值 Kripke 结构不同的是，R 定义在 R_W 上，即：R $S\times A\ c\times t\quad S\to\{\langle\{;\psi,\rangle\ \{\ ,\rangle\ \{$，并且时序逻辑公式在三值模型上的检测结果也用 $\{\langle\{w\},;\rangle,\langle;,;\rangle,\langle;,\mathrm{f}w\mathrm{g}\rangle\}$ 中的逻辑值表示。由于三值 BFTS 与多值 BFTS 可以用同一种方法表示，对于 $|W|=n$ 的 BFTS，可以将其相对于单个特征进行分解从而得到 n 个三值 BFTS，其中每个三值 BFTS 的世界双格逻辑值个数为 4，从而降低了格的规模。

定义 4.5 设 BFTS $M=(S,Act,s_0,\to,\mathrm{B}_W,\mathrm{R})$，特征集合 W 中包含 m 个元素($|W|=m$)，M 可**分解**为 m 个三值 BFTS $\{M_{w_1},...M_{w_i},...,M_{w_m}\}$。对任意的 $w_i\in W$，$M_{w_i}=(S_{w_i},Act,s_{0w_i},\to_{w_i},\mathrm{B}_{\{w_i\}},\mathrm{R}_{w_i})$，其中：

1) $S_{w_i}=S$；

2) $s_{0w_i}=s_0$；

3) $\to_{w_i}=\{(s,a,t)\,|\,\mathrm{R}_{w_i}(s,a,t)\neq\langle;,\{w_i\}\rangle\}$ 是 $\to$ 的一个子集；

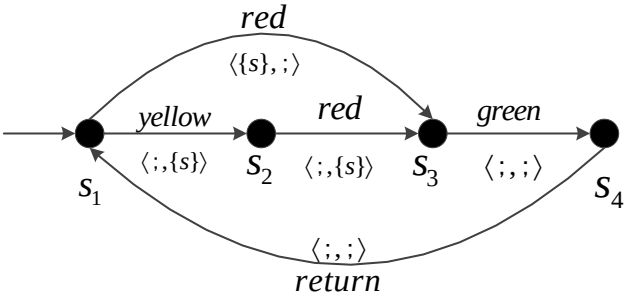
4) $\mathrm{B}_{\{w_i\}}=\langle P(w_i)\times P(w_i),\le_i,\le_t,\neg\rangle$ 为 B_W 的一个子双格；

5) $\mathrm{R}_{w_i}(s,a,t)=\mathrm{R}(s,a,t)\otimes\langle\{w_i\},\{w_i\}\rangle$。

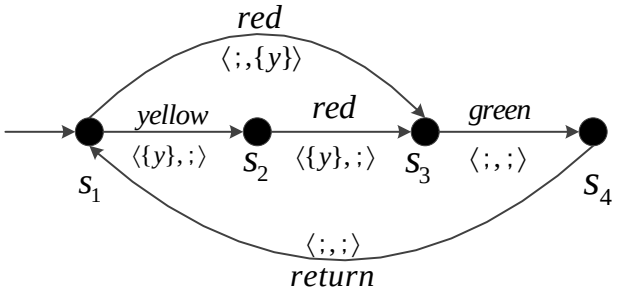
其中，条件 4)保证了 $\mathrm{B}_{\{w_i\}}=\{\langle\{w\},;\rangle,\langle;,;\rangle,\langle;,\{w\}\rangle\}\subseteq\mathrm{B}_W$，并且 $\le_i$，$\le_t$，$\neg$ 分别与定义在 B_W 上的序关系与补操作一致，因此 $\mathrm{B}_{\{w_i\}}$ 上的 _，ˆ，⊕，⊗ 操作亦与 B_W 上对应的操作一致。由

条件 5)易得 $\mathrm{R}(s,a,s')=\underset{w_i\in W}{\oplus}\mathrm{R}_{w_i}(s,a,s')$，即原 BFTS 上的迁移关系值等于分解后所有三值 BFTS 上对应迁移关系值在信息序关系下的并($\oplus$)。特别地，当原 BFTS 中特征集合 W 只包含一个特征时，即 $m=1$，通过定义 4.5 得到的三值 BFTS 与原 BFTS 相等。

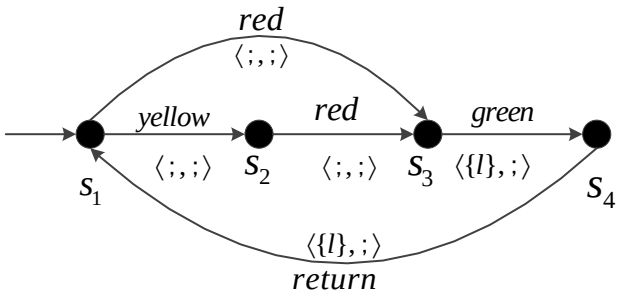
以例 2 中的信号灯软件产品线为例，图 4.2 为信号灯软件产品线 BFTS M 相对于特征集合中每一个特征 s、y、l 分解得到的三值 BFTS M_s（如图 4.2(a)所示）、M_y（如图 4.2(b)所示）、M_l（如图 4.2(c)所示）。以 M_s 为例，M_s 中迁移的逻辑值可以为$\langle;,\{s\}\rangle$、$\langle;,;\rangle$或$\langle\{s\},;\rangle$。M 中迁移 $(s_1,yellow,s_2)$ 的逻辑值为 $\langle\{y\},\{s\}\rangle$，于是该迁移的逻辑值在 M_s 中为 $\langle\{y\},\{s\}\rangle\otimes\langle\{s\},\{s\}\rangle=\langle\{s\},;\rangle$。同理，迁移 $(s_1,yellow,s_2)$ 在 M_y 中的逻辑值为 $\langle\{y\},\{s\}\rangle\otimes\langle\{y\},\{y\}\rangle=\langle\{y\},;\rangle$，在 M_l 中的逻辑值为$\langle\{y\},\{s\}\rangle\otimes\langle\{l\},\{l\}\rangle=\langle;,;\rangle$。



(a)



(b)



(c)

图 4.2 信号灯软件产品线 BFTS M 相对于特征 s、y、l 分解后的三值 BFTS：(a) M_s；(b) M_y；(c) M_l

接下来证明，BFTS M 分解为多个三值 BFTS，任意 ACTL 公式 φ 在 M 上的检测结果为 φ 在

分解后的所有三值 BFTS 上的检测结果在信息序关系上的并($\oplus$)。

定理 4.2 M 为定义在 B_W 上的 BFTS，其中 $|W|=m$， $m>1$。M 分解为 m 个三值模型 $\{M_{w_1},..M_{w_i},..M_{w_m}$，则对任意 ACTL 公式 φ，M 中的任意状态 s 有：$\|\varphi\|^M(s)=\underset{w_i\in W}{\oplus}\|\varphi\|^{M_{w_i}}(s)$.

证明：归纳基础：

(1) $\varphi=\text{true}$：

由于对于任意 BFTS M 中的任意状态 s，满足 $\|\text{true}\|^M(s)=\langle W,;\rangle$，

得 $\|\text{true}\|^M(s)=\underset{w_i\in W}{\oplus}\|\text{true}\|^{M_{w_i}}(s)$.

(2) $\varphi=\text{false}$：

由于对于任意 BFTS M 中的任意状态 s，满足 $\|\text{false}\|^M(s)=\langle;,W\rangle$，

得 $\|\text{false}\|^M(s)=\underset{w_i\in W}{\oplus}\|\text{false}\|^{M_{w_i}}(s)$.

归纳步骤：

(3) $\varphi=\neg\varphi_1$：

由定义可知：$\|\neg\varphi_1\|^M(s)=\neg\|\varphi_1\|^M(s)$，

又由归纳假设可知：$\|\varphi_1\|^M(s)=\underset{w_i\in W}{\oplus}\|\varphi_1\|^{M_{w_i}}(s)$，

所以有：$\neg\|\varphi_1\|^M(s)=\neg(\underset{w_i\in W}{\oplus}\|\varphi_1\|^{M_{w_i}}(s))$。

设 $\|\varphi_1\|^{M_{w_i}}(s)=\langle X_{w_i},Y_{w_i}\rangle$，

则有：$\underset{w_i\in W}{\oplus}\|\varphi_1\|^{M_{w_i}}(s)=\underset{w_i\in W}{\oplus}\langle X_{w_i},Y_{w_i}\rangle=\langle\underset{w_i\in W}{[}X_{w_i},\underset{w_i\in W}{[}Y_{w_i}\rangle$。

又由

$$\begin{aligned}\neg\underset{w_i\in W}{\oplus}\|\varphi_1\|^{M_{w_i}}(s)&=\langle\underset{w_i\in W}{[}Y_{w_i},\underset{w_i\in W}{[}X_{w_i}\rangle\\&=\underset{w_i\in W}{\oplus}\langle Y_{w_i},X_{w_i}\rangle\\&=\underset{w_i\in W}{\oplus}(\neg\langle X_{w_i},Y_{w_i}\rangle)\\&=\underset{w_i\in W}{\oplus}\neg\|\varphi_1\|^{M_{w_i}}(s)\\&=\underset{w_i\in W}{\oplus}\|\neg\varphi_1\|^{M_{w_i}}(s),\end{aligned}$$

所以：$\|\neg\varphi_1\|^M(s)=\underset{w_i\in W}{\oplus}\|\neg\varphi_1\|^{M_{w_i}}(s)$。

(4) $\varphi=\varphi_1\hat{}\,\varphi_2$：

由定义可知：$\|\varphi_1\hat{}\,\varphi_2\|^M(s)=\|\varphi_1\|^M(s)\hat{}\,\|\varphi_2\|^M(s)$。

又因为 φ_1， φ_2 为公式 $\varphi_1\hat{}\,\varphi_2$ 的子公式，

所以由归纳假设得：$\|\varphi_1\hat{}\,\varphi_2\|^M(s)=\underset{w_i\in W}{\oplus}\|\varphi_1\|^{M_{w_i}}(s)\hat{}\,\underset{w_i\in W}{\oplus}\|\varphi_2\|^{M_{w_i}}(s)$。

令 $\|\varphi_1\|^{M_{w_i}}(s)=\langle A_{w_i},B_{w_i}\rangle$， $\|\varphi_2\|^{M_{w_i}}(s)=\langle C_{w_i},D_{w_i}\rangle$，

则

$$\begin{aligned}
&\oplus_{w_i\in W}\|\varphi_1\|^{M_{w_i}}(s)\ \hat{}\ \oplus_{w_i\in W}\|\varphi_2\|^{M_{w_i}}(s)\\
&=\oplus_{w_i\in W}\langle A_{w_i},B_{w_i}\rangle\ \hat{}\ \oplus_{w_i\in W}\langle C_{w_i},D_{w_i}\rangle\\
&=\langle[_{w_i\in W}A_{w_i},[_{w_i\in W}B_{w_i}\rangle\ \hat{}\ \langle[_{w_i\in W}C_{w_i},[_{w_i\in W}D_{w_i}\rangle \qquad (1)\\
&=\langle([_{w_i\in W}A_{w_i})\setminus([_{w_i\in W}C_{w_i}),([_{w_i\in W}B_{w_i})[([_{w_i\in W}D_{w_i})\rangle\\
&=\langle[A_{w_1}\setminus([_{w_i\in W}C_{w_i})][\cdots[A_{w_m}\setminus([_{w_i\in W}C_{w_i})],[B_{w_1}\setminus([_{w_i\in W}D_{w_i})][\cdots[B_{w_m}\setminus([_{w_i\in W}D_{w_i})]\rangle。
\end{aligned}$$

又因为 $A_{w_i}\in \mathrm{P}(\{w_i\})$，$C_{w_i}\in \mathrm{P}(\{w_j\})$（$\mathrm{P}(U)$ 表示集合 U 的幂集），

因此若 $i\neq j$，则 $\mathrm{P}(\{w_i\})\setminus \mathrm{P}(\{w_j\})=;$，即 $A_{w_i}\setminus C_{w_i}=;$。

于是

$$\begin{aligned}
(1)&=\langle(A_{w_1}\setminus C_{w1})[\cdots(A_{w_m}\setminus C_{w_m}),(B_{w_1}[D_{w1})[\cdots(B_{w_m}[D_{w_m})\rangle\\
&=\oplus_{w_i\in W}\langle A_{w_i}\setminus C_{w_i},B_{w_i}[D_{w_i}\rangle\\
&=\oplus_{w_i\in W}(\langle A_{w_i},B_{w_i}\rangle\ \hat{}\ \langle C_{w_i},D_{w_i}\rangle)\\
&=\oplus_{w_i\in W}(\|\varphi_1\ \hat{}\ \varphi_2\|^{M_{w_i}}(s))
\end{aligned}$$

由此 $\oplus_{w_i\in W}\|\varphi_1\|^{M_{w_i}}(s)\ \hat{}\ \oplus_{w_i\in W}\|\varphi_2\|^{M_{w_i}}(s)=\oplus_{w_i\in W}(\|\varphi_1\ \hat{}\ \varphi_2\|^{M_{w_i}}(s))$，　(2)

则 $\|\varphi_1\ \hat{}\ \varphi_2\|^{M}(s)=\oplus_{w_i\in W}\|\varphi_1\ \hat{}\ \varphi_2\|^{M_{w_i}}(s)$。

(5) $\varphi=\varphi_1\_\varphi_2$：根据对偶性可证明。

(6) $\varphi=\langle a\rangle\varphi_1$（$a$ 为任一动作）：

由 ACTL 公式语义可知：$\|\langle a\rangle\varphi_1\|^{M}(s)=\overline{\ }_{s'\in S}(\mathrm{R}(s,a,s')\ \hat{}\ \|\varphi_1\|^{M}(s'))$，

根据 $\mathrm{R}(s,a,s')=\oplus_{w_i\in W}(\mathrm{R}_{w_i}(s,a,s'))$，

且由归纳假设可知：$\|\varphi_1\|^{M}(s')=\oplus_{w_i\in W}\|\varphi_1\|^{M_{w_i}}(s')$，

则 $\mathrm{R}(s,a,s')\ \hat{}\ \|\varphi_1\|^{M}(s')=\oplus_{w_i\in W}\mathrm{R}_{w_i}(s,a,s')\ \hat{}\ \oplus_{w_i\in W}\|\varphi_1\|^{M_{w_i}}(s')$。　(3)

由(2)式可知：

$$\begin{aligned}
(3)&=\oplus_{w_i\in W}(\mathrm{R}_{w_i}(s,a,s')\ \hat{}\ \|\varphi_1\|^{M_{w_i}}(s'))\\
&=\oplus_{w_i\in W}\|\langle a\rangle\varphi_1\|^{M_{w_i}}(s)
\end{aligned}$$

(7) $\varphi=[a]\varphi_1$（a 为任一动作）：根据对偶性可证明。

(8) $\varphi=\mathrm{AF}\varphi_1,\mathrm{AG}\varphi_1,\mathrm{EF}\varphi_1,\mathrm{EG}\varphi_1$：此类公式的不动点计算可以由上述公式 1)~7)有限迭代计算得到结果，于是根据归纳假设证明结果成立。

综上所述：对任意 ACTL 公式 φ，BFTS M 中的任意状态 s，有 $\|\varphi\|^{M}(s)=\oplus_{w_i\in W}\|\varphi\|^{M_{w_i}}(s)$。

以信号灯软件产品线为例，设 ACTL 公式 φ 为 $\langle yellow\rangle$true，则 φ 在产品线行为模型 BFTS M 的状态 s_1 上的验证结果为 $\langle\{y\},\{s\}\rangle$（其计算过程已在 4.2.1 节中给出），$\varphi$ 在 M_s、M_y、M_l

中的状态 s_1 上的验证结果分别为 $\langle ;,\{s\}\rangle$、$\langle \{y\},;\rangle$、$\langle ;,;\rangle$。于是，φ 在 M 上的验证结果等于 φ 在 M_s、M_y、M_l 上的验证结果的并($\oplus$)，即 $\|\varphi\|^{M}(s_1)=\|\varphi\|^{M_s}(s_1)\oplus\|\varphi\|^{M_y}(s_1)\oplus\|\varphi\|^{M_l}(s_1)$。

由定理 4.2 得知，任意 ACTL 公式 φ 在 BFTS M 上的检测结果为 φ 在 M 相对于单个特征分解后的所有三值 BFTS 上的检测结果在信息序关系上的并($\oplus$)。由此，基于 BFTS 的多值模型检测问题就可以转换为基于模型分解的多值模型检测问题。

4.3 本章小结

本章首先介绍了 ACTL 时序逻辑，并给出定义在 BFTS 上的语义，提出基于 BFTS 的多值模型检测。其次，我们研究了基于 BFTS 的多值模型检测的实现方式。实现该模型检测有两种方式：1)通过模型转换和公式转换将原 BFTS 多值模型检测问题转换为 χ Kripke 结构上的多值模型检测问题，从而可以利用基于 χ Kripke 结构的多值模型检测工具完成 BFTS 上的模型检测；2)通过 BFTS 的模型分解将原多值 BFTS 模型检测问题转换为多个三值 BFTS 模型检测问题，从而可以利用传统的二值模型检测工具完成 BFTS 上的模型检测。

于是，针对这两种实现方式，我们一方面提出了从 BFTS 到 χ Kripke 结构上的等价转换以及从 ACTL 公式到 CTL 公式的转换函数，并证明原 ACTL 公式在 BFTS 上的模型检测结果等于转换后的 CTL 公式在 χ Kripke 结构上的检测结果，从而实现方式 1)；另一方面，提出多值模型 BFTS 相对于软件产品线中的单个特征分解的方法，并证明了 ACTL 公式在原多值 BFTS 上的检测结果等于在分解后多个三值 BFTS 上的检测结果的并($\oplus$)，从而实现方式 2)。

第五章 实验设计与分析

本章主要介绍实验设计与结果分析。首先，介绍多值模型检测工具 χ Chek，重点介绍输入文件格式；其次，介绍开发的辅助工具——基于双格的软件产品线的不完备模型检测工具(Bilattice-based Partial Model Checker Assistant of Software Product Line，简称 BPMCA)，包括其开发环境以及相关算法的设计；最后以饮料机产品线为例，设计实验过程，并展示实验结果，以验证方法的有效性。

5.1 χ Chek 简介

χ Chek[40]是由加拿大多伦多大学(University of Toronto)开发的一款基于 JAVA 模型检测工具。该工具可支持多值模型上的模型检测，并能生成导致公式不能在模型上验证正确的反例。本文采用 χ Chek 实现软件产品线的多值模型检测，其输入模型是 χ Kripke 结构，并且模型以及多值逻辑构成的代数格均采用可扩展标记语言(eXtensible Markup Language，简称 XML)描述。

XML 是一种允许用户对自己的标记语言进行定义的源语言，可以用来标记数据、定义数据类型。在 χ Kripke 结构的 XML 文件中，定义了从 χ Kripke 结构到有向图的映射，其中有向图采用图形化可扩展语言(Graph eXchange Language，简称 GXL)描述，图中的结点(node)被原子命题标记，边(edge)采用多值逻辑标记，从而实现了对模型中状态和迁移的描述。

χ Chek 中对模型进行描述的 XML 文件包括以下 4 部分内容：

1) 命名空间的声明；
2) 描述模型中多值逻辑的 XML 文件的链接；
3) 结点：每一个结点代表模型中的一个状态，即每一个<node>标签必须有唯一标识，嵌套的<attr>标签用于标记该状态上的原子命题；此外，描述初始状态的结点必须在<node>标签后添加属性“xbel：initial='true'”；
4) 边：每一个边代表模型中的一个迁移，即每一个<edge>标签标记了一条模型的迁移，嵌套的<attr>标签用于标记迁移的逻辑值。

以图 2.7 中的四值 χ Kripke 结构 M 为例，M 包含三个状态：s_1、s_2、s_3，原子命题集合 $AP=\{p,q\}$。描述 M 的 XML 文件如表 5.1 所示，其中第 1~2 行声明了命名空间，且说明此图为有向图(edgemode='directed')；第 6~18 行描述了 M 中的状态，就每一个状态描述状态上所有原子命题的逻辑值并声明初始状态，如第 7 行到第 10 行描述了状态 s_1 为初始状态(xbel:initial='true')，状态上的原子命题 p 的逻辑值为$\langle ;,;\rangle$（第 8 行所示），状态上的原子命题 q 为$\langle \{w\},;\rangle$（第 9 行所示）；第 20~31 行描述了 M 中的迁移，如第 20~22 行描述了从 s_1 到 s_2 的迁移，其逻辑值为$\langle \{w\},;\rangle$。

表 5.1 四值 χ Kripke 结构 M 的 XML 文件

```
1.<gxl xmlns:xbel=’www.cs.toronto.edu/xbel’ xmlns:xlink=’xlink’>
2.   <graph ID=’model_example’ edgemode=’directed’>
3.      <!-- LINK TO LOGIC -->
4.      <xbel:logic xlink:type=’simple’ xlink:href=’examples/xml/4val-logic.xml’/>
5.      <!-- MODEL -->
6.      <!-- NODES -->
7.      <node ID=’s1’ xbel:initial=’true’>
8.         <attr type="prop" name="p" value="&lt；{},{}&gt； "/>
9.         <attr type="prop" name="q" value="&lt； {w},{}&gt； "/>
10.     </node>
11.     <node ID=’s2’>
12.        <attr type="prop" name="p" value="&lt； {w},{}&gt； "/>
13.        <attr type="prop" name="q" value="&lt； {},{}&gt； "/>
14.     </node>
15.     <node ID=’s3’>
16.        <attr type="prop" name="p" value="&lt； {},{}&gt； "/>
17.        <attr type="prop" name="q" value="&lt； {},{}&gt； "/>
18.     </node>
19.     <!-- EDGES -->
20.     <edge from=’s1’ to=’s2’>
21.        <attr name=’ weight’  value=’  &lt；{w},{}&gt;’ />
22.     </edge>
23.     <edge from=’s1’ to=’s3’>
24.        <attr name=’ weight’  value=’  &lt；{},{}&gt;’ />
25.     </edge>
26.     <edge from=’s2’ to=’s3’>
27.        <attr name=’ weight’  value=’  &lt；{w},{}&gt;’ />
28.     </edge>
29.     <edge from=’s3’ to=’s3’>
30.        <attr name=’ weight’  value=’  &lt；{},{}&gt;’ />
31.     </edge>
32.  </graph>
33.</gxl>
```

在描述多值逻辑方面，其描述文件应包括以下 4 部分内容：

(1) 命名空间的声明；

(2) 逻辑值：每个<node>标签标记格中的一个逻辑值，<node>标签可以有一个可选的<attr>标签对相应的逻辑值进行描述；

(3) 逻辑序关系：通过嵌套的<above>标签描述格中逻辑值顺序，每个<edge>标签描述格中一条边；

(4) 否定关系：通过嵌套的<neg>标签描述各种逻辑值间的否定关系，在文件中必须给出所有逻辑值间的否定关系。

以四值逻辑 $\mathrm{B}_{\{w\}}$（P15 中的图 2.6）为例，$\mathrm{B}_{\{w\}}$ 包含四个逻辑值：$\langle\{w\},;\rangle$、$\langle;,;\rangle$、$\langle\{w\},\{w\}\rangle$、$\langle;,\{w\}\rangle$。我们采用 XML 描述该逻辑如表 5.2 所示。其中第 1~3 行声明了空间，且说明此图为

有向图(edgemode='directed')；第 5~8 行描述了 $B_{\{w\}}$ 中的逻辑值，如第 5 行描述了逻辑值〈;,;〉；第 10~21 行描述了 $B_{\{w\}}$ 中的 above 关系，如第 10~12 行描述了 above 关系〈;,;〉[1] 〈{*w*},;〉；第 23~31 行描述了 $B_{\{w\}}$ 中的否定关系，如第 23~25 行描述了否定关系：〈{*w*},;〉=〈;,{*w*}〉。

表 5.2 四值逻辑 B_w 的 XML 文件

```
 1.<gxl xmlns:xbel='www.cs.toronto.edu/xbel' xmlns:xlink='xlink'>
 2.   <graph ID='4-val' edgemode='directed'>
 3.     <xbel:img xlink:type='simple' xlink:href='examples/xml/4val.gif'/>
 4.     <!-- LOGIC VALUES -->
 5.     <node ID=' &lt;{},{}&gt;' />
 6.     <node ID=' &lt;{w},{}&gt;' />
 7.     <node ID=' &lt;{},{w}&gt;' />
 8.     <node ID=' &lt;{w},{w}&gt;' />
 9.     <!-- LOGIC ORDER -->
10.     <edge from=' &lt;{w},{}&gt;' to=' &lt;{},{}&gt;' >
11.       <type value='above'/>
12.     </edge>
13.     <edge from==' &lt;{w},{}&gt;' to=' &lt;{w},{w}&gt;' >
14.       <type value='above'/>
15.     </edge>
16.     <edge from=' &lt;{w},{w}&gt;' to=' &lt;{},{w}&gt;' >
17.       <type value='above'/>
18.     </edge>
19.     <edge from=' &lt;{},{}&gt;' to=' &lt;{},{ w}&gt;' >
20.       <type value='above'/>
21.     </edge>
22.     <!-- NEGATION ORDER -->
23.     <edge from=' &lt;{w},{}&gt;' to=' &lt;{},{w}&gt;' >
24.       <type value='neg'/>
25.     </edge>
26.     <edge from=' &lt;{ },{ }&gt;' to=' &lt;{},{}&gt;' >
27.       <type value='neg'/>
28.     </edge>
29.     <edge from=' &lt; { },{w }&gt;' to=' &lt;{w},{}&gt;' >
30.       <type value='neg'/>
31.     </edge>
32.   </graph>
33.</gxl>
```

5.2 BPMCA 工具设计

本节主要介绍我们开发的工具 BPMCA。由于 χ Chek 的输入模型是 χ kripke 结构，输入公式是 CTL 公式，我们将基于 BFTS 的多值模型检测问题转换成基于 χ kripke 结构的多值模型检测问题。基于 4.2 节中提出的转换方法，我们开发了工具 BPMCA 来实现模型的自动转换。首先我们介绍工具开发的环境，然后给出相关算法以及类的设计。

5.2.1 BPMCA 工具的架构设计与实现

BPMCA 工具主要功能是实现从 BFTS 到 χ kripke 结构的自动转换，并能调用 Graphviz 工具绘制 BFTS，然后自动生成化简后的 χ kripke 结构以及代数格的 XML 文件。

图 5.1 为工具 BPMCA 的设计框架图。虚线方框内的部分为工具的主体，主要完成的工作包括：1)模型的输入；2)模型的转换和化简；3)多值逻辑的生成；4)生成模型与格的 XML 文件；5)用 Graphviz 生成 BFTS 模型状态自动机图并显示，调用 χ Chek 对模型进行检验。

工具的数据流如箭头方向所示。模型的输入可通过创建新模型或导入模型文件实现。输入模型后，可以选择调用 Graphviz 绘制输入模型。根据模型转换算法，BFTS 模型转换成为 χ Kripke 结构，并删去不可到达的状态和从这些状态出发的迁移完成模型的化简，最后相应的 XML 文件。另一方面，根据 BFTS 的特征集合 W，构建模型所基于的世界双格，并生成 XML 文件。

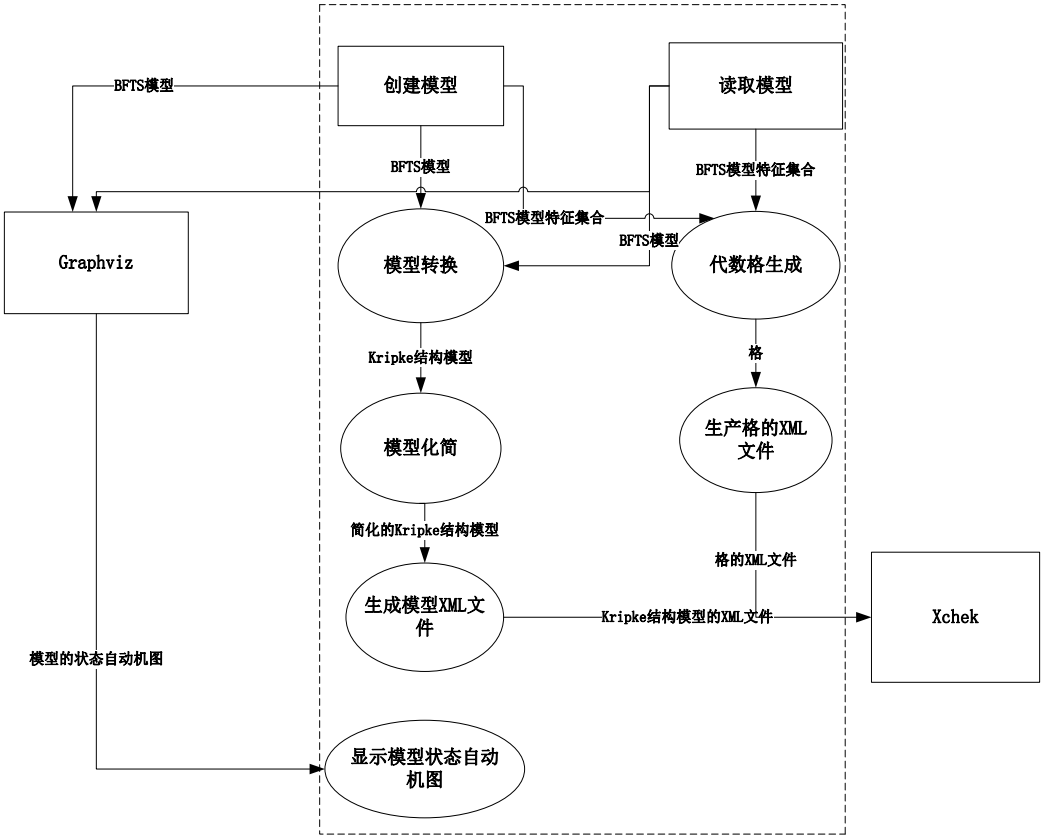


图 5.1 工具 BPMCA 的设计框架

BPMCA 工具是基于 Qt 和 Graphviz 工具开发的。Qt 是由奇趣公司于 1991 年开发的一个 C++应用程序开发框架。它既支持 GUI 程序的开发，也支持非 GUI 程序开发，如控制台工具和服务器等。作为面向对象的框架，Qt 使用特殊的代码生成扩展。在 C、C++图形界面开发工具相对较少的 Linux 背景下，Qt 显得尤为突出。考虑到 Qt 具有的这些优秀特性可以使得工具的开发较为方便且能提供更强大的功能拓展，我们最终选择了 Qt 作为开发环境。此外，

Graphviz(Graph Visualization Software)作为辅助工具可实现模型的可视化，即通过输入模型的所有信息，工具能自动生成模型的可视化图示，这有助于在模型检测过程中对模型的直观认识，以及通过图示完成对模型信息的核对。Graphviz 是一个由 AT&T 实验室开发的开源工具包，使用 DOT 脚本语言描述图形，可以自动的对描述的图形进行绘制。它的理念不同于一般的“所见即所得”的画图工具，而是采用了“所想即所得”的思想，即在绘图过程中，并不需要考虑结点的分布以及图形的布局等，只需要给出图中点和边及其他元素的信息，它就可以自动地以比较美观的方式绘制出图形。

图 5.2 为 BPMCA 工具类图。实现软件主要涉及了 4 个类：BFTS 类、GraphViz 类、CMarkup 类、Dialog 类。

(1) BFTS 类为实现软件功能最为核心的类，用于完成模型的装载，模型的转换等主要功能。BFTS 类调用了 GraphViz 类及 CMarkup 类用于实现相关的操作。

(2) GraphViz 类用于调用 GraphViz 工具生成 BFTS 模型图，主要实现了调用 GraphViz 工具的各种接口。

(3) CMarkup 类中包括了生成 XML 文件所需要的数据结构以及操作。

(4) Dialog 类是一个界面类，主要实现界面上各种信号的触发操作。

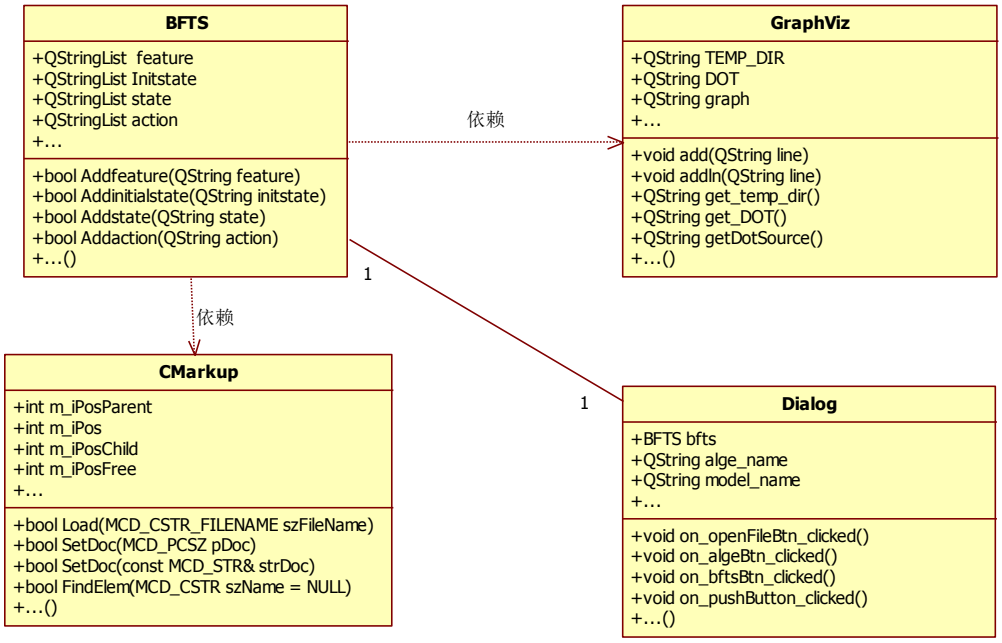


图 5.2 BPMCA 工具类图

对于 BFTS 类，类中属性（如表 5.3 所示）存储了 BFTS 模型中的特征、状态、迁移等内容，并且规定了 XML 文件的存储路径。其中 algepath 属性保存了格的 XML 文件的绝对路径，在生成模型的 XML 文件中的多值逻辑链接时将用到这一属性。BFTS 类中主要操作为 toXMLless 与 algebraGeneration（如表 5.4 所示），前者用于生成化简的 Kripke 模型的 XML 文件，后者用

于生成格的 XML 文件。

表 5.3 BFTS 类中的属性

属性名	数据类型	含义
feature	QStringList	BFTS 模型中的特征
initialstate	QStringList	BFTS 模型中的初始状态
state	QStringList	BFTS 模型中状态
action	QStringList	BFTS 中的动作
ori	QStringList	BFTS 中迁移的起点状态
dest	QStringList	BFTS 中迁移的终点状态
act	QStringList	BFTS 中迁移的动作
R	QStringList	BFTS 中迁移的标记值
xmldir	QString	XML 文件存储目录
algepath	QString	格的 XML 文件绝对路径

表 5.4 BFTS 类中的操作

操作声明	功能
bool Addfeature(QString feature)	添加特征
bool Addinitialstate(QString initstate)	添加初始状态
bool Addstate(QString state)	添加状态
bool Addaction(QString action)	添加动作
int Addtrans(QString ori，QString dest，QString act，QString R)	添加迁移
QString loadwbt(QString filepath)	从文件中读入 BFTS
bool visit()	遍历 BFTS
bool toXMLFull(QString filename)	将 BFTS 转换成未化简的 Kripke 结构并生成 XML 文件
bool toXMLLess(QString filename)	将 BFTS 转换成化简的 Kripke 结构，并生成 XML 文件
bool AlgebraGeneration(QString filename)	生成代数格并生成 XML 文件
void draw_picture()	生成模型状态自动机图
void clear()	清空模型信息
void saveModel(QString filepath)	将模型存储为文本文件

5.2.2 转换模型与代数格生成算法

对于转换模型的生成，根据定义 4.3，通过 BFTS 模型中的状态集合和动作集合的笛卡尔积得到等价 χ Kripke 结构的状态集合。然而，在该状态集合中，存在不可达状态，即模型中不存在迁移到达这些状态，如图 4.1(c)所示信号灯系统中的状态 (s_2, red)，$(s_2, green)$，$(s_2, yellow)$。这些状态的存在与否不影响模型检测结果，但会增大模型的规模，降低模型检测的效率。基于此，转换模型的生成算法通过去除生成模型中不可达状态以及从这些状态发出的迁移，得到简化后的模型。

转换模型的生成算法输入为 BFTS 模型，可通过导入 BFTS 模型文件，也可以通过手动输入模型各项信息：初始状态 *Initial State*，状态集合 *State*，动作集合 *Action*，迁移集合 *Transition*，特征集合 *Feature*；其输出为 χ Kripke 结构的 XML 文件。转换模型的生成算法流程见算法 5.1。

算法 5.1 由 BFTS 模型转换到 χ Kripke 模型并完成化简

输入： BFTS 模型
输出： 化简后的 χ Kripke 模型 XML 文件

```
1:  bfts is a BFTS model
2:  TrueState is a set of state in χ Kripke model
3:  TrueState is empty at first
4:  newstate is a temp variable for generating the new state in χ Kripke model
5:  neworigin and newdest are the same as newstate
6:  FOR EACH transition of bfts
7:      newstate=（transition.dest ，transition.act）;
8:      IF TrueState doesn't contain newstate THEN
9:        add newstate to TrueState;
10:     END IF
11:   END FOR
12:   FOR EACH state of bfts
13:     FOR EACH action of bfts
14:       newstate=(state，action);
15:       IF TrueState contains newstate THEN
16:         describe the newstate in the XML File
17:       END IF
18:     END FOR
19:   END FOR
20:   FOR EACH transition of bfts
21:     FOR EACH action of bfts
22:       neworigin=(transition.ori，action);
23:       IF TrueSate contains neworigin
24:         newdest=(transion.dest，transition.act);
25:         newtransition=(neworigin，newdest，transition.R);
26:         describe the newtransition in the XML File
27:       END IF
28:     END FOR
29:   END FOR
30:   Save the XML File
```

对于算法 5.1，其时间复杂度为$O(n^2)$，其基本思想是首先将原 BFTS 中的到达状态集合与动作集合 Action 做笛卡尔积获取χ Kripke 结构的状态集合 TrueState。其次，将原 BFTS 中状态集合 State 与 Action 做笛卡尔积，得到新的状态集合 newstate，若 newstate 中元素属于 TrueState 集合，则表示该状态是可达的。最后将新的 Transitions 集合中起点状态不在 TrueState 中的迁移删除，得到化简后的 Transitions 集合。

对于代数格的生成算法，根据定义 2.8，BFTS 模型中的特征集合W与自身的笛卡尔积为代数格的逻辑值集合，即$\langle U,V\rangle \in W\times W$。在 above 关系上，若逻辑值$\langle U_1,V_1\rangle$与$\langle U_2,V_2\rangle$满足$V_1=V_2$且$\mathrm{card}(U_1)==\mathrm{card}(U_2)+1$，则$\langle U_1,V_1\rangle$，$\langle U_2,V_2\rangle$满足 above 关系。此外，$\langle U,V\rangle$与$\langle V,U\rangle$互为否定关系。基于此，我们给出多值逻辑的生成算法，输入为 BFTS 模型中的特征集合，输出为描述代数格的 XML 文件。算法流程见算法 5.2，其时间复杂度为$O(n^2)$。

算法 5.2：生成模型多值逻辑的算法

输入：BFTS 模型中的特征集合
输出：描述代数格的 XML 文件

```
1:  W stands for the set of all feature
2:  U , V , M , N is the subset of W
3:  ⟨U,V⟩ and ⟨M,N⟩ stand for the logic value in the lattice
4:  describe namespace in the XML File
5:  FOR EACH logic value ⟨U,V⟩
6:      describe ⟨U,V⟩ in the XML File
7:  FOR EACH logic value ⟨U,V⟩
8:      For EACH logic value ⟨M,N⟩
9:          IF card(U)==card(M)−1 and V==N THEN
10:             describe ⟨U,V⟩ above ⟨M,N⟩ in the XML File
11:         END IF
12:         IF card(V)==card(N)+1 and U==M THEN
13:             describe ⟨U,V⟩ above ⟨M,N⟩ in the XML File
14:         END IF
15:     END FOR
16: FOR
17: FOR EACH ⟨U,V⟩
18:     describe ⟨U,V⟩ neg ⟨V,U⟩ in the XML File
19: END FOR
20: Save the XML FILE
```

5.2.3 BPMCA 工具的使用

我们以 4.2 节中的例 2 为例，通过 BPMCA 工具实现信号灯系统模型的自动转换以及相应的模型和代数格的 XML 文件生成。BPMCA 工具包含三个界面：模型输入界面(如图 5.3)、模型转换界面(如图 5.4)和模型可视化界面(如图 5.5)；其中模型输入界面用于 BFTS 模型信息的导入，模型转换界面用于自动生成对应的χ Kripke 结构模型 XML 文件以及所基于的多值逻辑代数格的 XML 文件，模型可视化界面用于生成 BFTS 模型的图示。

BPMCA 工具的输入界面如图 5.3 所示，输入模型名称后点击按钮“create BFTS model”，则完成 BFTS 模型文件的命名，此时文件中信息为空。然后输入 BFTS 模型中的所有信息，包括初始状态、状态、动作、迁移、特征、迁移逻辑值，如在“input feature”一栏中输入特征 *light*，点击“next”，即可保存已输入特征并继续输入其他特征，待所有特征输入完毕后点击“finish”保存所有特征输入信息。同理，输入模型其他信息，完成所有信息输入后，点击“save”，选择保存路径，得到信号灯软件产品线的 BFTS 模型文件，如表 5.5 所示，表中“s1 s2 yellow <{y},{s}>”表示迁移$(s1, yellow, s2)$的逻辑值为$\langle\{y\},\{s\}\rangle$。

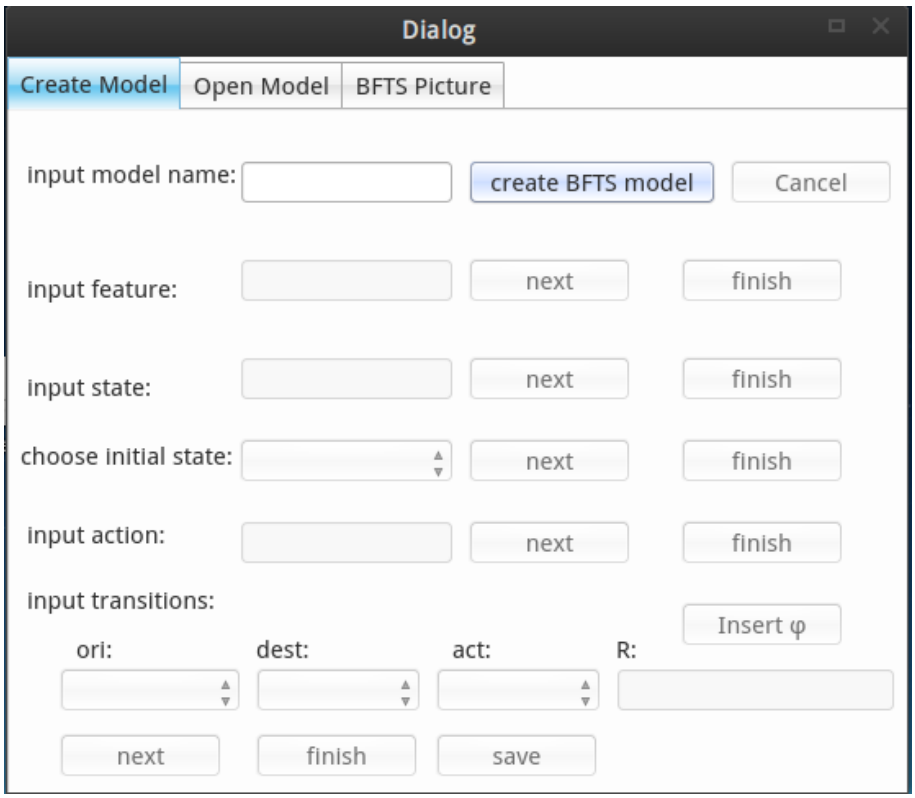


图 5.3 BPMCA 工具的输入界面

表 5.5 信号灯软件产品线的 BFTS 模型的文本描述

Feature:	**Action:**
light	yellow
yellow	red
skip_yellow	green
State:	return
s1	**Transitions:**
s2	s1 s2 yellow <{y},{s}>
s3	s1 s3 red <{s},{y}>
s4	s2 s3 red <{y},{s}>
Initial State:	s3 s4 green <{l},{}>
s1	s4 s1 return <{l},{}>

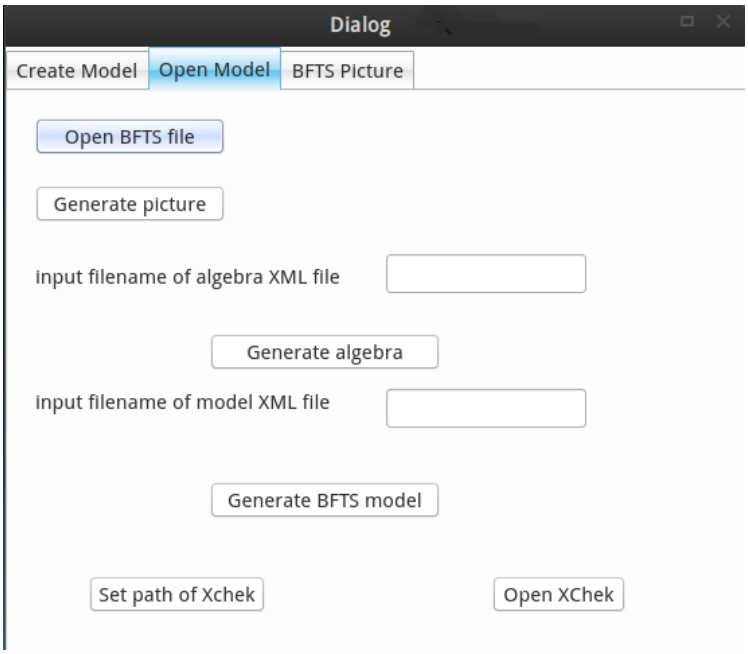


图 5.4 BPMCA 工具的模型转换界面

随后在模型转换界面中导入 BFTS 模型文件，然后可依次输入格的 XML 文件名和模型的 XML 文件名，并点击按钮生成文件。最后可以点击 Open χChek 按钮打开 χChek 对模型进行检验。此外，将界面切换到模型可视化界面中可自动绘制输入的 BFTS。

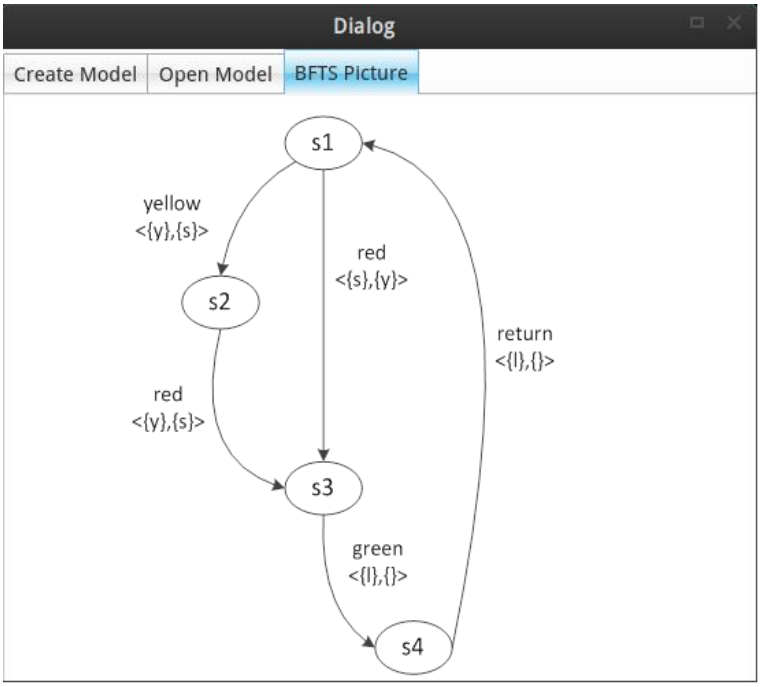


图 5.5 信号灯产品线的 BFTS 模型可视化显示

5.3 实例分析

本节主要以饮料机产品线为例，运用 χ Chek，完成饮料机产品线上的模型检测。为此，我们设计了相关待检测属性，其 ACTL 公式如表 5.6 中第 1 列所示。下面我们根据 ACTL 语法，分别解释每一个公式所描述的系统属性：

1) $AGAF((\langle tea\rangle \text{true}\_\langle coffee\rangle \text{true}) \rightarrow AF\langle cup\_taken\rangle \text{true})$：描述属性“无论系统处于什么状态，在将来如果用户选择了茶或者咖啡，那么将来用户一定会将杯子拿走”；
2) $AGAF(\langle sugar\rangle \text{true} \rightarrow AF\langle pour\_sugar\rangle \text{true})$：描述属性“无论系统处于什么状态，在将来如果用户选择了加糖，那么将来系统一定会加糖”；
3) $AGAF(\langle sugar\rangle \text{true}\_\langle no\_sugar\rangle \text{true})$：描述属性“无论系统处于什么状态，在将来用户都需要选择加糖或不加糖”；
4) $AGAF\langle ring\_a\_tone\rangle \text{true}$：描述属性“无论系统处于什么状态，在将来系统都会要响铃”；
5) $AGAF\langle display\_done\rangle \text{true}$：描述属性“无论系统处于什么状态，在将来系统都会显示‘完成’”。

表 5.6 ACTL 公式到 CTL 公式的转换 $ks': ACTL \rightarrow CTL$

ACTL 公式 φ	CTL 公式 φ'
$AGAF((\langle tea\rangle \text{true}\_\langle coffee\rangle \text{true}) \rightarrow AF\langle cup\_taken\rangle \text{true})$	$AGAF((EXtea\_EXcoffee) \rightarrow AFEXcup\_taken)$
$AGAF(\langle sugar\rangle \text{true} \rightarrow AF\langle pour\_sugar\rangle \text{true})$	$AGAF(EXsugar \rightarrow AFEXpour\_sugar)$
$AGAF(\langle sugar\rangle \text{true}\_\langle no\_sugar\rangle \text{true})$	$AGAF(EXsugar\_EXno\_sugar)$
$AGAF\langle ring\_a\_tone\rangle \text{true}$	$AGAF(EXring\_a\_tone)$
$AGAF\langle display\_done\rangle \text{true}$	$AGAF(EXdisplay\_done)$

由于 χ Chek 的输入公式为 CTL 公式，因此，通过定义 4.4 给出的从 ACTL 公式到 CTL 公式的转换函数 ks'，我们将这 5 个 ACTL 公式转换成相应的 CTL 公式，如表 5.6 中第 2 列所示。例如，与 ACTL 公式 $AGAF((\langle tea\rangle \text{true}\_\langle coffee\rangle \text{true}) \rightarrow AF\langle cup\_taken\rangle \text{true})$ 等价的 CTL 公式为 $AGAF((EXtea\_EXcoffee) \rightarrow AFEXcup\_taken)$；与 ACTL 公式 $AGAF\langle ring\_a\_tone\rangle \text{true}$ 等价的 CTL 公式为 $AGAF(EXring\_a\_tone)$。同时，运用工具 BPMCA 完成饮料机产品线的模型转换，并生成相应的 χ Kripke 结构和代数格的 XML 文件，以作为 χ Chek 的模型和代数格的输入文件。通过 χ Chek，我们得到这些属性在饮料机产品线上的检测结果（χ Chek 的运行界面如图 5.6 所示）。

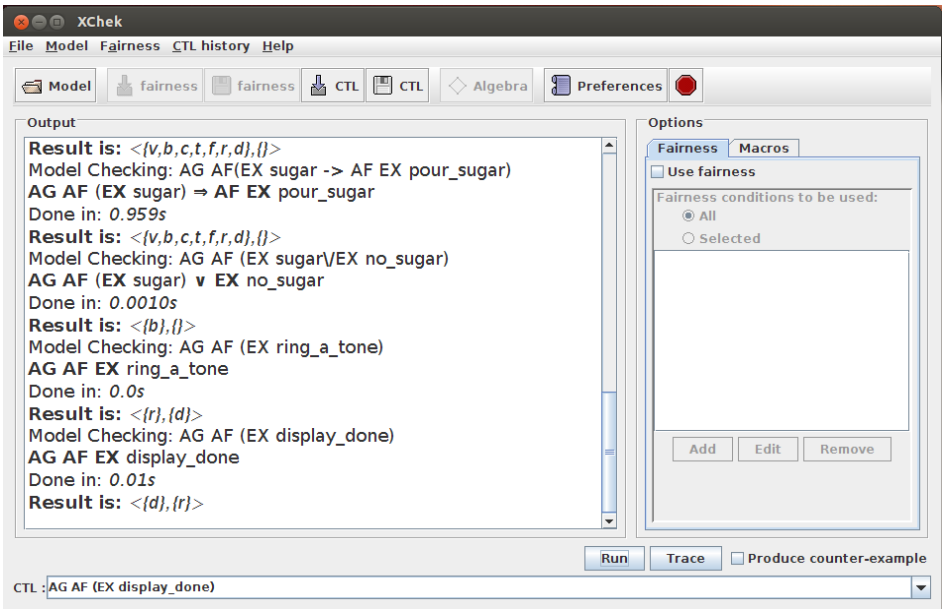


图 5.6 χ Chek 的运行界面

表 5.7 为这些公式在产品线上的检测结果，其中 W 表示饮料机产品线的特征集合。下面我们分别解释每一个公式的检测结果代表的含义：

1) $\mathrm{AGAF}((\langle tea\rangle \mathrm{true}\_\langle coffee\rangle \mathrm{true}) \rightarrow \mathrm{AF}\langle cup\_taken\rangle \mathrm{true})$：检测结果为 $\langle W,;\rangle$，表示软件产品线上的所有产品均满足该属性；

2) $\mathrm{AGAF}(\langle sugar\rangle \mathrm{true} \rightarrow \mathrm{AF}\langle pour\_sugar\rangle \mathrm{true})$：检测结果为 $\langle W,;\rangle$，表示软件产品线上的所有产品均满足该属性；

3) $\mathrm{AGAF}(\langle sugar\rangle \mathrm{true}\_\langle no\_sugar\rangle \mathrm{true})$：检测结果为 $\langle \{b\},;\rangle$，表示软件产品线上包含特征 b 的产品均满足该属性；

4) $\mathrm{AGAF}\langle ring\_a\_tone\rangle \mathrm{true}$：检测结果为 $\langle \{r\},\{d\}\rangle$，表示软件产品线上包含特征 r 且不包含特征 d 的产品均满足该属性；

5) $\mathrm{AGAF}\langle display\_done\rangle \mathrm{true}$：检测结果为 $\langle \{d\},\{r\}\rangle$，表示软件产品线上包含特征 d 且不包含特征 r 的产品均满足该属性。

表 5.7 χ Chek 上饮料机产品线的模型检测结果 $\|\varphi\|^{\mathrm{BFTS}}(s_0)$

ACTL 公式 φ	检测结果
1) $\mathrm{AGAF}((\langle tea\rangle \mathrm{true}\_\langle coffee\rangle \mathrm{true}) \rightarrow \mathrm{AF}\langle cup\_taken\rangle \mathrm{true})$	$\langle W,;\rangle$
2) $\mathrm{AGAF}(\langle sugar\rangle \mathrm{true} \rightarrow \mathrm{AF}\langle pour\_sugar\rangle \mathrm{true})$	$\langle W,;\rangle$
3) $\mathrm{AGAF}(\langle sugar\rangle \mathrm{true}\_\langle no\_sugar\rangle \mathrm{true})$	$\langle \{b\},;\rangle$
4) $\mathrm{AGAF}\langle ring\_a\_tone\rangle \mathrm{true}$	$\langle \{r\},\mathrm{f}d\mathrm{g}\rangle$
5) $\mathrm{AGAF}\langle display\_done\rangle \mathrm{true}$	$\langle \{d\},\mathrm{f}r\mathrm{g}\rangle$

在 4.2 节中，我们提出了基于模型分解的多值模型检测方法，即通过将 BFTS 相对于单个特征的模型分解，得到多个三值 BFTS，从而将多值 BFTS 上的模型检测问题分解为多个三值 BFTS 上的模型检测问题。下面，我们通过模型分解方法将饮料机产品线的 BFTS 模型相对于特征集合 W 中每个特征分解得到相应的三值 BFTS 模型 $\{M_v, M_b, M_f, M_c, M_t, M_r, M_d\}$。然后，结合 χ Chek 完成表 5.7 中的 ACTL 公式在这些三值 BFTS 模型上检验，得到的结果如表 5.8 所示；其中第 1 列中的序号代表表 5.7 中相同序号的 ACTL 公式，如序号 1)代表 ACTL 公式 $AGAF((\langle tea\rangle \text{ true} \_ \langle coffee\rangle \text{ true}) \rightarrow AF\langle cup\_taken\rangle \text{ true})$。

由表 5.8 可知，这些公式在产品线的 BFTS 上的检测结果等于它们在所有分解后三值 BFTS 模型上检测结果的并（$\oplus$）。以公式 1)为例，$AGAF((\langle tea\rangle \text{true} \_ \langle coffee\rangle \text{true}) \rightarrow AF\langle cup\_taken\rangle \text{true})$ 在产品线的 BFTS 上的检测结果为 $\langle W, ; \rangle$，在分解后三值 BFTS M_i 上的验证结果为 $\langle \{i\}, ; \rangle$ $(i = v, b, f, c, t, r, d)$，满足 $\langle W, ; \rangle = \langle \{v\}, ; \rangle \oplus \langle \{b\}, ; \rangle \oplus \langle \{f\}, ; \rangle \oplus \langle \{c\}, ; \rangle \oplus \langle \{t\}, ; \rangle \oplus \langle \{r\}, ; \rangle \oplus \langle \{d\}, ; \rangle$。

表 5.8 系统属性在 $\{M_v, M_b, M_f, M_c, M_t, M_r, M_d\}$ 上的验证结果

公式 \ 模型	M_v	M_b	M_f	M_c	M_t	M_r	M_d
1)	$\langle \{v\}, ; \rangle$	$\langle \{b\}, ; \rangle$	$\langle \{f\}, ; \rangle$	$\langle \{c\}, ; \rangle$	$\langle \{t\}, ; \rangle$	$\langle \{r\}, ; \rangle$	$\langle \{d\}, ; \rangle$
2)	$\langle \{v\}, ; \rangle$	$\langle \{b\}, ; \rangle$	$\langle \{f\}, ; \rangle$	$\langle \{c\}, ; \rangle$	$\langle \{t\}, ; \rangle$	$\langle \{r\}, ; \rangle$	$\langle \{d\}, ; \rangle$
3)	$\langle ; , ; \rangle$	$\langle \{b\}, ; \rangle$	$\langle ; , ; \rangle$	$\langle ; , ; \rangle$	$\langle ; , ; \rangle$	$\langle ; , ; \rangle$	$\langle ; , ; \rangle$
4)	$\langle ; , ; \rangle$	$\langle ; , ; \rangle$	$\langle ; , ; \rangle$	$\langle ; , ; \rangle$	$\langle ; , ; \rangle$	$\langle \{r\}, ; \rangle$	$\langle ; , \text{f}d\text{g} \rangle$
5)	$\langle ; , ; \rangle$	$\langle ; , ; \rangle$	$\langle ; , ; \rangle$	$\langle ; , ; \rangle$	$\langle ; , ; \rangle$	$\langle ; , \text{f}r\text{g} \rangle$	$\langle \{d\}, ; \rangle$

此外，我们对基于多值 BFTS 的产品线模型检测和基于三值 BFTS 的产品线模型检测进行了效率上的实验分析。在饮料机产品线例子中，根据特征图（如图 2.1 所示），产品线共包含 7 个特征，则定义在这 7 个特征上的基于世界双格有 $2^7 \times 2^7 = 4096$ 个逻辑值。表 5.9 为多值 BFTS 上的模型检测与三值 BFTS 上的模型检测对比，其中 M 为饮料机产品线的 BFTS 模型，M_i $(i = v, b, f, c, t, r, d)$ 为 M 相对与特征 i 分解的三值 BFTS 模型。实验数据显示，M 的代数格 XML 文件大小可达 14MB，而对于 M_i，其代数格 XML 文件只有 1.1KB，前者是后者的 10，000 多倍；就模型的载入时间而言，载入 M 需要 2min 以上，而载入 M_i 只需要 0.01s 左右，前者是后者的 10，000 多倍；就模型检测的内存要求，实现在 M 上的属性验证，χ Chek 要求内存 8GB，而对于 M_i 只需要 2GB，前者是后者的 4 倍。

于是，我们得出结论：采用将产品线模型相对于单个特征分解得到多个三值模型的模型检测方法能减少格代数的规模，缩短 χ Chek 加载模型所需的时间，降低对运行环境的要求，从而提高模型检测的效率。

表 5.9 多值 BFTS 上的模型检测与三值 BFTS 上的模型检测比较

模型	M_v	M_b	M_f	M_c	M_t	M_r	M_d	M
载入时间	0.018s	0.016s	0.053s	0.011s	0.032s	0.03s	0.205s	2m14.049s
代数格 XML 文件大小	1.1KB							14MB
内存要求	2GB							8GB

5.4 本章小结

本章主要结合实例设计实验以验证方法的有效性。首先介绍了多值模型检测工具 χ Chek，说明了它的输入模型和时序属性公式的要求以及输入文件格式；其次介绍了我们开发的辅助工具 BPMCA，包括总体框架设计、类的设计以及相关算法；最后以饮料机产品线为例，设计相关属性，完成了产品线上的模型检测，并通过实验对比分析了两种实现多值模型检测的方法。

第六章 总结及展望

本章首先总结全文的研究内容，然后讨论研究内容的局限性以及未来的研究工作。

6.1 论文总结

本文主要针对软件产品线初期存在的信息不确定情况，研究了软件产品线的不完备建模与验证。

首先，本文提出了 BFTS 以支持在软件产品线设计初期特征与系统行为之间的不确定情况。在第二章中，本文介绍了德摩根代数和世界双格。不同于德摩根代数，世界双格不仅定义了逻辑值之间的真值序关系以反映信息的真假程度，同时也定义了逻辑值之间的信息序关系以描述信息的精确程度，从而从实际角度帮助我们理解多值逻辑。BFTS 是一种基于世界双格的多值模型，本文采用世界双格中的逻辑值来描述产品线中特征与行为之间的关系：需要(require)，排斥(forbid)和不确定(unknown)，并通过定义标签函数实现迁移到世界双格上的映射。设W 为软件产品线的特征集合，若迁移 (s,a,s') 的逻辑值为$\langle U,V\rangle$，则表示迁移 (s,a,s') 被U 中的特征需要并且被V 中的特征排斥，而$W/(U\lceil V)$ 中的特征与迁移 (s,a,s') 关系不确定。在第三章中，除了定义 BFTS，本文还研究了软件产品线上特定产品的行为模型。我们采用了特征建模中对产品的定义，即符合特征约束关系的一组特征的集合。由于软件产品线中存在信息不确定情况，其行为模型中存在迁移与特征之间的关系不确定，本文首先通过投影定义产品的不完备模型，与软件产品线的行为模型相比，该模型中只包含产品中一定存在以及可能存在的系统行为，去除了产品中特征排斥的行为；然后通过精化关系定义产品的具体模型，不同于产品的不完备模型，该模型中只包含产品中一定存在的系统行为。

然后，本文采用 ACTL 描述系统的时序属性。由于传统的 ACTL 语义是定义在 LTS 上，对多值模型并不适用。在第四章中，本文给出了 ACTL 在 BFTS 上的语义，提出基于双格的多值模型检测。进一步，为实现软件产品线的多值模型检测，一方面，本文提出从 BFTS 到 χ Kripke 结构的转换方法，并开发了辅助工具 BPMCA 以实现模型之间的自动转换，从而使得能够结合现有的多值模型检测工具 χ Chek 实现软件产品线的多值模型检测；另一方面，本文提出针对单个特征的分解方法，从而将多值 BFTS 相对于单个特征分解为多个三值 BFTS，实现将多值模型检测问题分解为多个三值模型检测问题，降低了模型检测的复杂度。

最后，本文在第五章对文献中的饮料机产品线实例进行实验分析，通过实验数据验证了：1)基于 BFTS 的多值模型检测方法的有效性； 2)通过将多值模型问题分解为多个三值模型检测问题，可以降低模型检测的复杂度，即缩短了模型载入时间，降低了内存环境要求等。

6.2 未来工作展望

首先，与传统模型检测类似，多值模型检测中也存在状态爆炸问题，这使得多值模型检测的复杂度增加。然而，这一问题在本文的研究内容中并未涉及。随着软件产品线的发展，软件产品线的规模逐步增大，基于软件产品线的多值模型检测中状态爆炸问题不可忽视。在后续工作中，我们将考虑通过多值模型的抽象方法以减小状态空间。

其次，在本文的研究工作中并没有采用任何高级建模语言去描述软件产品线，后续将会考虑结合已有的针对软件产品线的建模语言 fPromela[22]以及相应的软件产品线模型检测工具 SNIP[22]解决这一局限性。

最后，本文的研究中并未考虑到特征组合对系统行为的影响，如当选择了特征 f_1 和 f_2，需要系统包含行为 (s,a,s')。在后续工作中，我们将考虑采用多值逻辑表达式代替多值逻辑值的方式标记系统行为，从而能够描述特征组合与系统行为之间的关系。

参考文献

[1] Clements P, Northrop L. Software product lines: practices and patterns. Massachusetts: Addison-Wesley, 2002.

[2] van der Linden F J, Schmid K, Rommes E. Software product lines in action. Berlin Heidelberg: Springer-Verlag, 2007.

[3] Schmid K, Rabiser R, Grünbacher P. A comparison of decision modeling approaches in product lines Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems. ACM, 2011: 119-126.

[4] van der Linden F J, Pohl K. Software Product Line Engineering: Foundations, Principles, and Techniques. 2005.

[5] Kang K C, Cohen S G, Hess J A, et al. Feature-oriented domain analysis (FODA) feasibility study. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1990.

[6] Classen A, Heymans P, Schobbens P Y. What's in a feature: A requirements engineering perspective Fundamental Approaches to Software Engineering. Berlin Heidelberg: Springer, 2008: 16-30.

[7] Baier C, Katoen J P. Principles of model checking. Cambridge: MIT press, 2008.

[8] Holzmann G J. The model checker SPIN. IEEE Transactions on software engineering, 1997, 23(5): 279-295.

[9] McMillan K L. Symbolic model checking. US: Springer, 1993.

[10] De Moura L, Bjørner N. Z3: An efficient SMT solver Tools and Algorithms for the Construction and Analysis of Systems. Berlin Heidelberg: Springer, 2008: 337-340.

[11] Salay R, Chechik M, Horkoff J, et al. Managing requirements uncertainty with partial models. Requirements Engineering, 2013, 18(2): 107-128.

[12] Niu N, Savolainen J, Yu Y. Variability modeling for product line viewpoints integration Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual. IEEE, 2010: 337-346.

[13] Chechik M, Devereux B, Easterbrook S, et al. Multi-valued symbolic model-checking. ACM Transactions on Software Engineering and Methodology (TOSEM), 2003, 12(4): 371-408.

[14] Famelis M, Salay R, Chechik M. Partial models: Towards modeling and reasoning with

uncertainty Software Engineering (ICSE), 2012 34th International Conference on. IEEE, 2012: 573-583.

[15] Bruns G, Godefroid P. Model checking partial state spaces with 3-valued temporal logics Computer Aided Verification. Berlin Heidelberg: Springer, 1999: 274-287.

[16] Fitting M. Bilattices in logic programming: Multiple-Valued Logic, 1990, Proceedings of the Twentieth International Symposium on. IEEE, 1990: 238-246.

[17] Ginsberg M L. Multivalued logics. A uniform approach to reasoning in artificial intelligence. Computational intelligence, 1988, 4(3): 265-316.

[18] Mosser S, Parra C, Duchien L, et al. Using domain features to handle feature interactions. Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems. ACM, 2012: 101-110.

[19] 邹盛享, 张伟, 赵海燕, 等. 面向软件产品家族的变化性建模方法. 软件学报, 2005, 16(1): 37-49.

[20] Classen A, Heymans P, Schobbens P Y, et al. Model checking lots of systems: efficient verification of temporal properties in software product lines. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1. ACM, 2010: 335-344.

[21] Classen A, Heymans P, Schobbens P Y, et al. Symbolic model checking of software product lines. Proceedings of the 33rd International Conference on Software Engineering. ACM, 2011: 321-330.

[22] Classen A, Cordy M, Heymans P, et al. Model checking software product lines with SNIP. International Journal on Software Tools for Technology Transfer, 2012, 14(5): 589-612.

[23] Cordy M, Classen A, Heymans P, et al. ProVeLines: a product line of verifiers for software product lines. Proceedings of the 17th International Software Product Line Conference co-located workshops. ACM, 2013: 141-146.

[24] 聂坤明, 张莉, 樊志强. 软件产品线可变性建模技术系统综述. 软件学报, 2013, 24(9).

[25] Schobbens P, Heymans P, Trigaux J C. Feature diagrams: A survey and a formal semantics. Requirements Engineering, 14th IEEE international conference. IEEE, 2006: 139-148.

[26] Hubaux A, Heymans P, Schobbens P Y, et al. Supporting multiple perspectives in feature-based configuration. Software & Systems Modeling, 2013, 12(3): 641-663.

[27] Apel S, Batory D, Kästner C, et al. Feature-Oriented Software Product Lines. Berlin: Springer, 2013.

[28] Fischbein D, Uchitel S, Braberman V. A foundation for behavioural conformance in software product line architectures. Proceedings of the ISSTA 2006 workshop on Role of software architecture for testing and analysis. ACM, 2006: 39-48.

[29] Fantechi A, Gnesi S. Formal modeling for product families engineering. Software Product Line Conference, 2008. SPLC'08. 12th International. IEEE, 2008: 193-202.

[30] Bauer S S, Juhl L, Larsen K G, et al. Extending modal transition systems with structured labels. Mathematical Structures in Computer Science, 2012, 22(04): 581-617.

[31] Ziadi T, Hélouët L, Jézéquel J M. Towards a UML profile for software product lines. Software Product-Family Engineering. Berlin Heidelberg: Springer, 2004: 129-139.

[32] Czarnecki K, Antkiewicz M. Mapping features to models: A template approach based on superimposed variants. Generative Programming and Component Engineering. Berlin Heidelberg: Springer, 2005: 422-437.

[33] Larsen K G, Nyman U, Wąsowski A. Modal I/O automata for interface and product line theories. Programming Languages and Systems. Berlin Heidelberg: Springer, 2007: 64-79.

[34] Gruler A, Leucker M, Scheidemann K. Modeling and model checking software product lines. Formal Methods for Open Object-Based Distributed Systems. Berlin Heidelberg: Springer, 2008: 113-131.

[35] Asirelli P, Ter Beek M H, Gnesi S, et al. Formal Description of Variability in Product Families. SPLC. 2011, 11: 130-139.

[36] ter Beek M H, Mazzanti F, Sulova A. VMC: A tool for product variability analysis. FM 2012: Formal Methods. Berlin Heidelberg: Springer, 2012: 450-454.

[37] Seger C J H, Bryant R E. Formal verification by symbolic evaluation of partially-ordered trajectories. Formal Methods in System Design, 1995, 6(2): 147-189.

[38] Gurfinkel A, Wei O, Chechik M. Yasm: A software model-checker for verification and refutation. Computer Aided Verification. Berlin Heidelberg: Springer, 2006: 170-174.

[39] Chechik M, Easterbrook S, Petrovykh V. Model-checking over multi-valued logics. FME 2001: Formal Methods for Increasing Software Productivity. Berlin Heidelberg: Springer, 2001: 72-98.

[40] Chechik M, Gurfinkel A, Devereux B. χChek: a multi-valued model-checker. Computer Aided Verification. Berlin Heidelberg: Springer, 2002: 505-509.

[41] Salay R, Famelis M, Chechik M. Language independent refinement using partial modeling. Fundamental Approaches to Software Engineering. Berlin Heidelberg: Springer, 2012: 224-239.

[42] De Nicola R, Vaandrager F. Action versus state based logics for transition systems. Semantics of

Systems of Concurrent Processes. Berlin Heidelberg: Springer, 1990: 407-419.

[43] Uchitel S, Kramer J, Magee J. Detecting implied scenarios in message sequence chart specifications[J]. ACM SIGSOFT Software Engineering Notes, 2001, 26(5): 74-82.

[44] Clarke E M, Emerson E A. Design and synthesis of synchronization skeletons using branching time temporal logic. Berlin Heidelberg: Springer, 1982.

[45] Baier C, Katoen J P. Principles of model checking. Cambridge: MIT press, 2008.

[46] Kauffman L H. De Morgan Algebras-completeness and recursion. Proceedings of the eighth international symposium on Multiple-valued logic. IEEE Computer Society Press, 1978: 82-86.

[47] Meller Y, Grumberg O, Shoham S. A framework for compositional verification of multi-valued systems via abstraction-refinement. Automated Technology for Verification and Analysis. Berlin Heidelberg: Springer, 2009: 271-288.

[48] Kripke S A. Semantical considerations on modal logic. 1963.

[49] Chechik M, Devereux B, Easterbrook S, et al. Multi-valued symbolic model-checking. ACM Transactions on Software Engineering and Methodology (TOSEM), 2003, 12(4): 371-408.

[50] Pnueli A. The temporal logic of programs. Foundations of Computer Science, 1977, 18th Annual Symposium on. IEEE, 1977: 46-57.

[51] Bruns G, Godefroid P. Model checking partial state spaces with 3-valued temporal logics. Computer Aided Verification. Berlin Heidelberg: Springer, 1999: 274-287.

[52] Godefroid P, Jagadeesan R. On the expressiveness of 3-valued models. Verification, Model Checking, and Abstract Interpretation. Berlin Heidelberg: Springer, 2003: 206-222.

致 谢

时光飞逝，转眼间我在南京航空航天大学三年的研究生学习生活即将结束，随之而来的是我人生历程的又一个新的开始。当我写完《基于多值逻辑的软件产品线模型检测》这篇论文时，内心既充满着感激，又有不安。

首先，我要感谢我的研究生导师魏欧老师。魏老师待人宽和、治学严谨，在一宽一严中体现着恩师人格之魅力。论文从开始的背景知识积累、选题，到论文的写作与修改定稿，这其中的每一个环节，都得到了魏老师的精心指导。在我读研期间，无论遇到什么问题或困难，魏老师都给我提供了热情的帮助，在魏老师的指导下，我从一个科研工作的门外汉到现在可以完成研究生期间的各项学习任务，并享受科研的乐趣。在此，谨向魏老师表示最崇高的敬意！

其次，我要感谢南京航空航天大学计算机学院的诸位领导和师友。是他们的悉心关怀和无私帮助，让我完成了我的学业。陈娟娟师姐就像是我在读研期间的第二个导师，当我初来学校，对这里的学习生活还很陌生时，是陈娟娟师姐给我提供了各种各样的帮助。在学习模型检测相关知识时，师姐总是给我耐心地讲解我遇到的各种问题，并给我提供很多有益的建议和思路。黄明宇师弟的软件开发能力挺强，在我实验过程中遇到阻碍时，他总能帮助我想到解决的办法。刘玉梅师妹学习认真，两年里我们经常讨论交流，这些都让我受益匪浅。此外，我还要感谢同届的其他同学。无论生活上还是学习上，他们总是给我各种支持，帮助我克服困难。有他们一路陪伴，我丝毫不感到孤单。

最后，我要感谢我的父母，在我的成长中，他们始终给我最无私、最贴心的关怀。他们的支持是我在外拼搏的支柱，让他们开心是我一直不变的追求。

学海无涯，面对知识殿堂，我自感渺小，尽管经过了我的努力，但论文中定有不足之处，这让我内心有些不安，恳请老师们给以批评与帮助，让我不断使之完善，我将不胜感激！

在学期间的研究成果及发表的学术论文

攻读硕士学位期间发表（录用）论文情况

1. 石玉峰，魏欧，周宇. 基于双格的软件产品线模型检测.计算机科学.2015,42(2):167-172
2. Yufeng Shi，Ou Wei，Yu Zhou. Model Checking Partial Software Product Line Designs. 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering: International Workshop on Innovative Software Development Methodologies and Practices. 2014: 21-29
3. 魏欧，石玉峰，徐丙凤等. 软件模型检测中的抽象模型研究综述. 计算机研究与发展.（已录用）
4. 黄鸣宇，石玉峰. 基于 χ Chek 的软件产品线多值模型检测方法. 计算机与现代化. 2014(8)

攻读硕士学位期间参加科研项目的情况

1. 2012 年 10 月至 2014 年 12 月，参与国家自然科学基金项目：基于抽象的软件符号模型检测研究（项目号：61170043），负责相关算法设计。