计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20140413
52(7):1580-1603,2015

软件模型检测中的抽象模型研究综述

魏 欧[1] 石玉峰[1] 徐丙凤[1,2] 黄志球[1] 陈 哲[1]
[1](南京航空航天大学计算机科学与技术学院 南京 210016)
[2](南京林业大学信息科学技术学院 南京 210037)
(owei@nuaa.edu.cn)

Abstract Modeling Formalisms in Software Model Checking

Wei Ou[1], Shi Yufeng[1], Xu Bingfeng[1,2], Huang Zhiqiu[1], and Chen Zhe[1]
[1](*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing* 210016)
[2](*College of Information Science and Technology, Nanjing Forestry University, Nanjing* 210037)

Abstract Abstraction is a fundamental technique for solving the state-explosion problem in software model checking. In this paper, we survey a variety of abstract modeling formalisms that have been developed for this over the years. We first provide an overview of abstract software model checking based on the theoretical framework of abstract interpretation. We then discuss in detail several abstract modeling formalisms that are represented by 1) boolean Kripke structures, supporting traditional over-approximation or under-approximation; 2) Kripke modal transition systems and mixed transition systems, respectively corresponding to 3-valued and 4-valued Kripke structures, supporting both over-approximation and under-approximation on a single model; and 3) models with hyper transitions, including generalized Kripke modal transition systems and hyper transition systems, allowing for more precise abstract model checking. We discuss the corresponding approximation relations and optimal abstract models, and highlight their shortcomings and the motivations for the development of new formalisms. We also introduce the completeness results of abstract modeling formalisms. Finally, we discuss the problems in theoretical and practical aspects of abstract models and point out future research directions.

Key words abstract models; over-approximation; under-approximation; model checking; multi-valued models

摘 要 抽象是解决模型检测中状态爆炸问题的一个基本方法.对近年来软件模型检测研究中所提出的一系列抽象模型进行综述.首先以抽象解释为理论框架阐述了抽象软件模型检测的各组成部分.然后根据模型的结构和功能特征,将抽象模型分为3类:1)传统的用于支持自上逼近或者自下逼近的布尔Kripke结构;2)分别对应于3值和4值Kripke结构的Kripke模态迁移系统(Kripke modal transition systems, KMTS)和混合迁移系统(mixed transition system, MixTS),可同时支持自上逼近和自下逼近的抽象;3)具有超迁移关系的广义Kripke模态迁移系统(generalized Kripke modal transition system, GKMTS)和超迁移系统(hyper transition system, HTS),可提供更精确的抽象模型检测;重点分析这些模型的提出原因、相应的逼近关系、最优模型及其局限性以及抽象模型完备性的研究结果.最后,分析了目前关于抽象模型的理论和应用研究中存在的问题,给出进一步研究的方向.

收稿日期:2014-05-09;**修回日期**:2014-11-20
基金项目:国家自然科学基金项目(61170043,61100034,61272083);国家"九七三"重点基础研究发展计划基金项目(2014CB744904)

关键词　抽象模型;自上逼近;自下逼近;模型检测;多值模型

中图法分类号　TP311

模型检测[1-3]是一种自动形式化验证技术,用于对一个计算机系统的正确行为属性进行判断.模型检测的基本方法是用一个状态迁移图 M 来表示所要检测的系统的模型,并用时序逻辑(如计算树逻辑(computation tree logic, CTL)、线性时序逻辑(linear temporal logic, LTL)、μ 演算等)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穷举搜索来判断该公式是否能够在模型上被满足.如果公式在模型上满足,即 $M\models\varphi$,则系统的正确性得到证实(verified);否则,就表明系统中存在错误,即 $M\models\neg\varphi$,系统正确性被证伪(falsified).与测试和形式化推理等其他验证技术比较起来,模型检测的优点在于它是自动的,能够检查系统模型的所有行为,并且可以通过反例帮助理解错误产生的原因.

模型检测在实际中应用的主要瓶颈是状态爆炸问题(state-explosion problem).由于模型检测基于穷举搜索对正确属性进行判断,所以它适用于对有限的、状态空间较小的系统进行分析.然而,在实际应用中经常存在着状态空间庞大的系统.例如,通信协议等并发系统中的状态空间往往随着并发分量的增加呈指数增长.对这类系统进行模型检测需要耗费大量的存储和计算资源.而对于软件系统,由于存在着无限数据域(如整数)、无界数据类型(如链表)、以及复杂的控制结构(如递归),导致软件系统的状态空间可以是无限大,直接对它们进行模型检测在实际中是不可行的.因此,正如图灵奖得主 Clarke 所说,解决状态爆炸问题是模型检测研究中的一个最根本的工作[4].

抽象是解决状态爆炸问题的一个重要的方法.它的基本思想是首先构造一个比原系统的具体模型小的有限抽象模型,然后通过正确属性在抽象模型上的检测结果推测出其在原来的具体模型上是否可满足.抽象方法的依据是,对于所给定的某种正确属性而言,待检测的原系统的许多信息(如某些程序变量的取值、进程的标识符、调用栈中活动记录等)是无关的.因此,这些信息可以从具体模型中抽象出去.这样不仅简化了模型,同时保留了必要的信息,使得抽象的模型检测得以有效地进行.抽象方法近年来引起越来越多的重视,特别是对于可能具有无限状态空间的软件系统,采用抽象方法对其进行模型检测是一个必不可少的措施.在以微软研究院、伯克利大学、卡内基梅隆大学、牛津大学、NASA Ames 研究中心、NEC 美国实验室等为代表的研究单位所研发的诸如 SLAM[5-6], BL-AST[7-8], SATBAS[9-10], JAVA PATHFINDER[11-12], F-SOFT[13] 等软件模型检测器都建立在对软件系统的抽象模型进行分析的基础上.而这些成功的例子也进一步表明抽象方法在扩展模型检测在实际中的应用范围以及克服状态爆炸问题上的有效性.

模型检测中的抽象研究是一个很广泛的领域.现有的一些综述文献从不同的方面对其进行了分析,如模型构建[14]和反例精化[15]等,而对抽象模型本身的关注还较少.抽象模型是抽象方法的研究基础.近年来,抽象分析的研究工作明确或者隐含地提出和采用了各种各样的抽象模型.这些模型具有不同的形式,如模态迁移系统、多值模型、博弈结构等,并且面向不同的应用目的,如验证正确性和检测错误等.这种多样性造成了对于各抽象模型的结构特征、功能特征、局限性以及不同模型之间差别的理解上的困难.

为从本质上深入理解现有研究成果的不同之处、发展过程和应用背景,本文对软件模型检测中的抽象模型进行系统地综述分析:首先采用抽象解释[16-18]的基本理论为分析框架,对模型检测中抽象方法的各组成部分进行阐述;然后重点对目前的各种抽象模型进行分析.抽象解释最早是由 Cousot 等人[16]提出来用于描述静态程序分析的一套理论和技术方法.本文将其基本理论中的标准的符号和分析机制应用于对模型检测中的抽象方法的分析.在这个基本框架内,对于目前的各种抽象模型,本文根据它们不同的结构和功能特征,提出将其分为 3 种主要类型:1)传统的支持证实的自上逼近方法以及支持证伪的自下逼近方法中所使用的布尔 Kripke 结构;2)分别对应于 3 值和 4 值 Kripke 结构的 Kripke 模态迁移系统(Kripke modal transition systems, KMTS)和混合迁移系统(mixed transition system, MixTS),可同时支持自上逼近和自下逼近的抽象;3)具有超迁移关系的广义 Kripke 模态迁移系统(generalized Kripke modal transition system, GKMTS)和超迁移系统(hyper transition system, HTS)可提供更精确的抽象模型检测.对于每一类模型,进一步分析对应的逼近关系、最优模型的结构定义条件及

其功能上的局限性,以及抽象模型的完备性研究结果,突出新的抽象模型产生的意义.在此基础上,本文最后分析了目前关于抽象模型的理论和应用研究中存在的问题,提出进一步研究的方向.

1 理论基础

首先介绍多值逻辑和相应的作为计算模型的多值 Kripke 结构,以及时序逻辑——命题 μ 演算(propositional μ-calculus)——在多值 Kripke 结构上的语义.

1.1 真值域与多值逻辑

真值域 $\mathcal{D}$ 由表示真值的元素 D、真值序关系 $\sqsubseteq$ 和求补运算符 $\neg: D\to D$ 组成,使得 $\mathcal{D}=(D,\sqsubseteq,\neg)$ 是德摩根代数.所谓真值序关系(truth ordering)即根据元素为真的程度所定义的元素之间的序关系,例如:$a\sqsubseteq b$ 表示"a 比 b 的真值小".

目前使用最广泛的经典布尔(bool)逻辑,即 2 值逻辑,值域只含有 t(真)与 f(假),并且 f$\sqsubseteq$t(如图 1(a)所示).而 3 值逻辑,如 Kleene 逻辑(如图 1(b)所示),扩展了 2 值逻辑,增加了一个元素 $\perp$ 用于表示未知信息.3 值逻辑的真值序关系扩展为 f$\sqsubseteq\perp$ 和 $\perp\sqsubseteq$t,其求补运算扩展为 $\neg\perp=\perp$.此外,3 值逻辑根据信息量的多少定义了一种信息序关系(information ordering),有 $\perp\leq$t 且 $\perp\leq$f,因此,$\perp$ 能够表示的信息量最少.

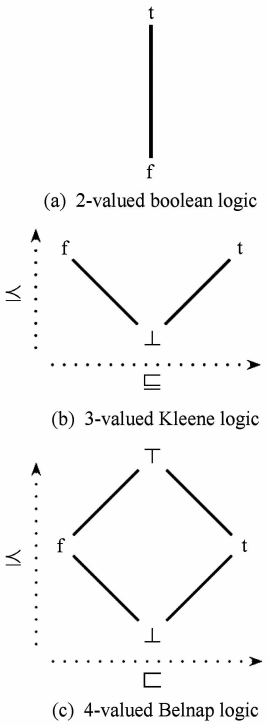


Fig. 1 Truth domains.

图 1 真值域

4 值逻辑,如 Belnap 逻辑(如图 1(c)所示),在 3 值逻辑的基础上扩展了一个元素 $\top$.扩展后的真值序为 f$\sqsubseteq\top$ 和 $\top\sqsubseteq$t,求补运算扩展为 $\neg\top=\top$.最后,通过使 $\top$ 成为信息量最大的元素来扩展信息排序,即 f$\leq\top$ 且 t$\leq\top$.这使得 4 值逻辑成为包含 2 值逻辑,并且在真值序关系和信息序关系下均构成完全分配格的最小结构,即双格(bilattice)[19].

1.2 计算模型

我们定义作为计算模型的多值 Kripke 结构[20-21],其本质上是状态转换系统(state transi-tion system),但是采用多值逻辑来定义状态迁移关系和对原子命题的状态标注.

定义 1. 定义在一组原子命题公式 AP 上的 $\mathcal{D}$ 值 Kripke 结构是一个四元组 $K=\langle S,\mathcal{D},R,I\rangle$,其中:

1) S 是状态集合;

2) $\mathcal{D}\in\{2,3,4\}$ 是逻辑;

3) $R: S\times S\to\mathcal{D}$ 是转换关系;

4) $I: AP\to[S\to\mathcal{D}]$ 是一个原子命题公式的解释,它对每一个原子命题建立了一个从状态到 $\mathcal{D}$ 的映射.

多值 Kripke 结构扩展了经典 2 值布尔 Kripke 结构(2-valued Kripke structure, 2VKS),能够表示由于抽象所造成的未知和不确定信息.为方便起见,我们省略其中表示真值域的逻辑符号,记 2VKS 为 $K=\langle S,R,I\rangle$.

对于状态转换关系 $R: S\times S\to\mathcal{D}$,状态集 $Q: S\to\mathcal{D}$ 相对于 R 的前像(pre-image)和后像(post-image)定义为:

$$pre[R](Q)\triangleq\lambda s\in S\cdot\bigvee_{t\in S}R(s,t)\wedge Q(t);$$

$$post[R](Q)\triangleq\lambda t\in S\cdot\bigvee_{s\in S}R(s,t)\wedge Q(s).$$

根据定义可以看出,$pre[R](Q)$ 是通过 R 可以到达 Q 的状态集;而 $post[R](Q)$ 则是通过 R 可以从 Q 到达的状态集.前像的对偶操作定义为

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

也就是说,$\tilde{pre}[R](Q)$ 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.

通常将 Kripke 结构中的某个状态 s_0 作为初始状态.该结构中一条路径(或者计算序列)是一个形如 $\pi=s_0,s_1,\cdots$ 的无穷序列.在某些情况下,我们只关心沿公平(fair)计算路径的正确性.例如:在互斥协议中,仅考虑每个进程都能够被无数次调度执行

的计算路径.在这种情形下,我们使用公平性约束来限制所期望的路径.我们说一条路径 π 是公平的当且仅当满足公平性约束的状态在 π 中能够无穷多次发生.

1.3　时序逻辑

介绍命题 μ 演算逻辑(L_μ)及其在多值 Kripke 结构上的语义[22].在实际中,常常使用 CTL 或 LTL 描述系统的时序属性.CTL 和 LTL 作为 μ 演算的子集,其对应的公式都可转换为 μ 演算的公式[1].L_μ 的优势在于它所表示的时序逻辑属性与不动点特征是等价的,因此可以使用有效的算法来对其进行检测,如符号模型检测.

定义 2. 设 Var 为变量集合,AP 为原子命题集合.命题 μ 演算 $L_\mu(AP)$ 是如下定义的一组公式:

$$\varphi ::= Z \mid p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond\varphi \mid \mu Z \cdot \varphi(Z),$$

其中 $p \in AP$,$Z \in Var$,$\varphi(Z)$ 在 Z 上语法单调.

简便起见,当原子命题集 AP 在上下文中明确时,记这种逻辑为 L_μ.另外,定义如下的缩写形式:

$$\varphi \vee \psi \triangleq \neg(\neg\varphi \wedge \neg\psi),$$

$$\varphi \Rightarrow \psi \triangleq \neg\varphi \vee \psi,$$

$$\Box\varphi \triangleq \neg\Diamond\neg\varphi,$$

$$\nu Z \cdot \varphi(Z) \triangleq \neg\mu Z \cdot \neg\varphi(\neg Z).$$

模态算子◇通常被称为存在路径量词,而□则被称为全称路径量词.例如:"p"表示在当前状态上 p 是可满足的,"$\Diamond p$"表示在接下来的某个状态上 p 将被满足,而"$\Box p$"表示 p 在接下来所有可能的状态上 p 都被满足.量词 μ 和 ν 分别代表最小不动点(least fixpoint)和最大不动点(greatest fixpoint).

如果公式 φ 中变量 Z 出现在不动点量词的辖域内则称变量 Z 为约束变量;否则称 Z 为自由变量.例如,在公式 $p \vee \Diamond Z$ 中 Z 是自由变量;在 $\mu Z \cdot p \vee \Diamond Z$ 中 Z 是约束变量.当公式 φ 不包含任何自由变量时,则称之为闭合公式.

给定 AP 上的 $\mathcal{D}$ 值 Kripke 结构 $K=\langle S, \mathcal{D}, R, I\rangle$,$L_\mu$ 公式 φ 在 K 上的语义由函数 $\|\cdot\|_\sigma$ 定义 $\|\cdot\|_\sigma$ 对每一个状态 $s \in S$ 赋予 φ 在 s 上的值,其中函数 $\|\cdot\|_\sigma$ 通过对公式结构的归纳定义如下:

$$\|p\|_\sigma(s) \triangleq I(p)(s),$$

$$\|Z\|_\sigma(s) \triangleq \sigma(Z)(s),$$

$$\|\varphi \wedge \psi\|_\sigma(s) \triangleq \|\varphi\|_\sigma(s) \wedge \|\psi\|_\sigma(s),$$

$$\|\neg\varphi\|_\sigma(s) \triangleq \neg\|\varphi\|_\sigma(s),$$

$$\|\mu x \cdot \varphi\|_\sigma(s) \triangleq lfp^{\sqsubseteq}(\lambda Q \cdot \|\varphi\|_{\sigma[x \mapsto Q]})(s),$$

$$\|\Diamond\varphi\|_\sigma(s) \triangleq pre[R](\|\varphi\|_\sigma)(s).$$

其中,$\sigma: Var \to [S \to \mathcal{D}]$ 是自由变量的环境,而 $lfp^{\sqsubseteq}(f)$ 是 f 相对于序关系 $\sqsubseteq$ 的最小不动点.K 上闭合公式 φ 的值由初始状态上 φ 的值 $\|\varphi\|(s_0)$ 给出,表示为 $\|\varphi\|^K$.

2　软件模型检测抽象方法概述

在静态程序分析中,为了能够基于部分信息对程序的属性进行分析,Cousot[16] 等人提出了抽象解释理论,用于对程序的近似语义(semantics approximation)进行计算.为此,抽象解释严格定义了具体程序语义与抽象形式语义之间的关系、正确性与精确性等不依赖于特定应用的抽象数学概念,并且提供了有效的用于对函数和不动点等复杂对象进行抽象构造的方法.软件的抽象模型检测可以看作是一种特定的静态程序分析,因此我们可以在抽象解释的框架内对其进行研究.本节首先对抽象解释的基本概念和结论进行回顾,然后对软件模型检测中的抽象方法(即基于状态转换系统对程序的抽象解释)的框架进行介绍.

2.1　抽象解释

2.1.1　具体域、抽象域以及正确性(concrete domain, abstract domain and soundness)

进行抽象解释首先需要确定所要分析的具体对象集合(如:程序状态或者是转换关系),称之为具体域 $\mathcal{C}$.然后定义一个用于对具体域 $\mathcal{C}$ 中的对象进行逼近的抽象域 $\mathcal{A}$.逼近关系的正确性由正确性关系(soundness relation)$\rho \subseteq \mathcal{C} \times \mathcal{A}$ 进行刻画,其中 $(c,a) \in \rho$ 表示 c 被 a 逼近,记作:$a \leq_\rho c$.例如:若将整数 $\mathbb{Z}$ 作为具体域,集合 $A=\{\text{even}, \text{odd}, \text{int}, \text{none}\}$ 作为抽象域,其中 none 对应于同时为奇数和偶数的不一致情况.可以基于奇偶性定义正确性关系 $\rho \subseteq \mathbb{Z} \times A$;在这种情况下,2 可以同时被 even(偶数)和 int(整数)逼近.

2.1.2　抽象与精度(abstraction and precision)

从 2.1.1 节例子可以看出,一个具体对象可以同时被几个抽象对象逼近.为了比较不同逼近之间的相对精确性,分别为 $\mathcal{C}$ 和 $\mathcal{A}$ 引入信息偏序(information ordering)$\leq_{\mathcal{C}}$ 和 $\leq_{\mathcal{A}}$,使得 $(\mathcal{C}, \leq_{\mathcal{C}})$ 与 $(\mathcal{A}, \leq_{\mathcal{A}})$ 为偏序集合.

对于抽象域的对象来说,$a_1 \leq_{\mathcal{A}} a_2$ 表示 a_1 比 a_2 能够逼近更多的具体元素.因此,a_1 所能够精确表示的信息量比 a_2 少,也就是说,a_1 的精度比 a_2 低.而将 a_1 和 a_2 作为描述具体对象的属性时,则 a_1 表示的属性比 a_2 要弱.例如:图 2 表示了上述奇偶性例子中 A 上的信息偏序.直观上而言,知道一个对象

是整数(int)比知道这个对象是偶数(even)或奇数(odd)所能提供的信息含量要少. 而对具体域的对象来说,$c_1 \leq_C c_2$ 则表示 c_2 比 c_1 能够被更精确的属性逼近. 在实际中,具体域或者抽象域可以定义在无序集上,如整数集. 在这种情形下,一般考虑将具体域扩展为该集合的幂集. 例如:在奇偶性例子中可使用幂集 $2^{\mathbf{Z}}$ 作为具体域并且根据集合的包含关系来定义信息偏序.

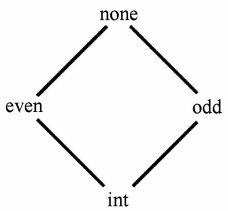


Fig. 2 Parity abstract domain.

图 2 奇偶抽象域

有了表示抽象对象精度的符号 $\leq_A$,可使用抽象函数(abstraction function)来定义最精确(也称为最好或者最优)逼近. 对抽象函数 $\alpha: C \to A$ 的定义如下:

$$\forall c \in C \cdot \forall a \in A \cdot a \leq_\rho c \Leftrightarrow a \leq_A \alpha(c),$$

也就是说,对于任何具体对象 c,$\alpha(c)$ 是 c 的最精确逼近,使得对于 A 中任何其他的对象 a,如果 a 逼近 c,则 a 的逼近精度总是小于或者等于 $\alpha(c)$. 例如:在奇偶性例子中, 2 在 A 中的抽象是 even.

抽象过程的对偶被称为具体化,通过具体函数(concretization function)$\gamma: A \to C$ 对其定义:

$$\gamma(a) = \{c \in C \mid \alpha(c) \leq_A a\},$$

也就是说,$\gamma(a)$ 表示所有能够被 a 逼近的具体对象. 例如: γ(even)是$\mathbb{Z}$ 中所有偶数的集合.

2.1.3 基于 Galois 连接的抽象(Galois connection-based abstraction)

α 和 γ 通常构成 C 和 A 之间的 Galois 连接.

定义 3. 设$(C, \leq_C)$和$(A, \leq_A)$为 2 个偏序集. 映射序对$\langle \alpha: C \to A, \gamma: A \to C \rangle$是一个从 C 到 A 的 Galois 连接当且仅当以下条件满足:

$$\forall c \in C \cdot \forall a \in A \cdot \gamma(a) \leq_C c \Leftrightarrow a \leq_A \alpha(c).$$

Galois 连接提供了对抽象过程的一种很自然的描述. 根据定义可以看出,α 和 γ 是单调的,并且同时满足 $\gamma \circ \alpha(c) \leq_C c$ 以及 $a \leq_A \alpha \circ \gamma(a)$,这说明抽象化($\alpha$)和具体化($\gamma$)过程能够保持具体对象和抽象对象的精度,并且是一致的.

采用正确性关系 ρ 定义的逼近与基于 Galois 连接的抽象之间存在着紧密联系. Loiseaux 等人[23]证明从对具体域和抽象域的描述方面而言,这两者是等价的.

定理 1[23]. 1)对于关系 $\rho \subseteq Q_1 \times Q_2$,映射序对$\langle post[\rho], p\tilde{r}e[\rho] \rangle$是从 2^{Q_1} 到 2^{Q_2} 上的一个 Galois 连接; 2)如果映射序对$\langle \alpha, \gamma \rangle$是从 2^{Q_1} 到 2^{Q_2} 上的一个 Galois 连接,那么,存在唯一的关系 $\rho \subseteq Q_1 \times Q_2$,使得 $\alpha = post[\rho]$,并且 $\gamma = p\tilde{r}e[\rho]$.

Galois 连接自身所具有的一些数学属性使得它适用于抽象分析. 例如:抽象函数 α 和具体函数 γ 可以相互定义和构造;一方的属性可以用另一方进行表达. 此外,Galois 连接提供了通过对复杂对象的组成部分进行抽象实现对复杂对象整体进行抽象的方法, 如对函数和不动点的抽象等.

2.1.4 函数和不动点抽象(function and fixpoint abstractions)

考虑定义在具体域 C 上的函数集合 $\mathcal{F} = C \to C$ 以及定义在抽象域 A 上的函数集合 $\mathcal{F}^\sharp = A \to A$. 如果用 $\mathcal{F}^\sharp$ 对 $\mathcal{F}$ 进行逼近,那么可以根据 C 与 A 之间的正确性关系 $\rho \subseteq C \times A$ 来定义 $\mathcal{F}$ 与 $\mathcal{F}^\sharp$ 之间的函数的正确性关系 $\hat{\rho} \subseteq \mathcal{F} \times \mathcal{F}^\sharp$,使得 $f^\sharp \leq_{\hat{\rho}} f$ 当且仅当 $f^\sharp$ 保持 f 计算的正确性, 即:

$$f^\sharp \leq_{\hat{\rho}} f \Leftrightarrow \forall a \in A \cdot \forall c \in C \cdot a \leq_\rho c \Rightarrow f^\sharp(a) \leq_A f(c).$$

类似地,可以将 A 上的衡量精度的信息偏序关系扩展到 $\mathcal{F}^\sharp$ 上,即:

$$f_1^\sharp \leq_{\hat{\rho}} f_2^\sharp \Leftrightarrow \forall a \in A \cdot f_1^\sharp(a) \leq_A f_2^\sharp(a).$$

特别重要的是,对函数的抽象可以通过基于 C 和 A 上的 Galois 连接进行合成.

定理 2[16]. 令 $\mathcal{F}$ 和 $\mathcal{F}^\sharp$ 分别为如上定义的 C 和 A 上的函数集合, 如果 C 和 A 之间存在着 Galois 连接$\langle \alpha, \gamma \rangle$,那么对具体函数 f 的最精确逼近是:

$$f_\alpha^\sharp \triangleq \alpha \circ f \circ \gamma.$$

注意以上关于 $f_\alpha^\sharp$ 的定义不一定是可计算的(finitely computable). 因此在实际中,通常需要设计有效的算法来获得对具体函数 f 的正确的,但不一定是最精确的逼近函数.

程序分析中的许多属性可以用不动点来表示,抽象解释提供了对函数不动点进行抽象分析的方法. 考虑定义为完备格的具体域$(C, \leq_C, \vee_C, \wedge_C, \bot_C, \top_C)$和抽象域$(A, \leq_A, \vee_A, \wedge_A, \bot_A, \top_A)$,以及单调连续函数集合 $\mathcal{F}$ 和 $\mathcal{F}^\sharp$. 根据 Tarski 不动点定理,$\mathcal{F}$ 和 $\mathcal{F}^\sharp$ 中的函数存在不动点;同时,根据抽象解释的结果可以保证对不动点的逼近的正确性.

定理 3[16]. 令 $\mathcal{F}$ 和 $\mathcal{F}^\sharp$ 分别为如上定义的 C 和

$\mathcal{A}$ 上的函数集合. 对于 $f\in\mathcal{F}$ 以及 $f^{\sharp}\in\mathcal{F}^{\sharp}$,如果 $f\overset{\sharp}{\rho}f$,那么,$lfp\,f^{\sharp}\leq_{\rho}lfp\,f$.

进一步,根据 Kleene 不动点定理,函数的不动点可通过从一个给定的初始值开始的迭代计算的极限值得到. 因此,为了获得对具体函数不动点的逼近,可对抽象函数进行迭代计算. 对于有限抽象域 $\mathcal{A}$,计算 $\bigvee_{\mathcal{A}n\geq 0}f^{\sharp n}(\perp_{\mathcal{A}})$ 可在有限步骤内收敛,获得 $f^{\sharp}$ 的最小不动点,根据定理 3,这是对 $f^{\sharp}$ 的最小不动点的逼近.

然而,许多情况下,上述迭代计算并不一定保证收敛,如 $\mathcal{A}$ 定义在无限抽象域上时. 为此,抽象解释引入了了加宽(widening)算子来加速迭代计算,确保收敛,获得一个对 $f^{\sharp}$ 的最小不动点的逼近;同时,采用收窄(narrowing)算子来对该逼近进一步计算改进精度. 具体来说,$\mathcal{A}$ 上的加宽算子 $\nabla:\mathcal{A}\times\mathcal{A}\to\mathcal{A}$ 满足要求:1)对于 a_1 和 a_2,有 $a_1,a_2\leq_{\mathcal{A}}a_1\nabla a_2$;2)对任意递增序列 $a_1,a_2,\cdots$,由定义 $a'_1=a_1$ 和 $a'_{i+1}=a'_i\nabla a_{i+1}$ 得到的序列 $a'_1,a'_2,\cdots$ 确保收敛. 根据设定的加宽算子,定义迭代计算:1) $f^{\sharp\uparrow 0}\triangleq\perp_{\mathcal{A}}$;2) $f^{\sharp\uparrow i+1}\triangleq f^{\sharp\uparrow i}\nabla f^{\sharp}(f^{\sharp\uparrow i})(i\geqslant 0)$;该计算在有限的 k 步骤内确保收敛,并且有 $f^{\sharp\uparrow k}\leq_{\mathcal{A}}lfp\,f^{\sharp}$,即 $f^{\sharp\uparrow k}$ 是 $f^{\sharp}$ 的最小不动点的逼近.

另一方面,$\mathcal{A}$ 上的收窄算子 $\Delta:\mathcal{A}\times\mathcal{A}\to\mathcal{A}$ 满足要求:1)对于 a_1 和 a_2,$a_1\sqcap a_2\leq_{\mathcal{A}}a_1\Delta a_2\leq_{\mathcal{A}}a_2$;2)对任意序列 $a_1,a_2,\cdots$,由定义 $a'_1=a_1$ 和 $a'_{i+1}=a'_i\Delta a_{i+1}$ 得到的序列 $a'_1,a'_2,\cdots$ 确保收敛. 这种情况下,以 $f^{\sharp\uparrow k}$ 为初始值,定义迭代计算:1) $f^{\sharp\downarrow 0}\triangleq f^{\sharp\uparrow k}$;2) $f^{\sharp\downarrow i+1}\triangleq f^{\sharp\downarrow i}\nabla f^{\sharp}(f^{\sharp\downarrow i})(i\geqslant 0)$;该计算确保在有限的 l 步骤内能够收敛,并且同时有 $f^{\sharp\uparrow k}\leq_{\mathcal{A}}f^{\sharp\downarrow l}\leq_{\mathcal{A}}lfp\,f^{\sharp}$,即 $f^{\sharp\downarrow l}$ 仍是 $f^{\sharp}$ 的最小不动点的逼近,同时相对于 $f^{\sharp\uparrow k}$ 则更加精确.

2.2　软件模型检测中的抽象

抽象解释设计之初是为了进行静态程序分析. 在这种情况下,程序被表示为定义在程序状态集合上的转换函数(transfer function)F,所要分析的程序属性被定义为 F 的不动点. 给定一个程序状态集合的抽象域 $\mathcal{A}$,对程序的抽象解释(即 $\mathcal{A}$ 上的函数)可以通过将 F 中对程序状态的操作提升到抽象状态的层面上来构造. 根据 2.1 节中抽象解释的结果,对程序属性的逼近可通过计算抽象函数的不动点来得到[16,24]. 一般情况下,基于抽象解释的静态程序分析所验证的属性是可用 ACTL 表示的安全属性(universal safety property),如:描述在特定控制位置上所有可能的程序状态信息,即程序不变量(program invariant). 例如,如果变量 x 作为分母出现在除法表达式中,采用区间抽象域得到的 x 取值范围的逼近值为 $[a,b]$,并且 $0\notin[a,b]$,则可以判断相应的除法表达式不会发生被 0 除的错误. 而软件模型检测还需要分析包含存在性(existentiality)和活性(liveness)在内的程序动态行为. 为此,我们需要使用状态转换系统(state transition systems)(如 Kripke 结构)对程序进行建模,以获得更丰富的程序行为信息. 相应地,软件模型检测的抽象也就集中到对转换系统的抽象,而不是对转换函数的抽象. 然而,抽象解释作为一种关于抽象的基本理论,仍可作为不同应用领域中抽象分析的框架. 本节使用抽象解释的基本概念对软件模型检测中的抽象进行概述,介绍其中的主要组成部分.

给定一个程序 P 以及欲对其进行分析的行为属性 φ(如图 3 所示),具体的软件模型检测的过程主要包括 2 个步骤:1)模型构建. 使用一个状态转换系统 M 来表示程序 P 的语义(如图 3 中Ⅰ所示),称其为 P 的模型. 2)模型检测. 用时序逻辑公式 φ 描述所关心的行为属性,通过计算 $\|\varphi\|^{M}$ 的值来验证(如图 3 中Ⅱ所示). 由于程序的状态空间巨大甚至是无限,通常无法直接构造模型 M. 因此我们需要构造一个精简的有限抽象模型 $M^{\sharp}$(如图 3 中Ⅲ所示),使用它来逼近具体模型 M. 通过抽象实现对 φ 在 $M^{\sharp}$ 上的有效验证(如图 3 中Ⅳ所示),并进一步根据 $M^{\sharp}$ 与 M 之间的逼近关系所对应的程序属性的保持关系,判断 φ 在 M 上成立与否. 当由于抽象导致无法得到确定的模型检测结果时,需要通过精化构造更加精确的抽象模型(如图 3 中Ⅴ所示).

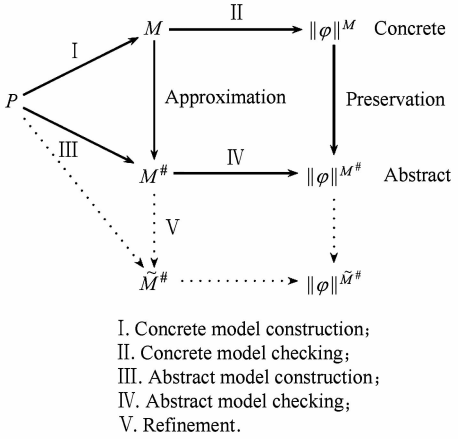


Fig. 3　Overview of abstraction in software model checking.

图 3　软件模型检测中的抽象

因此,整个抽象过程主要包括以下 3 个方面:1)设计抽象框架;2)构造抽象模型;3)抽象精化.

2.2.1 抽象框架(abstraction framework)

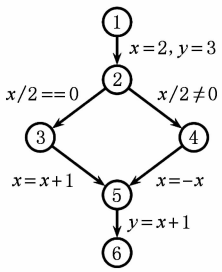
抽象框架形式化地严格描述了具体分析和抽象分析之间的关系.对软件模型检测而言,抽象框架包含 5 部分,下面分别对其进行介绍.

2.2.1.1 具体模型和抽象模型(concrete and abstract models)

软件模型检测中的具体模型空间 $\mathcal{M}$ 由所有定义在具体程序状态上的转换系统构成. $\mathcal{M}$ 通常对应于程序的结构操作语义(structural/small-step operational semantics),采用布尔 Kripke 结构来表示[25].具体来说,一个程序 P 可以采用标注的控制流图(control flow graph, CFG)$\langle Loc, Edge, \pi\rangle$ 来表示,其中 Loc 是 P 的有限的程序执行位置集合,$Edge \subseteq Loc \times Loc$ 是边的集合,π 在每条边 e_i 上标记一条程序语句 π_i. 例如,对于图 4(a)所示的程序 P_{ex},所对应 CFG 如图 4(b)所示.程序的状态空间是程序变量所有可能赋值的集合.根据程序的语义,每条边 e_i 上的语句 π_i 定义了相应程序执行位置之间的状态转换关系 r_i. 如对于位置 i 的赋值语句 $v:=expr$ 和任意的 2 个程序状态 $s,t,s,t\in r_i$ 当且仅当 $t=s[v \mapsto s(expr)]$,其中 $s(expr)$ 为表达式 $expr$ 在状态 s 下的取值. 例如,对于程序 P_{ex} 中行 3 的语句 $x=x+1$,有$(\langle x \mapsto 1, y \mapsto 1\rangle, \langle x \mapsto 2, y \mapsto 1\rangle) \in r_3$. 整个程序的状态转换系统由 CFG 中每条边所对应的状态转换关系的并集构成.

```
0:  void main() {
1:    int x=2, y=3;
2:    if (x/2==0) {
3:        x=x+1; }
      else {
4:        x=-x; }
5:    y=x+1;
6:    }
```

(a) P_{ex}



(b) CFG of P_{ex}

```
0:  void main() {
1:    p=true;
2:    if (*) {
3:        p=p? true, *; }
      else {
4:        p=p? false, true; }
5:    p=p;
6:    }
```

(c) The boolean program obtained using the predicate $p: x>0$

Fig. 4 A simple program.

图 4 一个简单的程序例子

抽象模型空间 $\mathcal{M}^{\sharp}$ 对 $\mathcal{M}$ 中模型的行为进行逼近,通常也采用某种状态转换系统表示,使得具体模型和抽象模型之间具有相似的结构特征.特别地,对程序 P 的抽象模型的构造,可通过对每条边 e_i 赋予一个逼近 r_i 的抽象转换关系 $r_i^{\sharp}$ 来进行.例如,对于上述 $x=x+1$ 语句,如果用谓词变量 $p: x>0$ 来定义抽象模型,即考虑执行 $x=x+1$ 对于 p 的影响.那么,如果 p 的值在执行 $x=x+1$ 之前为真,则在执行之后,p 的值仍然为真,即有:$(\langle p \mapsto \text{true}\rangle, \langle p \mapsto \text{true}\rangle) \in r_3^{\sharp}$.

抽象模型是抽象框架的基础,它决定了可以通过其进行有效分析的行为属性的集合.在软件模型检测领域,研究人员已经提出了许多种不同的模型用于对布尔 Kripke 结构进行抽象[23,26-30].在第 4 节中,我们将具体讨论不同抽象模型的结构特点,性质及其发展过程,并以此为主要线索展开为对不同抽象框架的讨论.

2.2.1.2 时序逻辑(temporal logic)

对于所需要分析的程序行为属性可通过某种时序逻辑描述,如:μ 演算,CTL,LTL 等. μ 演算采用不动点刻画时序逻辑属性,表达能力较 CTL 与 LTL 强;而 CTL 与 LTL 公式更易于理解,常在实际中被使用. L_{μ} 中所有 $\neg$ 操作符仅仅作用在原子命题上,并且仅包含模态算子□的公式被称为 $\Box L_{\mu}$ 子集;类似地, L_{μ} 中的 $\Diamond L_{\mu}$ 子集包含所有 $\neg$ 操作符仅仅作用在原子命题上,并且仅包含模态算子◇的公式.例如,公式 $\nu Z \cdot \neg p \wedge \Box Z$(即 CTL 中的 $AG\neg p$)属于 $\Box L_{\mu}$;公式 $\nu Z \cdot p \vee \Diamond Z$(即 CTL 中的 EFp)属于 $\Diamond L_{\mu}$. 我们分别称 $\Box L_{\mu}$ 和 $\Diamond L_{\mu}$ 所表示的属性为全称属性(universal properties)和存在属性(existential properties);它们分别描述程序中所有路径和某些路径上的行为属性.时序相关的属性也可以分为安全性和活性[31].通俗地讲,安全性表示“坏的事件不会发生”,关注程序行为的不变性,如程序不存在死锁,不存在数组越界等;活性则表示“好的事件会发生”,用于描述程序执行的进度,如程序的终止性,系统请求得到相应等.对于复杂的系统行为则需要采

用全称/存在属性和安全/活性属性组合所构成的时序属性来描述,如采用 ACTL 公式 AGEFp 所表示的属性.

2.2.1.3　正确性(soundness)

为了保证抽象模型检测的正确性,具体模型和抽象模型之间的逼近关系,也就是正确性关系 ρ 被定义为一种属性保持关系(property-preserving relation),即:对于一组属性 $\mathcal{L}$,一个抽象模型 $M^\sharp \in \mathcal{M}^\sharp$,以及一个具体模型 $M \in \mathcal{M}$,如果对于任意的属性 $\varphi \in \mathcal{L}$,它在 $M^\sharp$ 上的模型检测结果可以保持到 M 上,那么 $M^\sharp$ 逼近 M,记作 $M^\sharp \leq_\rho M$. 特别地,如果 $M^\sharp$ 上 φ 为真的结果能够被保持到 M 上,即 $\| \varphi \|^{M^\sharp} \Rightarrow \| \varphi \|^{M}$,则称该逼近可用于对属性 φ 的证实;例如,对于程序不存在死锁的属性 ψ,如果 $M^\sharp$ 包含的行为是 M 的超集,即自上逼近,那么如果 ψ 在 $M^\sharp$ 上为真,那么 ψ 在 M 上也为真,该属性得到证实. 类似地,如果只有假的结果能够被保持,即 $\neg \| \varphi \|^{M^\sharp} \Rightarrow \neg \| \varphi \|^{M}$,则称该逼近适用于对属性 φ 的证伪. 例如,如果 $M^\sharp$ 包含的行为是 M 的子集,即自下逼近,那么如果 ψ 在 $M^\sharp$ 上为假,即存在发生死锁的行为,那么 ψ 在 M 上也为假,属性得到证伪.

注意这种属性保持关系仅仅是从抽象模型到具体模型,称为弱保持(weak preservation)[28]. 基于弱保持的抽象分析的结果可以是不确定的. 例如:在对属性的证实分析中,如果 φ 在 $M^\sharp$ 上为假,那么我们无法确定 φ 在 M 上是否成立. 如果属性保持关系既包括从抽象模型到具体模型上,也包括从具体模型到抽象模型上,那么这样的保持称为强保持(strong preservation)[28].

在进程代数和程序精化的框架中,对转换系统的行为间的关系提出了模拟(simulation)的概念. 模拟也可以作为刻画软件模型检测中抽象分析的正确性(即模型间逼近关系)的一种结构化特征(structural characterization). 此外,抽象分析的正确性也可以用 Galois 连接和多值逻辑来进行描述. 这方面的内容也将在下一节中针对不同的抽象模型进行介绍.

2.2.1.4　精度与最优性(precision and optimality)

抽象模型的精度可以通过对公式的证实和证伪的能力来进行衡量. 例如:给定两个抽象模型 $M_1^\sharp$ 和 $M_2^\sharp$,如果在 $M_2^\sharp$ 上比在 $M_1^\sharp$ 上能够对更多属性得到确定的验证结果,也就是说,对于任意被验证的属性 φ,有 $\| \varphi \|^{M_1^\sharp} \Rightarrow \| \varphi \|^{M_2^\sharp}$,我们就认为 $M_2^\sharp$ 的精度比 $M_1^\sharp$ 的精度高,记作 $M_1^\sharp \leq M_2^\sharp$. 从这种意义上说,模型的精度之间的关系对应于模型之间的逼近关系. 即:如果 $M_1^\sharp \leq M_2^\sharp$ 那么对属性验证而言,$M_1^\sharp$ 是 $M_2^\sharp$ 的抽象模型. 例如,对于前面所提到的程序 P_{ex}(如图 4(a)所示)中行 3 的 $x=x+1$,如果仅采用谓词变量 $p:x>0$ 来定义抽象模型,那么对于 x 奇偶属性无法作出任何判断. 如果增加谓词变量 $q:x/2==0$ 来进行抽象,那么抽象模型中可包含如下状态转换关系:

$(\langle p \mapsto \text{true}, q \mapsto \text{true}\rangle, \langle p \mapsto \text{true}, q \mapsto \text{false}\rangle)$,

$(\langle p \mapsto \text{true}, q \mapsto \text{false}\rangle, \langle p \mapsto \text{true}, q \mapsto \text{true}\rangle)$,

从而能够验证更多属性,因此相对于仅用 p 所构造的抽象模型更加精确. 实际中所采用的抽象模型一般都是布尔 Kripke 结构的超集,这样允许使用统一的符号,如模拟来刻画抽象模型的精度以及具体和抽象模型之间的逼近关系.

给定精度的概念后,当对一个具体模型进行抽象分析的时候,最精确的(即最优的)抽象模型能够对尽可能多的属性作出确定的分析结果. 在实际中,抽象模型通常根据给定的对具体状态空间的抽象来定义. 抽象模型最优性[22-23,30,32]研究模型结构上的条件,如抽象状态转换关系的构造,称为最优条件,使得在给定一组抽象状态集合下,根据这种条件所得到的抽象模型是对原有的具体模型的最精确逼近. 根据抽象解释的基本理论,这些条件可通过对抽象模型中各部分的最精确逼近来获得. 第 3 节将对各种抽象模型介绍相关的最优条件的定义.

2.2.1.5　完备性(completeness)

完备性[33-35]考虑是否存在着能够对系统属性作出确定分析的抽象模型,即:对于一个抽象框架中的抽象模型集合 $\mathcal{M}^\sharp$,如果对于任意一个具体模型 M 和属性 $\varphi \in \mathcal{L}$,当 $\| \varphi \|^{M}$ 为真时,都存在着一个有限的抽象模型 $M^\sharp \in \mathcal{M}^\sharp$,使得 $M^\sharp$ 逼近 M 且 $\| \varphi \|^{M^\sharp}$ 为真,那么就认为这个抽象框架对于属性集 $\mathcal{L}$ 是完备的. 第 5 节将讨论相关的研究结果.

2.2.2　从程序中构造抽象模型

选定抽象框架之后,我们需要构造用于进行抽象分析的抽象模型,使得其能够逼近原程序所对应的具体模型(如图 3 中Ⅲ所示). 虽然最精确的抽象模型可以根据最优条件来刻画,但是实际中通常无法直接使用这些条件来构造抽象模型. 因为这些条件一般根据具体模型的结构而定义,而构建具体模型往往因为状态空间的庞大和无限而不可行,并且这也违背了采用抽象方法对状态空间进行化简的目的. 因此,为了避免生成具体模型,在实际中一般采用

直接从程序文本中构建抽象模型的方法.这个过程可以看作对程序的抽象解释,即将程序语义定义在抽象状态空间上,并且通过转换系统来表示.为此,首先需要根据与属性分析相关的数据特征确定抽象状态空间,然后在抽象状态空间上构建抽象模型,其中状态转换关系通过程序的抽象语义来计算.

在目前的软件模型检测器中比较常用的一种构造抽象模型的方法是 Graf 和 Saidi 提出的谓词抽象(predicate abstraction)[36],这种方法被广泛应用于各种软件验证工具中[6,8-10,13,37].谓词抽象的抽象状态空间通过基于程序变量上的谓词来定义,相应的状态转换系统可通过对最弱前置条件的分析来获得.具体来说,对于一个以标注的控制流程图表示的程序 $P=\langle Loc,Edge,\pi\rangle$,给定一组定义在程序变量上的谓词集合 $Pred$,谓词抽象首先使用 $Pred$ 中的谓词构建一个逼近 P 的布尔程序(boolean program, BP),这被称为语法抽象(syntactic abstraction),然后对 BP 定义抽象语义获得 P 的抽象模型[5,38].一个布尔程序 $BP=\langle Loc,Edge,\pi^{\#}\rangle$ 与 P 具有相同的控制结构,但 P 中的每一个具体语句都被定义在 $Pred$ 中谓词(布尔变量)上的一个或多个抽象语句逼近.这些抽象语句描述了对应的具体语句对相应的谓词的值的改变.采用定理证明器或者 SAT 求解器,这些抽象语句通过对每一个谓词 p,分别计算仅使用 $Pred$ 中的谓词所表示的蕴含最弱前置条件 $WP(\pi_i,p)$ 和 $WP(\pi_i,\neg p)$ 的布尔表达式来获得.例如,考虑语句 $x=x+1$ 以及谓词集合 $\{p:x>0\}$,对于最弱前置条件 $WP(x=x+1,p)$ 和 $WP(x=x+1,\neg p)$,有:

$$WP(x=x+1,p)\equiv x>-1,$$

$$WP(x=x+1,\neg p)\equiv x\leqslant -1.$$

采用 $\{x>0\}$ 中的谓词构造能够蕴含它们的布尔表示式,有:

$$p\Rightarrow x>-1,$$

$$\text{false}\Rightarrow x\leqslant -1,$$

也就是说,如果在执行 $x=x+1$ 之前 p 为真,则执行之后 p 仍为真;否则, p 的值未知.因此,语句 $x=x+1$ 被抽象为一个定义 p 上语句: $p=p?\text{true}:*$,其中的 $*$ 表示"未知".类似地,对于 P_{ex} 中条件语句 $\text{if}(x/2==0)$,由于仅仅根据 $x>0$,我们无法推测出条件 $x/2==0$ 何时为真或假,因此对该条件的抽象用"未知"表示.对 P_{ex} 语法抽象所得到的布尔程序如图 4(c)所示.在计算出布尔程序 BP 之后,就可以根据不同的抽象语义来构造抽象模型.例如,可将 BP 中的未知解释为在"真"和"假"之间的不确定选择(non-determinisitc choice)来得到对程序 P 的自上逼近[5,7].如对于语句 $p=p?\text{true}:*$,可将其解释为:如果当前状态下 p 的值为真,执行该语句后 p 为真,否则, p 的值可以不确定地取"真"或者"假";同样,对于条件语句 $\text{if}(*)$,根据自上逼近的语义,抽象程序从第 2 行可以不确定地执行到第 3 行或者第 4 行.

为了降低由于使用布尔向量表示状态集合所造成抽象域的规模,文献[5]在 SLAM 工具的实现中提出了被称为 Cartesian 抽象(cartesian abstraction)的谓词抽象方法,其中的抽象状态由 3 值向量表示,允许使用值 $\perp$ 表示由于抽象引起的信息丢失.文献[39]中为了提高谓词抽象的有效性提出了去除冗余谓词的方法,即去除一些可以被另一些谓词表达的谓词或者不影响抽象程序行为的谓词.在 BLAST[7] 中则通过考虑对程序的不同部分采用不同的谓词进行精化的方法,即惰性抽象,降低计算的复杂程度,提高抽象模型检测的效率.另外,在 JAVA PATHFINDER 中采用了 3 值逻辑明确表示由于抽象状态所造成的未知信息[11-12],避免了对虚假反例的判断.而文献[39-40]在软件验证工具 YASM 中则进一步采用 4 值逻辑构造更加精确的抽象模型,能够同时保持对属性的证实与证伪的确定判断结果.关于谓词抽象技术以及相关的优化方法可以参考文献[14].

2.2.3 抽象精化(abstraction refinement)

抽象总是会造成一定程度上的信息丢失,导致抽象模型检测无法得到确定的结果.即使我们能够构造出最精确的抽象模型,也不表示该模型已经精确到足以分析所有的属性(注意抽象模型是基于抽象状态集来进行构建的;"最优化"是相对于给定的抽象状态空间而言的).当模型检测结果不确定时,精化就显得尤其重要(如图 3 中Ⅴ所示).这个包含了抽象以及精化的自动过程称为抽象精化:它从一个粗糙的原始抽象模型开始,通过迭代精化,直到得到一个包含足够信息能够对属性作出确定判断的抽象模型.

然而,一般来说,对状态空间无限大的程序自动寻找到这样的抽象模型是不可计算的[7];否则,就意味着程序验证问题是可判定的.因此,实际中通常采用一种基于反例引导的抽象精化方法来对抽象模型进行精化[41].一般来说,抽象反例是指在抽象模型中能够证明属性无法成立的状态路径.精化的目标是通过在程序的具体模型上对反例的分析,去除虚

假反例,获得新的信息来构造更加精确的抽象模型.这些信息可以通过计算最弱前置条件[42-43],Craig插值[44]或者分析由SAT判定器[45-46]产生的不可满足性结果来获得.例如,对于图4(c)中的抽象程序采用自上逼近得到的抽象模型中从行2可以非确定地选择执行到行3或者行4,因此对属性ψ:$AG(x>0)$的抽象模型检测结果为"假",所对应的反例为执行路径1→2→4→5,而该路径在具体程序中并不可执行,因此为虚假反例.为去除该反例,可增加谓词$x/2==0$对行2的if$(x/2==0)$语句进行更精确地抽象,实现对原抽象模型的精化.

为了提高精化的有效性,即更快地得到一个规模较小并且能够提供确定检测结果的抽象模型,文献[47]提出结合概率学习方法以及进化算法来解决反例中抽象状态分割的问题.文献[48]则进一步提出快速判断反例真假的算法,以及通过增加辅助变量避免状态分割中复杂度高的问题,实现对抽象模型的精化.与完全基于反例的抽象精化有所不同,文献[49-50]基于抽象解释理论,通过对安全可达状态和错误不可达状态的前向和后向定点计算实现对抽象域的精化.这种方法在文献[51]中被用于对基于插值的模型检测中抽象模型的精化计算.文献[52]通过抽象解释的前向或后向函数完备性来描述基于反例的抽象精化,证明根据具体模型去除虚假反例的方法与采用程序语义对应的转换函数迭代计算抽象状态空间在本质上是一致的.具体的抽象精化的方法有多种多样,对其基本框架的比较可以参考文献[15].

3　抽象模型

本节讨论各种不同表现形式的抽象模型,包括:1)传统的支持证实的自上逼近方法以及支持证伪的自下逼近方法中所使用的布尔Kripke结构;2)分别对应于3值和4值Kripke结构的KMTS和MixTS,可同时支持自上逼近和自下逼近的抽象;3)具有超转换关系的GKMTS和HTS,可提供更精确的抽象模型检测.以此为主导介绍相应的抽象框架的研究内容,分析各种抽象模型对应的逼近关系,最优模型的结构定义条件及其功能上的局限性,突出新的抽象模型产生的意义.

3.1　布尔Kripke结构

首先介绍采用经典布尔Kripke结构(2-valued Kripke structure, 2VKS)表示的抽象模型.在这样的模型中,一个L_μ公式φ的值或者为t(真),或者为f(假).我们讨论基于2VKS的自上逼近(over-approximation)和自下逼近(under-approximation),分别用于对$\Box L_\mu$公式(全局属性)和$\Diamond L_\mu$公式(存在属性)的证实(或者等价地,分别应用于对存在属性和全局属性的证伪).

3.1.1　自上逼近

时序属性描述了期望系统发生的行为.因此一种直观地对系统模型的逼近的定义方法是考虑它们行为的"包含"关系.这可以使用模拟(simulation)概念来进行刻画.

定义4. 设$K_1=\langle S_1,R_1,I_1\rangle$和$K_2=\langle S_2,R_2,I_2\rangle$是2个2VKS.一个二元关系$\rho\subseteq S_1\times S_2$是$K_1$和$K_2$上的模拟关系,当对于任意$(s_1,s_2)\in\rho$,都有以下条件成立:

1) $I_1(s_1)=I_2(s_2)$;

2) $\forall t_1\in S_1\cdot R_1(s_1,t_1)\Rightarrow\exists t_2\in S_2\cdot R_2(s_2,t_2)\wedge\rho(t_1,t_2)$.

如果存在一个模拟关系ρ,使得K_1的初始状态通过ρ与K_2的初始状态相联系,那么从K_1的初始状态出发的所有计算行为都可以被从K_2的初始状态出发的计算行为所模拟.这种情况下,我们说K_2模拟K_1,记作:$K_2\leq_\rho K_1$.直观上而言,K_2的行为集合包含了K_1的行为集合.因此,对于全局属性$\Box L_\mu$,如果它在K_2上成立,那么它在K_1上必然也成立.

模拟的概念可以被用来定义布尔Kripke结构上的抽象模型与具体模型之间的正确性关系.即对于一个具体模型K,以及一个抽象的2VKS模型$K^\sharp$,如果$K^\sharp$模拟K,则$K^\sharp$是K的一个可适用于对全局属性进行证实分析的逼近.这样的逼近被称为自上逼近[26].进一步,抽象模型的精确性也可以通过基于模拟的序关系来定义,即:如果$K_2^\sharp$模拟$K_1^\sharp$,则$K_2^\sharp\leq K_1^\sharp$.直观上来说,这意味着$K_1^\sharp$可以比$K_2^\sharp$验证更多的$\Box L_\mu$中的属性.因此,$K_1^\sharp$比$K_2^\sharp$更精确.

自上逼近被广泛应用于对安全性的验证,如软件模型检测工具SLAM[5-6]和BLAST[7-8],可用来判断程序中是否存在错误状态.近年来,Cook等人开发的对程序终止性(termination)的自动验证工具TERMINATOR[53-54],也是基于自上逼近的抽象方法,其本质上是将对终止性的验证转化为一系列对程序可达性的判断;因此虽然它能够保持对终止性成立的判断结果,但是对于可能的非终止路径,还需要额外地判断其真实性.

如 2.2 节所述,在实际中抽象模型往往是基于给定的抽象状态空间来构造的.对于自上逼近,给定一个定义具体状态空间和抽象状态空间的二元关系 ρ,对具体模型的一个自上逼近的抽象模型可通过存在抽象(existential abstraction)的方法定义[26,55-56].

定义 5. 设 $K=\langle S,R,I\rangle$是一个具体 2VKS,$S^{\#}$ 是一个抽象状态的集合,$\rho\subseteq S\times S^{\#}$ 是一个二元关系.对 K 的存在抽象,$K^{\#\exists\exists}=\langle S^{\#},R^{\#},I^{\#}\rangle$的定义为

1) $I^{\#}(s^{\#})\triangleq\bigcap_{(s,s^{\#})\in\rho}I(s)$;

2) $R^{\#}\triangleq\{(s^{\#},t^{\#})\mid\exists s,t\cdot\rho(s,s^{\#})\wedge\rho(t,t^{\#})\wedge R(s,t)\}$.

在上述定义中,ρ 描述了具体状态和抽象状态之间的逼近关系.对于任意 2 个抽象状态 $s^{\#}$ 与 $t^{\#}$,如果它们对应的具体状态之间存在着状态转换关系,那么根据存在抽象的定义,在抽象模型中,从 $s^{\#}$ 可到达 $t^{\#}$.因此,K 中的任何行为在 $K^{\#\exists\exists}$ 中都有抽象行为相对应,这保证了抽象的正确性.例如:对于图 5(a)中的 K_{ex1} 模型,对给定分别对应具体状态集合$\{c_1,c_2\}$, $\{c_3\}$,$\{c_4\}$的抽象状态 a_{12},a_3,a_4.根据存在抽象,对 K_{ex1} 自上逼近的抽象模型如图 5(b)所示.

到目前为止,我们介绍了如何采用模拟的概念来刻画布尔 Kripke 结构上对全局属性$\square L_{\mu}$ 的保持关系,以及采用存在抽象定义相应抽象模型的方法.在进程代数的研究中,模拟的概念被用来描述进程间的模拟关系.而在软件模型检测中,抽象的目的是构建能够逼近具体模型并且能够对尽可能多的属性得到确定性检测结果的抽象模型.因此对抽象模型的精度以及最优性的研究具有重要意义.而基于模拟的理论框架中并没有提供进行这方面分析的方法,因此它不适合用于对最优性问题进行研究.

为了解决这个问题,Loiseaux 等人[23]从抽象解释的角度来研究抽象模型检测的属性保持关系.其方法的基本思想是:给定 2 个布尔 Kripke 结构之间的模拟关系,针对结构中状态转换关系所对应的前像函数 pre,可定义出与此模拟关系等价的基于 Galois 连接的 2 个结构间的逼近关系;而最精确的逼近,即最优性,则可依据抽象解释中函数抽象的结果,通过对函数 pre 的抽象而得到.具体来说,给定 2 个 2VKS:$K_1=\langle S_1,R_1,I_1\rangle$和 $K_2=\langle S_2,R_2,I_2\rangle$,以及它们之间的一个模拟关系 ρ,根据定理 1,我们可以定义 K_1 和 K_2 之间的序关系$\leq_{\langle\alpha,\gamma\rangle}$,其中 $\alpha=post[\rho]$且 $\gamma=p\tilde{r}e[\rho]$,并且当条件:

$$pre[R_2]\supseteq\alpha\circ pre[R_1]\circ\gamma \tag{1}$$

成立时,有 $K_2\leq_{\langle\alpha,\gamma\rangle}K_1$.

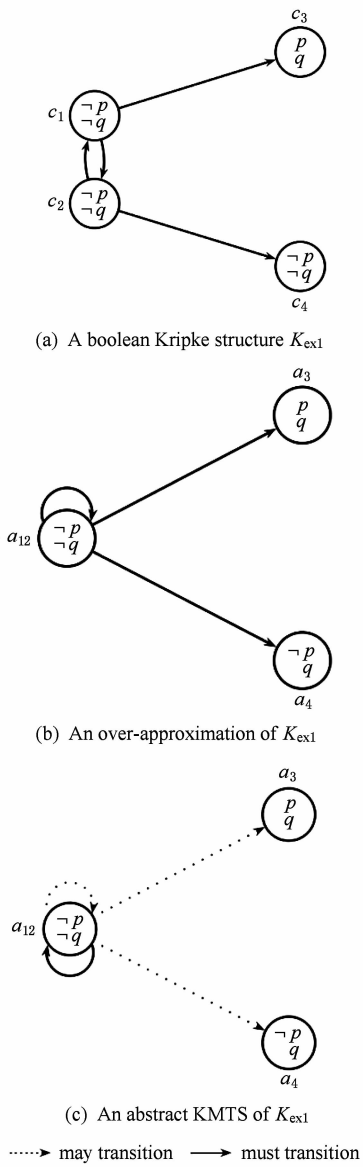


Fig. 5 A boolean Kripke structure and its abstractions.

图 5 一个布尔 Kripke 结构及其抽象模型

直观上,式(1)表示 K_1 中的每一个执行路径在 K_2 中都有相应的执行路径与之对应,而这恰好与模拟的概念一致.进一步,如果将 R_1 看作具体模型的转换关系,R_2 看作抽象模型的转换关系,那么式(1)则表示了关于前像函数 pre 的逼近.因此,给定一个具体模型 $K=\langle S,R,I\rangle$以及一个定义在 S 和抽象状态空间 $S^{\#}$ 上的关系 ρ,根据定理 2,对 R 的最精确的逼近可以定义为使得式(1)中等号成立的关系 $R^{\#}\subseteq S^{\#}\times S^{\#}$.特别地,对于 S 的一个等价划分 P,设 P 与 $S^{\#}$ 同构;那么,如果 ρ 是从 S 到 $S^{\#}$ 的函数,并且 I 关于 ρ 是一致的,即在同一等价类中,s_1 和 s_2 有$I(s_1)=I(s_2)$,那么根据存在抽象所定义的抽象模型 $K^{\#\exists\exists}$ 就相对 ρ 的最精确的逼近[23,55].文献[23]

的重要意义在于建立了模拟与 Galois 连接之间关于抽象模型检测中属性保持的联系:模拟提供了对抽象模型和具体模型之间逼近关系的直观解释,而基于 Galois 连接的抽象可以利用抽象解释理论来分析抽象模型构造的精确性.

3.1.2　自下逼近

自下逼近是自上逼近的对偶,对其可采用模拟关系的逆进行描述,即当 $K\leq_{\rho}K^{\#}$ 时,称 $K^{\#}$ 自下逼近 K.在自下逼近中,$K^{\#}$ 的行为是 K 行为的子集;因此,如果一个存在属性,即 $\Diamond L_{\mu}$ 中的一个公式在 $K^{\#}$ 上成立,那么它在 K 上也成立.所以自下逼近可用于证实存在属性,检测程序中存在的缺陷.如:结合具体的测试和符号执行对状态空间进行搜索[57-59],根据抽象状态之间必然存在的转换关系定义抽象模型[60-62],或者通过将程序中的循环展开有限步骤等进行有界模型检测[12,63-64].

与通过存在抽象定义自上逼近抽象模型相对应,给定一个抽象状态集 $S^{\#}$,我们可以通过全局抽象(universal abstraction)来定义自下逼近抽象模型.直观上,为了保证每一个抽象的执行路径都可对应于具体模型中的某个执行路径,对于任意两个抽象状态 $s^{\#}$ 与 $t^{\#}$,只有当 $s^{\#}$ 所表示的每一个具体状态都可以到达 $t^{\#}$ 所表示的某个具体状态时,在抽象模型中才允许从 $s^{\#}$ 转换到 $t^{\#}$.

定义 6. 设 $K=\langle S,R,I\rangle$ 是一个表示具体模型的 2VKS,$\rho\subseteq S\times S^{\#}$ 是一个二元关系,$S^{\#}$ 是一个抽象状态集.K 的全局抽象——$K^{\#\forall\exists}=\langle S^{\#},R^{\#},I^{\#}\rangle$ 的定义如下:

1) $I^{\#}(s^{\#})\triangleq\bigcap_{(s,s^{\#})\in\rho}I(s)$;

2) $R^{\#}\triangleq\{(s^{\#},t^{\#})\mid\forall s\cdot\exists t\cdot\rho(s,s^{\#})\Rightarrow\rho(t,t^{\#})\wedge R(s,t)\}$.

与基于等价划分的抽象状态空间进行存在抽象能够获得最精确的自上逼近类似,根据对偶性,自下逼近的最优性也可通过对基于等价划分的抽象状态空间进行全局抽象而获得.

3.1.3　自上逼近和自下逼近的结合

自上逼近和自下逼近的局限性在于它们仅仅能够对 L_{μ} 公式的子集($\Box L_{\mu}$ 或者 $\Diamond L_{\mu}$)保持正确性.在实际中,我们常常需要表达既包括全局性也包括存在性的时序逻辑属性,例如在任意情况下,系统中都存在着允许用户发送请求的执行路径.为了能够对 L_{μ} 中所有的公式都保持抽象分析的正确性,很自然的一种方法就是将自上逼近和自下逼近结合起来,通过互模拟来刻画具体模型与抽象模型之间的逼近关系.

定义 7. 设 K_1 和 K_2 是 2 个 2VKS.当 $K_1\leq_{\rho}K_2$ 且 $K_1\leq_{\rho^{-1}}K_2$ 时,二元关系 $\rho\subseteq S_1\times S_2$ 是 K_1 和 K_2 上的互模拟关系.

如果抽象模型 $K^{\#}$ 和 K 是互模拟的,则 $K^{\#}$ 和 K(相对于 ρ)具有等同的计算行为.因此,一个 L_{μ} 公式能够被 $K^{\#}$ 满足当且仅当它能被 K 满足;这是一种强保持关系,可同时支持证实和证伪的抽象模型检测.例如:对于具有对称性的系统,可以采用对称交换群所导出的状态轨道集合为抽象状态空间,在这上面采用存在抽象构造的自上逼近模型与采用全局抽象构造的自下逼近模型相同,其实质上就是原模型相对于对称交换群的商结构[65].该商结构与原模型之间存在互模拟关系,因此保持对所有的 L_{μ} 属性进行抽象模型检测的正确性.

3.1.4　局限性

如 3.1.3 节所述,为了能够保持所有 L_{μ} 属性的模型检测结果,可采用布尔 Kripke 结构来构造抽象模型.但是这种方法的主要问题就是基于互模拟的抽象约束性过强.在实际中,并不是总能找到像对称商结构这样的抽象模型,使得既与原模型之间存在互模拟,同时又能够显著降低原模型的状态规模.在许多情况下,一个表示具体模型的布尔 Kripke 结构关于互模拟的最小抽象模型其实就是它自身,而这样的抽象模型对于降低抽象模型的规模显然没有任何帮助.基于混合转换关系的抽象模型就是为解决这个问题而提出来的,其中的状态转换关系一方面来自于自上逼近,另一方面来自于自下逼近,但是并不要求这两者是相同的.这是与互模拟情况下抽象模型的不同之处,也使得自上逼近和自下逼近的结合更具有通用性.

3.2　混合转换结构

本节介绍 2 种具有混合转换关系的结构:KMTS[66]和 MixTS[28],它们分别与 3 值 Kripke 结构(3-valued Kripke structure, 3VKS)和 4 值 Kripke (4-valued Kripke structure, 4VKS) 结构相对应.

3.2.1　KMTS 与 3VKS

与布尔 Kripke 结构相比,KMTS 是一种表达能力更强的模型.KMTS 包含 2 种状态转换关系,may 转换(may transitions)和 must 转换(must transitions),分别表示可能与必然发生的状态转换关系,并且分别对应自上逼近与自下逼近.KMTS 要求 must 转换是 may 转换的子集,也就是说,每一个必然的状态转换都是可能的状态转换.此外,KMTS 中的状态用原子命题的文字(literals)子集

进行标记. 因此,在某个状态上,一个原子命题的值可以是未知 — 既不为真也不为假. 图 5(c)、图 6(a)(b)给出了一些 KMTS 的例子. 其中对于图 5(c)中的模型,根据 may 转换(虚线),从状态 a_{12} 可能到达 a_{12},a_3,a_4;另外,根据 must 转换(实线),从 a_{12} 必然能够到达 a_{12} 自身,即存在一个环路. 布尔 Kripke 结构是 KMTS 的一种特殊情况,被称为完全(complete) KMTS[67],因为其中的 may 和 must 转换可看作是相同的,可以合并为一个确定的状态转换关系;同时每一个原子命题在每个状态上也都有一个确定值,要么为真,要么为假. 因此布尔 Kripke 结构可以看作是一种不包含不确定信息的 KMTS.

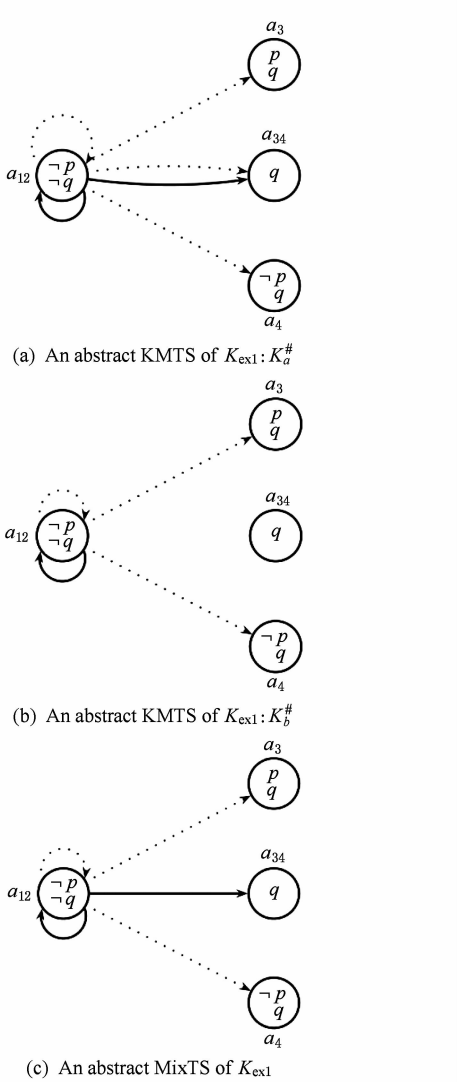


Fig. 6 KMTS and MixTS abstactions of K_{ex1}.

图 6 K_{ex1} 的 KMTS 和 MixTS 抽象

定义 8. 原子命题集合 AP 上的 Kripke 模态转换关系(Kripke modal transition, KMTS)是一个四元组 $K=\langle S, R^{must}, R^{may}, I\rangle$,其中 S 是状态集, R^{must} 和 $R^{may}: S\times S\to 2$ 分别是 must 转换和 may 转换,其中,$R^{must}\subseteq R^{may}$,$I: S\to 2^{literal(AP)}$ 是对每个状态分配一个 AP 的文字子集的标记函数,其中对于任何 $p\in AP$,p 和 $\neg p$ 不能同时被分配给同一个状态.

L_μ 在 KMTS 上的语义采用 3 值逻辑定义. 对于原子命题 p 与状态 s,如果 $p\in I(s)$,那么 p 在 s 上的值为 t(真);如果 $\neg p\in I(s)$,那么值为 f(假);否则,即 p 和 $\neg p$ 都不属于 $I(s)$ 时,p 在 s 上的值为未知($\perp$). 对于◇φ 的值,通过对存在操作符◇分别在 may 转换上以及 must 转换上的解释得到.

$$\|\Diamond\varphi\|(s)=\begin{cases}\mathrm{t}, \text{当}\ \exists t\in S\cdot R^{must}(s,t)\wedge\\ \quad \|\varphi\|(t)=\mathrm{t};\\ \mathrm{f}, \text{当}\ \forall t\in S\cdot R^{may}(s,t)\to\\ \quad \|\varphi\|(t)=\mathrm{f};\\ \perp, \text{其他}.\end{cases}\tag{2}$$

也就是说,对于公式◇φ 以及 KMTS 上的状态 s,如果存在着一个从 s 到某个状态 t 的 must 转换,并且 φ 在 t 上为真,那么◇φ 在 s 上的值为真;否则,如果对任意一个从 s 到某个状态 t 的 may 转换,φ 在 t 上的值都为假,那么◇φ 在状态 s 上的值为假;如果以上 2 种情况都不成立,那么◇φ 在 s 上的值为未知. 例如:对于图 5(c)所示的 KMTS,公式◇$\neg p$ 在状态 a_{12} 上为真,因为存在一个从 a_{12} 到 a_{12} 的 must 转换,并且 $\neg p$ 在 a_{12} 上成立;而公式在◇p 上的值未知,因为不存在从 a_{12} 到使得 p 为真的状态之间的 must 转换,但是存在着从 a_{12} 到 a_3 的 may 转换,而且 p 在 a_3 为真;此外,公式◇($p\wedge\neg q$)在 a_{12} 上为假,因为 $p\wedge\neg q$ 在所有从 a_{12} 经 may 转换到达的状态上都为假.

具体模型 K 与用于抽象的 KMTS 模型 $K^{\#}$ 之间的逼近关系可以通过要求其保持确定的模型检测结果来定义,即:如果公式 φ 在 $K^{\#}$ 上为真(或假),那么它在 K 上也为真(或假). 当 φ 在 $K^{\#}$ 上未知时,则无法判断 φ 是否在 K 成立,这表示由于抽象导致了信息丢失,从而无法得到确定的结果. 这样定义的逼近关系被称为确定逼近(exact-approximation)[39-40]. 与互模拟的情况类似,确定逼近并非仅仅适用于 L_μ 的某个子集(如□L_μ 或者◇L_μ),而是能够保持所有 L_μ 公式的确定模型检测结果;同时,由于确定逼近允许未知的模型检测结果的存在,它提供了一种比互模拟更灵活的自上逼近与自下逼近相结合的抽象框架.

确定逼近可以通过混合模拟(mixed simulation)[21,28]来进行刻画. 混合模拟是对模拟概念的扩

展,针对 must 和 may 转换关系分别定义了与其对应的模拟关系.

定义 9. 设 $K_1=\langle S_1,R_1^{must},R_1^{may},I_1\rangle$ 和 $K_2=\langle S_2,R_2^{must},R_2^{may},I_2\rangle$ 为 2 个 KMTS. 关系 $\rho\subseteq S_1\times S_2$ 是 K_1 和 K_2 上的混合模拟关系,当对于任意的状态对 $(s_1,s_2)\in\rho$ 有以下 3 个条件成立:

1) $I(s_2)\subseteq I(s_1)$;

2) $\forall t_2\in S_2\cdot\exists t_1\in S_1\cdot R_2^{must}(s_2,t_2)\Rightarrow R_1^{must}(s_1,t_1)\wedge\rho(t_1,t_2)$;

3) $\forall t_1\in S_1\cdot\exists t_2\in S_2\cdot R_1^{may}(s_1,t_1)\Rightarrow R_2^{may}(s_2,t_2)\wedge\rho(t_1,t_2)$.

如果存在一个混合模拟关系 ρ 使得 K_1 的初始状态通过 ρ 与 K_2 的初始状态相关联,那么相对于混合模拟所对应的序关系,K_2 比 K_1 小,记作 $K_2\leq_\rho K_1$. 直观上,这表示 K_2 没有 K_1 精确,因为在这种情况下,从 K_2 的初始状态出发的行为集合与从 K_1 的初始状态出发的行为集合相比,包含更少的必然(must)转换和更多的可能(may)转换;因此,K_2 比 K_1 包含更多不确定信息,满足更少的属性,精确度相对较低.

对于一个给定的具体模型所对应的布尔 Kripke 结构 K,以及关联具体状态空间 S 和抽象状态空间 $S^\sharp$ 的一个二元关系 $\rho\subseteq S\times S^\sharp$,根据自上逼近和自下逼近相结合的思想,我们可以构造定义在 $S^\sharp$ 上的一个逼近 K 的 KMTS,其中的 may 转换和 must 转换分别通过存在抽象和全局抽象来构造. 例如:图 5(c)表示逼近图 5(a)中具体模型 K_{ex1} 的一个 KMTS. 它与图 5(b)中所示的对 K_{ex1} 自上逼近的 2VKS 使用相同的抽象状态集. 根据 3.1 节中的自上逼近和自下逼近的结果,如果抽象状态集对应于具体状态空间的一个等价划分,那么这种通过存在抽象与全局抽象构造的 KMTS 抽象模型相对于混合模拟的序关系满足最优性. 文献[27]中 Cleaveland 等人对此作了详细的分析.

KMTS 模型与 3VKS 是同构的[21]. 因此,KMTS 与 3VKS 之间可以相互转化:同时为 may 和 must 的转换对应 t(真),仅为 may 而非 must 的转换对应 $\perp$(未知),空转换对应 f(假);对于原子命题 p,如果 p 被标记在一个状态上,则 p 在该状态上的值为真;如果 $\neg p$ 被标记,则值为 f;否则,值为 $\perp$. 根据 3 值逻辑中信息序关系的定义,$a\leq b$ 表示 a 比 b 有更少的确定信息. 这种序关系可以很自然地被扩展到对 3VKS 精确度的比较,这与采用混合模拟来衡量 KMTS 之间的精确度是等价的[21].

定理 4. 设 $K_1=\langle S_1,R_1^{must},R_1^{may},I_1\rangle$ 和 $K_2=\langle S_2,R_2^{must},R_2^{may},I_2\rangle$ 为 2 个 KMTS,并且 K_1' 和 K_2' 是与其分别对应的 3VKS. 关系 $\rho\subseteq S_1\times S_2$ 是 K_1 和 K_2 上的混合模拟关系,即 $K_2\leq_\rho K_1$,当且仅当对于任意的状态对 $(s_1,s_2)\in\rho$ 有如下条件成立:

$$\forall\varphi\in L_\mu\cdot\|\varphi\|^{K_2'}(s_2)\leq\|\varphi\|^{K_1'}(s_1).$$

使用多值逻辑给我们提供了另外一种用来描述抽象模型之间的确定逼近关系的方式. 使得可以采用统一的概念,即逻辑值的信息序关系,来对不同对象(如原子命题的状态标注、状态间的转换关系、抽象模型检测结果)的精确度进行分析. 同时多值逻辑也可以被用作真值域直接对包含不确定信息的抽象模型进行推理计算. 在软件模型检测器的应用方面,基于自上逼近的工具,如 SLAM 和 BLAST,对于所产生的反例,由于自上逼近可能引入原程序不存在的行为,因此必须检查其真实性. 在 JAVA PATHFINDER[11-12]中,采用了 3 值逻辑构建抽象模型,能够明确表示由于抽象所造成的未知信息,从而避免对虚假反例的判断.

3.2.2　局限性

除了 KMTS 之外,还有另外几种抽象结构与 3VKS 具有相同的表达能力[21],包括 Democratic Kripke 结构[27]、模态转换系统(modal transition system, MTS)[67-69]、不完备 Kripke 结构(partial Kripke structure, PKS)[70-71]. 它们都要求 $R^{must}\subseteq R^{may}$,即 must 转换是 may 转换的子集. 这种约束条件最初是为了确保采用这种形式所定义的软件需求规约是可实现的(implementable)而引入的[68,72]. 但是,这种约束也妨碍使用 KMTS 来构造更加精确的抽象模型. 例如:对于图 5(a)所示的具体模型 K_{ex1},除了 a_{12},a_3,a_4 之外,假设存在另外一个抽象状态 a_{34} 对应于 $\{a_3,a_4\}$,并且用 q 标记. 根据存在抽象和全局抽象,可以构造一个 KMTS 模型 $K_a^\sharp$(如图 6(a)所示)对 K_{ex1} 确定逼近. 考虑 L_μ 公式 $\Diamond q$ 和 $\Diamond(p\wedge\neg p)$(注意根据 L_μ 语义,对 $\|p\wedge\neg p\|$ 的计算需要先分别计算 $\|p\|$ 和 $\|\neg p\|$),它们的值在 K_{ex1} 中的具体状态集合 $\{c_1,c_2\}$ 上分别为真和假. 但是在 $K_a^\sharp$ 中,虽然在与 $\{c_1,c_2\}$ 对应的抽象状态 a_{12} 上,$\Diamond q$ 为真,但是 $\Diamond(p\wedge\neg p)$ 在 a_{12} 上的值却是未知. 所以 $K_a^\sharp$ 无法保证对这 2 个公式同时获得确定的模型检测结果. 进一步,假设我们删除 a_{12} 和 a_{34} 之间的状态转换关系,可以得到如图 6(b)所示的 KMTS 模型 $K_b^\sharp$;在这个模型上,可以获得 $\Diamond(p\wedge\neg p)$ 在 a_{12} 上

为假的确定结果,但是◇q 在 a_{12} 上的结果则变成了未知. 事实上,对于本例给定的抽象状态集合,在KMTS的框架内,无法构造出使得◇q 和◇($p \wedge \neg p$)同时获得确定结果的抽象模型. 这个问题可以通过考虑以混合转换系统作为模型的抽象框架来解决. 它是对 KMTS 的扩展,能够提供更精确的抽象分析结果.

3.2.3 MixTS 和 4VKS

MixTS[28,73]是对 KMTS 的扩展,它去除了约束条件 $R^{\text{must}} \subseteq R^{\text{may}}$. 也就是说,KMTS 是 MixTS 的一种特殊形式. MixTS 之间的确定逼近关系和精度也可以通过混合模拟来进行刻画. 此外,可以采用类似于将 KMTS 转化为 3VKS 的方法,将 MixTS 转化为等价的 4VKS,只需要另外将状态间的为 must 而非 may 的转换赋值为 ⊤. 对于满足一致性(consistency)[72]的 MixTS 抽象模型 M, L_μ 公式在 M 上的模型检测结果总是为真、假、或者未知. 可同样根据式(2)定义 L_μ 在 MixTS 上的 3 值逻辑定义.

使用 MixTS 可以定义比 KMTS 更精确的抽象模型. 例如:对于图 5(a)中的 K_{exl},同样使用抽象状态集$\{a_{12}, a_3, a_4, a_{34}\}$可以定义如图 6(c)所示的抽象 MixTS 模型,其中 a_{12} 与 a_{34} 之间存在着一个 must 转换,但是并不存在 may 转换. 容易验证在这种情况下公式◇q 和◇($p \wedge \neg p$)在 a_{12} 状态上分别为真和假,可以同时得到确定的结果;因此,比采用 KMTS 构造的抽象模型更加精确.

这种精确度增长的原因在于建立 MixTS 抽象模型时考虑了抽象状态之间相对于信息表示的精确度. 在文献[28]中,Dams 等人研究了使用 MixTS 来构建抽象模型时的最优性问题. 其中,抽象状态空间通过一个偏序集$\langle S^\sharp, \leq_\sharp \rangle$给出($\leq_\sharp$ 表示 $S^\sharp$ 上的信息序关系). 给定一个表示具体模型的布尔 Kripke 结构 $K = \langle S, R, I \rangle$. K 上状态集的幂集$\langle \mathcal{P}(S), \subseteq \rangle$通过一个 Galois 连接$\langle \alpha, \gamma \rangle$与$\langle S^\sharp, \leq_\sharp \rangle$相关. Dams 等人[28]证明最精确的 MixTS 模型 $K^\sharp$,可通过在 $S^\sharp$ 上采用存在抽象和全局抽象进行构造,但是抽象转换关系必须定义在相对最精确的抽象状态. 也就是说,$K^\sharp$ 中的转换关系 $R^{\sharp\,\text{must}}$ 与 $R^{\sharp\,\text{may}}$ 根据下列条件定义:

$$R^{\sharp\,\text{must}}(a,b) \Leftrightarrow b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid R^{\forall\exists}(\gamma(a), Y')\}\},$$

$$R^{\sharp\,\text{may}}(a,b) \Leftrightarrow b \in \{\alpha(Y) \mid Y \in \min\{Y' \mid R^{\exists\exists}(\gamma(a), Y')\}\}.$$

其中:

$$R^{\forall\exists}(S,T) \triangleq \forall s \in S \cdot \exists t \in T \cdot R(s,t), \quad (\forall\exists\,\text{rule}),$$

$$R^{\exists\exists}(S,T) \triangleq \exists s \in S \cdot \exists t \in T \cdot R(s,t), \quad (\exists\exists\,\text{rule}),$$

分别对应全局抽象和存在抽象. 例如:对于图 6(c)中的抽象状态 a_{12}, $a_3 a_{34}$,分别对应于 $\gamma(a_{12}) = \{c_1, c_2\}$, $\gamma(a_3) = \{c_3\}$, $\gamma(a_{34}) = \{c_3, c_4\}$. 由于在具体模型 K_{exl} 上(图 5(a)), $\gamma(a_{12})$ 和 $\gamma(a_3)$ 之间存在状态转换关系,同时 $\gamma(a_{12})$ 和 $\gamma(a_{34})$ 之间也存在状态转换关系. 根据存在抽象,从 a_{12} 到 a_3 以及从 a_{12} 到 a_{34} 之间都有 may 转换. 但是,由于 $\gamma(a_3) \subset \gamma(a_{34})$,也就是说, a_3 比 a_{34} 更加精确,所以为获得最优的抽象 MixTS 模型,根据 Dams 等人[28]所定义的条件,仅在 a_{12} 和 a_3 之间定义 may 转换,而在 a_{12} 和 a_{34} 之间则不添加 may 转换,得到图 6(c)所示的 MixTS 模型. 这种定义使得◇q 和◇($p \wedge \neg p$)在 a_{12} 上同时可以得到确定的模型检测结果.

文献[28]中的不足之处在于并没有给出如何系统地得到上述最优性构造条件的方法. 在其论文中,Dams 等人主要是依靠直觉先给出上述条件,然后证明其正确性. 在文献[32]中,Schmidt 通过构造状态转换系统之间的 Galois 连接,分析对应的自上逼近和自下逼近,从而得出与文献[28]中相同的结果. 在文献[22]中,Gurfinkel 等人结合抽象解释理论与基于双格的多值逻辑提出了一种系统地构造抽象模型的方法. 这种方法以多值逻辑为真值域,首先定义了对集合的逼近,根据抽象解释中对函数的抽象获得对基于集合的 L_u 解释的最精确逼近;然后,对基于模型的 L_u 解释的最精确逼近可以归约为对其中每一个组成部分的抽象,例如,标注函数和◇操作符等;再进一步针对多值 Kripke 结构作相应的定义得到了对其进行抽象的最优条件. 与文献[28,32]相比,文献[22]中的方法更系统化,所得到的结果也更加通用. 特别地,上述对 MixTS 构造最精确逼近的条件可以看作将文献[22]中的结果应用于 4VKS 上的特殊情况. 在软件模型检测的应用方面,在 Gurfinkel 等人[38,40,74]采用 4 值逻辑构造程序的抽象模型,实现了软件验证工具 YASM,能够同时对属性的证实与证伪作确定的判断.

3.2.4 局限性

根据上述结果,给定一个布尔 Kripke 结构和一个抽象状态集,可以定义一个对所有 L_u 公式保持确定模型检测结果的最优的抽象 MixTS 模型. 那么一

个有意思的问题是:是否可以使用表现形式更强的模型来获得更加精确的抽象模型?注意到目前为止我们所考虑的抽象模型都是可以用多值 Kripke 结构来表示的,其中的状态转换关系的起始与终止都定义在相同的状态空间上,即每个转换都是从一个抽象状态到另一个抽象状态.文献[34,75-76]证明这种形式上的要求会对模型检测造成精度上的损失.例如:考虑 K_{ex1} 上的属性 $\psi=\Diamond(q\wedge(p\vee\neg p))$ 它在 c_1 和 c_2 上为真.然而,对于如图 6(c)所示的最精确的抽象 MixTS 模型,它在与 $\{c_1,c_2\}$ 对应的抽象状态 a_{12} 上的值为未知,因此无法在抽象模型上判断它在对应的具体状态 c_1 和 c_2 上的值.为了解决这个问题,可以考虑包含超转换(hyper-transitions)的抽象模型,它允许表示从一个状态到一个状态集的转换关系,能够提供更精确的抽象.这是下面所讨论的内容.

3.3　超转换结构

我们介绍 2 种超转换结构:GKMTS[29] 和 HTS[30,77].相对于 MixTS 模型,GKMTS 增加了 must 超转换,能够对更多的存在属性得到确定的检测结果,从而提高精确度.HTS 则不仅包含 must 超转换,同时也允许 may 超转换,因此比 GKMTS 在结构形式上更加灵活,能够同时对更多的全局属性获得确定的验证结果.

3.3.1　GKMTS

GKMTS 与 MixTS 类似,主要不同之处在于允许 must 转换可以从一个状态到达一个状态集,称为 must 超转换.

定义 10. 状态集 S 上的一个超转换是一个有序对 (s,A),其中 $s\in S$,且 $A\subseteq S$ 是一个非空集合.

定义 11. 一个广义 Kripke 模态迁移系统(generalized Kripke modal transition system, GKMTS)是一个四元组:$K=\langle S,R^{must},R^{may},I\rangle$,其中 S,R^{may} 和 I 与在 MixTS 中的定义相同,$R^{must}\subseteq S\times 2^S$.此外,对于每一个 $(s,A)\in R^{must}$ 和 $S'\in A$,有 $(s,s')\in R^{may}$.

L_μ 在 GKMTS 上的语义与在 KMTS 上的定义类似,除了存在操作符 $\Diamond$ 关于 must 转换部分需要改为在 must 超转换上进行定义:

$$\|\Diamond\varphi\|s)=\begin{cases}\mathrm{t},\ \text{当}\ \exists A\subseteq S\cdot R^{must}(s,A)\wedge\bigwedge\limits_{t\in A}\|\varphi\|(t)=\mathrm{t};\\ \mathrm{f},\ \text{当}\ \forall t\in S\cdot R^{may}(s,t)\rightarrow\|\varphi\|(t)=\mathrm{f};\\ \bot,\ \text{其他}.\end{cases}\tag{3}$$

也就是说,对于公式 $\Diamond\varphi$ 以及 GKMTS 上的状态 s,如果存在着从一个 s 到某个状态集合 A 的 must 超转换,并且 φ 在 A 中所有状态上都为真,那么 $\Diamond\varphi$ 在 s 上为真;例如,对于图 5(a)中的 K_{ex1} 以及抽象状态集 $\{a_{12},a_3,a_4\}$,我们可以定义如图 7(a)所示的抽象 GKMTS;因为存在一个从 a_{12} 到 $\{a_3,a_4\}$ 的 must 超转换,并且公式 q 和 $p\vee\neg p$ 在 $\{a_3,a_4\}$ 上都为真,因此前面讨论的公式 $\psi=\Diamond(q\wedge(p\vee\neg p))$ 在 a_{12} 上为真.

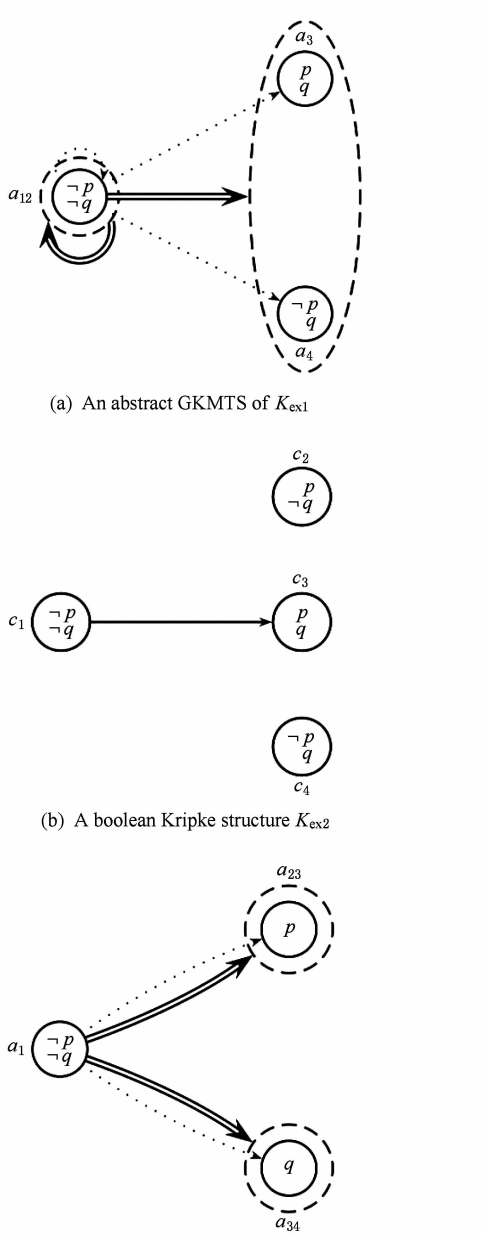


Fig. 7　GKMTS abstactions.

图 7　GKMTS 抽象模型

布尔 Kripke 结构可以看作是一种特殊的 GKMTS,其中每个 must 超转换的目标集合仅包含一个状态.与 KMTS 类似,GKMTS 上的逼近关系既可以使用 3 值逻辑语义来描述,也可以使用广义混合模拟来等价地表达.

定义 12. 设 $K_1=\langle S_1,R_1^{\text{must}},R_1^{\text{may}},I_1\rangle$ 和 $K_2=\langle S_2,R_2^{\text{must}},R_2^{\text{may}},I_2\rangle$ 为 2 个 GKMTS. 二元关系 $\rho\subseteq S_1\times S_2$ 是 K_1 和 K_2 上的广义混合模拟关系,当对每个 $(s_1,s_2)\in\rho$,有以下 3 个条件成立:

1) $I(s_2)\subseteq I(s_1)$;

2) $\forall A_2\subseteq S_2\cdot\exists A_1\subseteq S_1\cdot R_2^{\text{must}}(s_2,A_2)\Rightarrow R_1^{\text{must}}(s_1,A_1)\wedge\bar{\rho}(A_1,A_2)$,其中 $\bar{\rho}$ 是对 ρ 在集合上的扩展,使得 $\bar{\rho}(A_1,A_2)\Leftrightarrow\forall s_1'\in A_1\cdot\exists s_2'\in A_2\cdot\rho(s_1',s_2')$;

3) $\forall t_1\in S_1\cdot\exists t_2\in S_2\cdot R_1^{\text{may}}(s_1,t_1)\Rightarrow R_2^{\text{may}}(s_2,t_2)\wedge\rho(t_1,t_2)$.

给定一个具体的布尔 Kripke 结构 $K=\langle S,R,I\rangle$,一个抽象状态集 $S^\#$,以及对应的具体化函数 $\gamma:S^\#\to 2^S$. 对于逼近 K 的抽象 GKMTS 模型 $K^\#$,除了 must 超转换之外的其他部分可以采用与构造抽象 KMTS 模型类似的方式构造. 而对于 must 超转换,其构造条件如下定义:对于任意的抽象状态 $s^\#\in S^\#$ 和状态集合 $A\subseteq S^\#$,在 $K^\#$ 中存在一个从 $s^\#$ 到 A 的 must 超转换(即 $(s^\#,A)\in R^{\#\text{must}}$)的条件为:

$\forall s\in\gamma(s^\#)\cdot\exists t^\#\in S^\#\cdot\exists t\in\gamma(t^\#)\cdot R(s,t)$,($\forall\exists\exists$ rule).

文献[77]证明,如果抽象状态集对应于具体状态空间的一个等价划分,那么上述方法构造的抽象 GKMTS 模型 $K^\#$ 是对具体模型 K 的最精确逼近.

状态间的超转换概念首先由 Larsen 等人[78]引入,之后 Shoham 等人[30]使用超转换来避免抽象模型构造过程中的精度损失. de Alfaro 等人[76]类似地使用超转换来处理由 must 转换引起的不精确,并且提出了抽象转换结构(abstract transition system, ATS)模型. ATS 基于博弈理论采用 3 值逻辑定义了两名博弈者之间的博弈结构,而 ATS 之间的逼近关系则通过对博弈者获胜策略(winning strategy)的保持关系来定义.

3.3.2 HTS

在 GKMTS 的基础之上,Shoham 等人[30,77]提出了一种表现形式更丰富的抽象模型结构——HTS(hyper transition system). HTS 通过添加 may 超转换对 GKMTS 进行扩展. 使用 may 超转换的目的是为了提高对全局属性抽象模型检测的精度. 例如:对于图 7(b)所示的布尔 Kripke 结构 K_{ex2},假设有抽象状态 a_1,a_{23},a_{34},分别对应具体状态集合 $\{c_1\},\{c_2,c_3\},\{c_3,c_4\}$. 在这些抽象状态上构造抽象 GKMTS 模型(如图 7(c)所示). 考虑属性 $\Box p$ 和 $\Box q$(注意与 $\Box p$ 和 $\Box q$ 等价的公式分别为 $\neg\Diamond\neg p$ 和 $\neg\Diamond\neg q$),它们在 K_{ex2} 中状态 c_1 上的值都为真,但是在抽象 GKMTS 模型中对应的状态 a_1 上的值为未知. 假如我们试图通过除掉不确定的 may 转换来提高精度;那么,如果删除 a_1 到 a_{23} 之间的 may 转换,将使属性 $\Box q$ 获得为真的确定结果,但是 $\Box p$ 仍然是未知;而如果删除 a_1 到 a_{34} 之间的 may 转换,将使属性 $\Box p$ 获得为真的确定结果,但是 $\Box q$ 仍是未知;也就是说,在 GKMTS 的框架内,我们无法构造出能够同时对 $\Box p$ 和 $\Box q$ 作出确定检测结果的抽象模型. 超转换系统就是为解决这个问题而提出来的.

一个 HTS 模型 $K=\langle S,R^{\text{must}},R^{\text{may}},L\rangle$ 中的各部分与 GKMTS 中的定义相同,除了允许 may 超转换,即 $R^{\text{may}}\subseteq S\times 2^S$. HTS 之间的逼近关系可以使用超混合模拟(hyper mixed simulation)[77]进行结构上的刻画. 超混合模拟将广义混合模拟扩展到 may 超转换上:如果相对于超混合模拟 ρ 所对应的序关系,K_2 比 K_1 小,即 $K_1\preceq_\rho K_2$,那么 K_2 比 K_1 含有更少的 must 超转换和更多的 may 超转换;也就是说,K_2 比 K_1 包含更多不确定的信息,精确度相对较低.

给定一个表示具体模型的布尔 Kripke 结构 K,抽象状态集合 $S^\#$ 和具体化函数 γ,对于逼近 K 的抽象 HTS 模型 $K^\#$,除了 may 超转换之外的其他部分可以采用与构造抽象 GKMTS 类似的方式进行构造. 而对于 may 超转换 $R^{\#\text{may}}$,其构造条件如下定义:对于任意的状态 $s^\#\in S^\#$ 和状态集合 $A\subset S^\#$,从 $s^\#$ 到 A 之间存在着一个 may 超转换(即 $(s^\#,A)\in R^{\#\text{may}}$)的条件为:

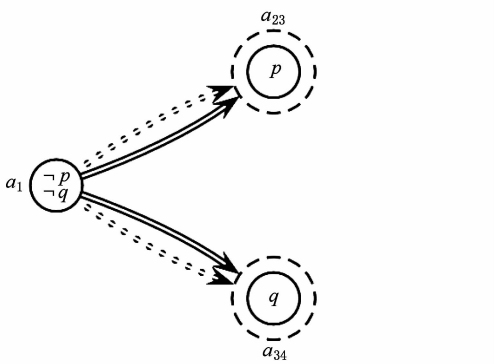
$\forall s\in\gamma(s^\#)\cdot\forall t\in S\cdot\exists t^\#\in A\cdot R(s,t)\Rightarrow t\in\gamma(t^\#)$,($\forall\forall\exists$ rule).

这个条件表明,从 $s^\#$ 出发的一个 may 超转换是对从 $s^\#$ 所对应的具体状态出发的所有转换关系的自上逼近. 与构造最优 MixTS 模型类似,如果要求 must 超转换和 may 超转换都只作用在最小的状态集合上,那么可以得到最优的 HTS 模型;也就是说,对于 $s^\#\in S^\#$,$A_1,A_2\subseteq S^\#$,并且 $A_1\subset A_2$,如果根据构造条件允许具有同样类型的超转换 $s\to A_1$ 和 $s\to A_2$ 存在,那么在最优的模型中我们只保留 $s\to A_1$.

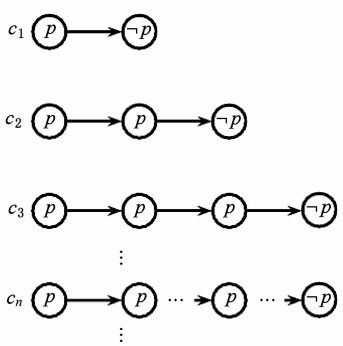
L_μ 在满足一致性的 HTS 上的语义定义与式(3)类似,但存在操作符◇关于 may 转换部分改为在 may 超转换上进行定义. 注意由于从一个状态出发的一个 may 超转换是对从该状态所对应的具体状态出发的所有转换关系的自上逼近,因此有:

$$\|\Diamond\varphi\|(s)=\begin{cases}\mathrm{t}, & \text{当}\exists A\subseteq S\cdot R^{\mathrm{must}}(s,A)\wedge \bigwedge\limits_{t\in A}\|\varphi\|(t)=\mathrm{t};\\ \mathrm{f}, & \text{当}\exists A\subseteq S\cdot R^{\mathrm{may}}(s,A)\wedge \bigwedge\limits_{t\in A}\|\varphi\|(t)=\mathrm{f};\\ \bot, & \text{其他}.\end{cases}\quad(4)$$

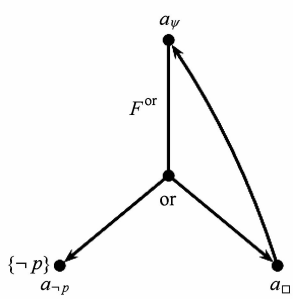
例如,对于图 7(b)中的 K_{ex2} 以及抽象状态集 $\{a_1,a_{23},a_{34}\}$,可以构造如图 8(a)所示的抽象 HTS 模型;在此模型上,对于属性$\Box p$, 考虑 $\psi=\neg\Box p=\Diamond\neg p$. 根据式(4)可知:$\|\Diamond\neg p\|(a_1)=\mathrm{f}$. 因此,$\Box p$ 在 a_1 上为真;同样,在此模型中,可以判断$\Box q$ 在 a_1 上也为真. 也就是说,通过采用 HTS,可以构造出同时能够对属性$\Box p$ 和$\Box q$ 得到确定检测结果的抽象模型,获得比 GKMTS 更高的精度.



(a) An abstract HTS of K_{ex2}



(b) A boolean Kripke structure K_{ex3}



(c) The maximal FTS model of the property AF$\neg p$

must hyper-transitions
may hyper-transitions

Fig. 8　HTS abstraction and FTS model.

图 8　HTS 抽象模型与 FTS 模型

事实上,在文献[30,77]中,Shoham 等人证明,对于给定的一个抽象状态空间,如果它足够精确,允许通过对子公式语义归纳计算的方式得到对逻辑公式的确定分析结果,那么就可以在该状态空间上构造一个抽象 HTS 模型,使得在该模型上能够得到对逻辑公式的确定检测结果. 也就是说,HTS 抽象框架相对于给定的抽象状态空间能够提供最精确的抽象.

3.3.3　局限性

虽然通过采用超转换可以获得具有更高抽象精度的模型,但是超转换也导致状态转换关系变得复杂,使得其在实际应用上受到限制. 例如,在与符号模型检测的结合方面,由于符号模型检测用布尔变量表示模型的状态,并通过特征函数来表示状态间的二元转换关系,因此这种方法可以拓展到 MixTS 抽象模型上:通过用多值逻辑来标注转换关系的值,或者用传统的方法对"可能的"和"必然的"转换关系分别进行表示. 但是对于超转换关系,由于它表示的是从一个状态到一组状态的非确定转换关系,所以直接用上面的二元关系进行表示存在困难. 例如:图 7(a)中所示的 a_{12} 到集合$\{a_3,a_4\}$的 must 超转换关系表示从状态 a_{12} 可以非确定地选择到达 a_3 或者 a_4,但是并不保证两者必须同时可以到达. 如果采用定义在状态空间上的二元关系直接将这个超转换关系表示为 $a_{12}\wedge(a_3\vee a_4)$,由于这个公式与$(a_{12}\wedge a_3)\vee(a_{12}\vee a_4)$等价,因此它实际上表示的是从状态 a_{12}出发必须同时可以到达 a_3 和 a_4,这与原来的超转换关系所表示的含义并不一致. 因此直接采用现有的符号模型检测对超转换系统进行验证并不适合,这也使得其在实际中的应用受到限制.

4　完备性

第 3 节讨论了各种不同表现形式的抽象模型、相应的抽象框架、以及与模型检测精度相关的内容,本节介绍抽象模型的完备性. 与抽象精度关注如何设计能够验证更多属性的抽象模型不同,完备性关注在某种抽象框架内是否对于任意的逻辑公式都存在一个能够对其进行确定分析的有限抽象模型.

文献[33]研究了线性时间属性(如 LTL 所表示的时序逻辑属性)的完备性,证明必须通过在抽象模型中引入公平性约束(fairness constraints)才可获得完备性.对于分支时间属性(如 CTL,μ 演算所表示的时序逻辑属性),Namjoshi[75] 证明为了保证抽象框架的完备性,除了公平性约束之外,超转换也是必要的.因此,之前所介绍的抽象模型,包括最精确的抽象模型 HTS 在内,由于没有考虑公平性约束,所以都是不完备的.通过一个例子来说明公平性约束的必要性.考虑图 8(b)所示的具体模型 K_{ex3},它有无穷个初始状态,并且每一个初始状态 c_i 都可以通过 i 步到达一个 p 为假的状态.假设要检查属性 ψ:AF $\neg p$.显然,ψ 在 K_{ex3} 上可以成立.然而,若试图基于有限的抽象状态集对 K_{ex3} 构造一个自上逼近的抽象模型,那么任何这样的模型中都将存在一个 p 总是为真的环路,即存在着一个使得 EGp 为真的路径,因此无法得到对 ψ 的确定验证结果.例如,假设使用两个抽象状态 a_p 和 $a_{\neg p}$ 分别对应 K_{ex3} 中所有 p 为真和假的状态,那么根据存在抽象,所得到的抽象模型中有 2 个状态转换:$a_p \to a_p$ 和 $a_p \to a_{\neg p}$,而 a_p 到其自身的环路就是使得 EGp 为真的路径.为此,需要增加要求 p 为真的状态不能无限多次出现的公平性约束,将这种环路排除掉,从而得到 ψ 为真的检测结果.

在文献[75]的基础上,Dams 和 Namjoshi 提出了聚焦转换系统(focused transition systems, FTS),并以此为模型对分支时间属性定义了完备的抽象框架[34].聚焦转换系统在文献[35]中进一步被泛化为树自动机(tree automata).具体而言,一个 FTS 模型是一个七元组:$K=\langle S, R^{must}, R^{may}, F^{or}, F^{and}, I, \Phi\rangle$,其在 MixTS 模型基础上增加了 focus 关系 $F^{or}\subseteq S\times 2^S$, defocus 关系 $F^{and}\subseteq S\times 2^S$,以及公平性约束 Φ.对于状态集合 $V\subseteq S$,如果 $V\in F^{or}(s)$,则 V 中所有状态表示的信息的并集与 S 表示的信息相同,也就是说,可以将 V 看作对 S 的一个分割.例如,假设有状态 a, a_1, a_2 分别表示"x 是正整数","x 是正奇数"和"x 是正偶数",可定义 $\{a\}$ 和 $\{a_1, a_2\}$ 均为 $F^{or}(a)$ 中的元素.另一方面,如果 $V\in F^{and}(s)$,则 V 中每个状态表示的信息都包含 S 表示的信息,也就是说,V 中每个状态都比 S 更抽象.例如,假设有状态 a_3 和 a_4,分别表示"x 是整数"和"x 是正数",可定义 $\{a, a_1, a_2\}$ 的每一个非空子集均为 $F^{or}(a)$ 中的元素.

K 上的时序逻辑公式 φ 的可满足性根据基于 K 定义的 3 值博弈结构和基于 φ 定义的交错树自动机(alternating tree automata)A_φ 进行判断.在博弈的过程中,一方试图证明 A_φ 中状态所表示的时序属性在 K 的初始状态上成立,这可以归结为判断 A_φ 中的状态 q 所表示的 φ 的子公式在 K 中的状态 s 上是否成立.直观上,如果 q 表示析取子公式,则对 s 应用 focus 关系,进行 focus 操作,将 s 分割为多个子状态,即 $F^{or}(s)$ 中的某个状态集合 V;由于 s 对应于这些子状态的并集,通过证明 $F^{or}(s)$ 中每个子状态都满足 q 的某个析取项,从而证明 q 在 s 上成立.例如,假设 q 对应析取公式 $\varphi_q=r_1\vee r_2$,对于上述例子中的状态 a,为了判断 φ_q 是否在 a 上成立,可采 focus 操作,考虑集合 $\{a_1, a_2\}$.如果有 r_1 在 a_1 上成立,以及 r_2 在 a_2 上成立,则可证明 φ_q 在 a 上成立.相应地,当 q 表示合取子公式时,则对 s 应用 defocus 关系,进行 defocus 操作,将 s 泛化到一个更加抽象的状态集上,即 $F^{and}(s)$,其中的每个抽象状态都是对 s 的逼近,因此 s 对应于这些状态的交集;通过证明 q 中的每个合取项都能够被 $F^{and}(s)$ 中的某个状态满足,从而证明 q 能够被 s 满足.例如,假设 q 对应析取公式 $\varphi_q=r_1\wedge r_2$,对于状态 a,为了判断 φ_q 是否在 a 上成立,可采用 defocus 操作,考虑集合 $\{a_3, a_4\}$ 如果有 r_1 在 a_3 上成立,以及 r_2 在 a_3 上成立,可则证明 φ_q 在 a 上成立.由此可以看出,focus 和 defocus 关系恰好分别对应 HTS 中的 must 和 may 超转换关系.文献[34]进一步证明对于任何分支时间属性 φ,都存在一个 FTS 模型 K_φ 作为其最大模型(maximal model),即 K_φ 逼近所有满足这个属性的模型,同时它自身也满足 φ.例如,对于属性 ψ:AF $\neg p$,其对应的 FTS 模型 K_φ 如图 8(c)所示,其中所有的转换都是 may 转换,并且对于 a_ψ 有:$\{a_{\neg p}, a_\square\}\in F^{or}(a_\psi)$.为了证明 ψ 在 a_ψ 上为真,注意 ψ 和 L_μ 公式 $\mu X\cdot \neg p\vee\square X$ 等价,为此我们将 a_ψ 分割为 $a_{\neg p}$ 和 $a_\square$.显然,ψ 在 $a_{\neg p}$ 上成立(用 $\neg p$ 进行标记);此外,通过公平性约束排除 a_ψ 和 $a_\square$ 无限次出现的路径,可以证明 ψ 在 $a_\square$ 上也成立;因此,ψ 在 a_ψ 上成立.事实上,ψ 的最大 FTS 模型本质上就是从 ψ 转换而来的交错树自动机.

5 总结与展望

本文以抽象解释作为系统分析抽象方法的理论基础,介绍了软件模型检测中抽象的各个组成部分,重点对近年来提出的各种抽象模型进行讨论,不仅包括传统的用于支持证实或者证伪的模型,也包括

同时支持证实与证伪的模型.对每一类模型,介绍了它们所对应的逼近关系,以及对逼近关系的刻画,分析了相应的最优抽象模型的结构定义条件,阐述了其局限性,以突出新的模型产生的意义.最后讨论了抽象模型的完备性研究结果.

抽象模型是软件模型检测中抽象研究的重要部分,在理论和应用上已取得许多进展,在未来研究中,以下4个方面值得关注:

1) 对于超转换系统,3.3.3节提到由于超转换关系难以直接用符号化方法表示,因此与符号化模型检测的结合方面存在困难.如何获得具备与超转换系统相同精度的抽象符号模型检测是个有意义的问题.在对抽象模型的完备性的讨论中可以看出,超转换与focus和defocus操作是等价的.因此,可以通过考虑构造等价于这些操作的模型检测过程(如图3中Ⅳ所示),获得更精确的验证结果.文献[72-73]在MixTS模型上对μ演算定义了一种新的对应于focus操作的归纳语义,具有与包含must超转换的GKMTS模型相应的精度,并且以此为基础,在MixTS模型上实现了更精确的符号模型检测.在未来需要继续研究如何实现与defocus等价的支持符号模型检测的分析过程,以获得与HTS模型相同的精度.

2) 在3值抽象模型上,除了归纳语义,Bruns和Godefroid对时序逻辑还定义了一种完全语义(thorough semantics)[71,79]:在这种情况下,一个时序逻辑公式φ在抽象模型$M^{\#}$上的值为真(或假),当且仅当φ在所有与$M^{\#}$对应的具体模型M上的值都为真(或假);仅当存在2个具体模型M_1和M_2,使得φ在M_1和M_2上分别为真和假时,φ在$M^{\#}$上的值为未知.与归纳语义相比,完全语义能够对更多的公式提供确定的验证结果.但是,计算完全语义等价于对公式的可满足性判断,其计算复杂度更高.为此,文献[80-81]研究了在形式上被称为自最小化(self-minimizing)的时序逻辑公式,这类公式在归纳语义和完全语义下的模型检测结果一致.通过一种自最小化过程,每一个μ演算公式都可以被转化为一个自极小化公式.文献[82]将此类结果进一步扩展到GKMTS模型上.如何将结果扩展到HTS模型上还需要进一步研究.

3) 结合自上逼近和自下逼近的抽象能够同时提供对属性的验证和对错误的检测.如何避免传统方法中构建抽象模型的昂贵代价,在实际中有效地实现此类抽象分析是近年来软件自动验证领域重点关注的一个问题.文献[83]提出综合动态和静态分析进行软件模型检测,建议采用测试或符号执行等技术来搜索程序状态空间,同时对已搜索的状态空间进行抽象分析.文献[84-85]采用测试作为对程序的自下逼近来检测错误,并利用自上逼近的抽象来判断正确性;其中测试结果被用来引导对抽象模型的精化,而抽象分析的结果则被用来帮助生成新的测试用例.这种方法进一步在文献[86]中被用于对过程间程序(interprocedural programs)的验证.对于并发程序,文献[58]采用测试与谓词抽象的方法进行软件验证;在文献[59]中,通过采用符号执行提供了对包含任意初始状态和非确定输入的并发程序的抽象分析.最近,文献[87-88]实现了通过逐步搜索程序路径进行自下逼近以及采用插值进行自上逼近的程序验证工具.文献[89]则以上下文无关文法所定义的语言为基础,提出了对递归程序中程序总结(procedure summary)的自下逼近的计算方法.对这些工作加以深入分析,并有效实现综合自上逼近和自下逼近的分析与其他验证方式(如组合验证[90-92])的结合,是今后关于抽象研究的一个重点.

4) 对于模型检测的新的应用领域,需要在现有模型的基础上设计新的抽象模型.例如,文献[93-94]对于3VKS等价的模态转换系统(modal transition system,MTS)模型进行了扩展,允许在状态转换关系上标注整数区间值,用于对系统中的转换时间和代价等量化信息进行建模和分析.为了对非确定和随机系统进行抽象验证,文献[95]将区间Markov链(interval Markov chains)与MTS相结合,提出了为交互Markov链(interactive Markov chains)构建抽象模型的方法;文献[96]则针对概率自动机(probabilistic automata)定义了抽象概率自动机作为抽象模型,其本质上可以看作是约束Markov链(constraint Markov chains)与MTS的结合.为了对混成系统上的μ演算属性进行验证,文献[97]对混成自动机定义了包含may和must状态转换关系的抽象模型,并且提供了描述属性保持的模拟关系.随着模型检测在实际中的不断应用,对于新的抽象模型的研究必将会继续深入开展,是未来工作的一个重要方向.

参　考　文　献

[1] Clarke E M, Grumberg O, Peled D. Model Checking [M]. Cambridge: MIT Press, 1999

[2] Baier C, Katoen J P. Principles of Model Checking [M]. Cambridge: MIT Press, 2008

[3] Lin Huimin, Zhang Wenhui. Model checking: Theories, techniques and applications [J]. Acta Electronica Sinica, 2002, 30(12): 1907-1912 (in Chinese)
(林惠民, 张文辉. 模型检测: 理论,方法与应用[J]. 电子学报, 2002, 30(12): 1907-1912)

[4] Clarke E M. My 27-year quest to overcome the state explosion problem [C] //Proc of the 24th Annual IEEE Symp on Logic in Computer Science (LICS'09). Piscataway, NJ: IEEE, 2009: No. 3

[5] Ball T, Podelski A, Rajamani S . Boolean and Cartesian abstraction for model checking C programs [G] //LNCS 2031: Proc of the 7th Int Conf on Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer, 2001: 268-283

[6] Ball T, Levin V, Rajamani S K. A decade of software model checking with SLAM [J]. Communications of the ACM, 2011, 54(7): 68-76

[7] Henzinger T A, Jhala R, Majumdar R, et al. Lazy abstraction [C] //Proc of 29th SIGPLANSIGACT Symp POPL'02. New York: ACM, 2002: 301-306

[8] Beyer D, Henzinger T A, Jhala R, et al. The software model checker Blast [J]. International Journal on Software Tools for Technology Transfer, 2007, 9(5/6): 505-525

[9] Clarke E, Kroening D, Sharygina N, et al. SATABS: SAT-based predicate abstraction for ANSI-C [G] //LNCS 3440: Proc of the 11th Int Conf on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'05). Berlin: Springer, 2005: 570-574

[10] Donaldson A F, Kaiser A, Kroening D, et al. Counterexample-guided abstraction refinement for symmetric concurrent programs [J]. Formal Methods in System Design, 2012, 41(1): 25-44

[11] Păsăreanu C S, Dwyer M B, Visser W. Finding feasible abstract counter-examples [J]. International Journal on Software Tools for Technology Transfer, 2003, 5(1): 34-48

[12] Giannakopoulou D, Păsăreanu C S. Interface generation and compositional verification in JavaPathfinder [G] //LNCS 5503: Proc of the 12th Int Conf on Fundamental Approaches to Software Engineering (FASE'09). Berlin: Springer, 2009: 94-108

[13] Ivančić F, Yang Z, Ganai M K, et al. F-Soft: Software verification platform [G] //LNCS 3576: Proc of 17th Int Conf on Computer-Aided Verification (CAV'05). Berlin: Springer, 2005: 301-306

[14] Qu Wanxia, Li Tun, Guo Yang, et al. Advances in predicate abstraction [J]. Journal of Software, 2008, 19(1): 27-38 (in Chinese)
(屈婉霞, 李暾, 郭阳, 等. 谓词抽象技术研究[J]. 软件学报, 2008, 19(1): 27-38)

[15] Dams D. Comparing abstraction refinement algorithms [J]. Electronic Notes in Theoretical Computer Science, 2003, 89(3): 405-416

[16] Cousot P, Cousot R. Abstract interpretation frameworks [J]. Journal of Logic and Computation, 1992, 2(4): 511-547

[17] Cousot P, Cousot R, Mauborgne L. Theories, solvers and static analysis by abstract interpretation [J]. Journal of the ACM (JACM), 2012, 59(6): 1-56

[18] Li Mengjun, Li Zhoujun, Chen Huowang. Program verification techniques based on abstract interpretation theory [J]. Journal of Software, 2008, 19(1): 17-26 (in Chinese)
(李梦君, 李舟军, 陈火旺. 基于抽象解释理论的程序验证技术[J]. 软件学报, 2008, 19(1): 17-26)

[19] Fitting M. Bilattices are nice things [C] //Proc of Conf on Self-reference. Stanford, California: Center for the Study of Language and Information (CSLI), 2002: 53-77

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

[21] Godefroid P, Jagadeesan R. On the expressiveness of 3-valued models [G] //LNCS 2575: Proc of 4th Int Conf on Verification, Model Checking, and Abstract Interpretation (VMCAI'03). Berlin: Springer, 2003: 206-222

[22] Gurfinkel A, Wei O, Chechik M. Systematic construction of abstractions for model-checking [G] //LNCS 3855: Proc of 7th Int Conf on Verification, Model Checking, and Abstract Interpretation (VMCAI'06). Berlin: Springer, 2006: 381-397

[23] Loiseaux C, Graf S, Sifakis J, et al. Property preserving abstractions for the verification of concurrent systems [J]. Formal Methods in System Design, 1995, 6(1): 11-44

[24] Chen Liqian, Wang Ji, Hou Suning. An abstract domain of one-variable interval linear inequalities [J]. Chinese Journal of Computers, 2010, 33(3): 427-439 (in Chinese)
(陈立前, 王戟, 侯苏宁. 单变量区间线性不等式抽象域[J]. 计算机学报, 2010, 33(3): 427-439)

[25] Cousot P. Abstract interpretation based formal methods and future challenges [C] //Proc of Informatics-10 Years Back, 10 Years Head. Berlin: Springer, 2001: 138-156

[26] Clarke E M, Grumberg O, Long D E. Model checking and abstraction [J]. ACM Trans on Programming Languages and Systems (TOPLAS), 1994, 16(5): 1512-1542

[27] Cleaveland R, Iyer P, Yankelevich D. Optimality in abstractions of model checking [G] //LNCS 983: Proc of Int Symp on Static Analysis(SAS'95). Berlin: Springer, 1995: 51-63

[28] Dams D, Gerth R, Grumberg O. Abstract interpretation of reactive systems [J]. ACM Trans on Programming Languages and Systems (TOPLAS), 1997, 19(2): 253-291

[29] Shoham S, Grumberg O. Monotonic abstraction-refinement for CTL [G] //LNCS 2988: Proc of the 10th Int Conf on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04). Berlin: Springer, 2004: 546-560

[30] Sharon S, Grumberg O. 3-valued abstraction: More precision at less cost [C] //Proc of the 21st IEEE Symp on Logic in Computer Science(LICS'06). Piscataway, NJ: IEEE, 2006: 399-410

[31] Lamport L. Specifying concurrent program modules [J]. ACM Trans on Programming Languages and Systems (TOPLAS), 1983, 5(2): 190-222

[32] Schmidt D A. Closed and logical relations for over-and under-approximation of powersets [G] //LNCS 3148: Proc of the 11th Int Symp on Static Analysis (SAS'04). Berlin: Springer, 2004: 22-37

[33] Kesten Y, Pnueli A. Verification by augmented finitary abstraction [J]. Information and Computation, 2000, 163 (1): 203-243

[34] Dams D, Namjoshi K S. The existence of finite abstractions for branching time model checking [C] // Proc of the 19th Annual IEEE Symp on Logic in Computer Science (LICS'04). Piscataway, NJ: IEEE, 2004: 335-344

[35] Dams D, Namjoshi K S. Automata as abstractions [G] // LNCS 3385: Proc of the 6th Int Conf on Verification, Model Checking, and Abstract Interpretation (VMCAI'05). Berlin: Springer, 2005: 216-232

[36] Graf S, Saïdi H. Construction of abstract state graphs with PVS [G] //LNCS 1254: Proc of the 9th Int Conf on Computer-Aided Verification (CAV'97). Berlin: Springer, 1997: 72-83

[37] Chaki S, Clarke E M, Groce A, et al. Modular verification of software components in C [J]. IEEE Trans on Software Engineering, 2004, 30(6): 388-402

[38] Gurfinkel A, Chechik M. Why waste a perfectly good abstraction? [G] //LNCS 3920: Proc of the 12th Int Conf on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06). Berlin : Springer, 2006: 212-226

[39] Clarke E M, Grumberg O, Talupur M, et al. Making predicate abstraction efficient: How to eliminate redundant predicates [G] //LNCS 2725: Proc of the 15th Int Conf on Computer-Aided Verification (CAV'03). Berlin: Springer, 2003: 126-140

[40] Gurfinkel A, Wei O, Chechik M. Model checking recursive programs with exact predicate abstraction [G] //LNCS 5311: Proc of the 6th Int Symp on Automated Technology for Verification and Analysis (ATVA'08). Berlin: Springer, 2008: 95-110

[41] Clarke E, Grumberg O, Jha S, et al. Counterexample-guided abstraction refinement for symbolic model checking [J]. Journal of the ACM (JACM), 2003, 50(5): 752-794

[42] Ball T. Formalizing counterexample-driven refinement with weakest preconditions [C] //Proc of the NATO Advanced Study Institute on Engineering Theories of Software Intensive Systems. Berlin: Springer, 2005:42-68

[43] Lakhnech Y, Bensalem S, Berezin S, et al. Incremental Verification by Abstraction [G] //LNCS 2031: Proc of the 7th Int Conf on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01). Berlin: Springer, 2001: 98-112

[44] McMillan K L. Lazy abstraction with interpolants [G] // LNCS 4144: Proc of the 18th Int Conf on Computer-Aided Verification (CAV'06). Berlin: Springer, 2006: 123-136

[45] Chauhan P, Clarke E, Kukula J, et al. Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis [G] //LNCS 2517: Proc of the 4th Int Conf on Formal Methods in Computer-Aided Design (FMCAD'02). Berlin: Springer, 2002: 33-51

[46] Clarke E M, Gupta A, Strichman O. SAT-based counterexample-guided abstraction refinement [J]. IEEE Trans on Computer-Aided Design of Integrated Circuits and Systems, 2004, 23(7): 1113-1123

[47] He F, Song X, Hung W N N, et al. Integrating evolutionary computation with abstraction refinement for model checking [J]. IEEE Trans on Computers, 2010, 59(1): 116-126

[48] Tian C, Duan Z, Zhang N. An efficient approach for abstraction-refinement in model checking [J]. Theoretical Computer Science, 2012, 461(7): 76-85

[49] Cousot P, Ganty P, Raskin J F. Fixpoint-guided abstraction refinements [G] //LNCS 4634: Proc of the 14th Int Symp on Static Analysis(SAS'07). Berlin: Springer, 2007: 333-348

[50] Ranzato F, Doria O R, Tapparo F. A forward-backward abstraction refinement algorithm [G] //LNCS 4905: Proc of the 9th Int Conf on Verification, Model Checking, and Abstract Interpretation (VMCAI'08). Berlin: Springer, 2008: 248-262

[51] D'Silva V, Purandare M, Kroening D. Approximation refinement for interpolation-based model checking [G] // LNCS 4905: Proc of the 9th Int Conf on Verification, Model Checking, and Abstract Interpretation (VMCAI'08). Berlin: Springer, 2008: 68-82

[52] Giacobazzi R, Quintarelli E. Incompleteness, counterexamples, and refinements in abstract model-checking [G] //LNCS 2126: Proc of the 8th Int Symp on Static Analysis (SAS'01). Berlin: Springer, 2001: 356-373

[53] Cook B, Podelski A, Rybalchenko A. Terminator: Beyond safety [G] //LNCS 4144: Proc of the 18th Int Conf on Computer-Aided Verification (CAV'06). Berlin: Springer, 2006: 415-418

[54] Cook B, Podelski A, Rybalchenko A. Proving program termination [J]. Communications of the ACM, 2011, 54 (5): 88-98

[55] Grumberg O. Abstraction and refinement in model checking [G] //LNCS 4111: Proc of the 4th Int Symp on Formal Methods for Components and Objects (FMCO'05). Berlin: Springer, 2006: 219-242

[56] Grumberg O. 2-Valued and 3-Valued abstraction refinement in model checking [J]. Logics and Languages for Reliability and Security, 2010, 25(1): 105-128

[57] Yorsh G, Ball T, Sagiv M. Testing, abstraction, theorem proving: Better together! [C] //Proc of the ACM/SIGSOFT Int Symp on Software Testing and Analysis (ISSTA'06). New York: ACM, 2006: 145-156

[58] Pasareanu C S, Pelánek R, Visser W. Predicate abstraction with under-approximation refinement [J]. Logical Method in Computer Science, 2007, 3(1): 1-22

[59] Albarghouthi A, Gurfinkel A, Wei O, et al. Abstract analysis of symbolic executions [G] //LNCS 6174: Proc of the 22nd Int Conf on Computer-Aided Verification (CAV'10). Berlin: Springer, 2010: 495-510

[60] Barner S, Grumberg O. Combining symmetry reduction and under-approximation for symbolic model checking [J]. Formal Methods in System Design, 2005, 27(1/2): 29-66

[61] Ball T, Kupferman O, Sagiv M. Leaping loops in the presence of abstraction [G] //LNCS 4590: Proc of the 19th Int Conf on Computer-Aided Verification (CAV'07). Berlin: Springer, 2007: 491-503

[62] Ball T, Kupferman O. Better under-approximation of programs by hiding variables [G] //LNCS 4349: Proc of the 8th Int Conf on Verification, Model Checking, and Abstract Interpretation (VMCAI'07). Berlin: Springer, 2007: 314-328

[63] Clarke E, Kroening D, Lerda F. A tool for checking ANSI-C programs [G] //LNCS 2988: Proc of the 10th Int Conf on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04). Berlin: Springer, 2004: 168-176

[64] Kroening D, Ouaknine J, Strichman O, et al. Linear completeness thresholds for bounded model checking [G] //LNCS 6806: Proc of the 23rd Int Conf on Computer-Aided Verification (CAV'11). Berlin: Springer, 2011: 557-572

[65] Wei O, Gurfinkel A, Chechik M. Identification and counter abstraction for full virtual symmetry [G] //LNCS 3725: Proc of the 13th Advanced Research Working Conf on Correct Hardware Design and Verfication Methods (CHARME'05). Berlin: Springer, 2005: 285-300

[66] Huth M, Jagadeesan R, Schmidt D. Modal transition systems: A foundation for three-valued program analysis [G] //LNCS 2028: Proc of the 10th European Symp on Programming (ESOP'01). Berlin: Springer, 2001: 155-169

[67] Godefroid P, Huth M, Jagadeesan R. Abstraction-based model checking using modal transition systems [G] //LNCS 2154: Proc of the 12th Int Conf on Concurrency Theory (CONCUR'01). Berlin: Springer, 2001: 426-440

[68] Larsen K G, Thomsen B. A modal process logic [C] //Proc of the 3rd Annual Symp on Logic in Computer Science (LICS'88). Piscataway, NJ: IEEE, 1988: 203-210

[69] Beneš N, Křetínský J, Larsen K G, et al. EXPTIME-completeness of thorough refinement on modal transition systems [J]. Information and Computation, 2012, 218(1): 54-68

[70] Bruns G, Godefroid P. Model checking partial state spaces with 3-valued temporal logics [G] //LNCS 1877: Proc of the 11th Int Conf on Computer-Aided Verification (CAV'99). Berlin: Springer, 1999: 274-287

[71] Bruns G, Godefroid P. Generalized model checking: Reasoning about partial state spaces [G] //LNCS 1877: Proc of the 11th Int Conf on Concurrency Theory (CONCUR'00). Berlin: Springer, 2000: 168-182

[72] Wei O, Gurfinkel A, Chechik M. On the consistency, expressiveness, and precision of partial modeling formalisms [J]. Information and Computation, 2011, 209(1): 20-47

[73] Wei O, Gurfinkel A, Chechik M. Mixed transition systems revisited [G] //LNCS 5403: Proc of the 10th Int Conf on Verification, Model Checking, and Abstract Interpretation (VMCAI'09). Berlin: Springer, 2009: 349-365

[74] Gurfinkel A, Wei O, Chechik M. Yasm: A software model-checker for verification and refutation [G] //LNCS 4144: Proc of the 18th Int Conf on Computer-Aided Verification (CAV'06). Berlin: Springer, 2006: 170-174

[75] Namjoshi K. Abstraction for branching time properties [G] //LNCS 2725: Proc of the 18th Int Conf on Computer-Aided Verification (CAV'03). Berlin: Springer, 2003: 288-300

[76] de Alfaro L, Godefroid P, Jagadeesan R. Three-valued abstractions of games: Uncertainty, but with precision [C] //Proc of the 19th Annual IEEE Symp on Logic in Computer Science (LICS'04). Piscataway, NJ: IEEE, 2004: 170-179

[77] Shoham S, Grumberg O. 3-valued abstraction: More precision at less cost [J]. Information and Computation, 2008, 206(11): 1313-1333

[78] Larsen K G, Xinxin L. Equation solving using modal transition systems [C] //Proc of the 5th Annual IEEE Symp on Logic in Computer Science (LICS'90). Piscataway, NJ: IEEE, 1990: 108-117

[79] Godefroid P, Piterman N. LTL generalized model checking revisited [J]. International Journal on Software Tools for Technology Transfer, 2011, 13(6): 571-584

[80] Godefroid P, Huth M. Model checking vs generalized model checking: Semantic minimizations for temporal logics [C] //Proc of the 20th IEEE Symp on Logic in Computer Science (LICS'05). Piscataway, NJ: IEEE, 2005: 158-167

[81] Gurfinkel A, Chechik M. How thorough is thorough enough? [G] //LNCS 3725: Proc of the 13th Advanced Research Working Conf on Correct Hardware Design and Verification Methods (CHARME'05). Berlin: Springer, 2005: 65-80

[82] Nejati S, Gheorghiu M, Chechik M. Thorough checking revisited [C] //Proc of the 6th Int Conf on Formal Methods in Computer Aided Design (FMCAD'06). Piscataway, NJ: IEEE, 2006: 106-116

[83] Godefroid P, Klarlund N. Software model checking: Searching for computations in the abstract or the concrete [G] //LNCS 3771: Proc of the 5th Int Conf on Integrated Formal Methods (IFM'05). Berlin: Springer, 2005: 20-32

[84] Gulavani B S, Henzinger T A, Kannan Y, et al. SYNERGY: A new algorithm for property checking [C] // Proc of the 14th ACM SIGSOFT Int Symp on Foundations of Software Engineering (FSE'05). New York: ACM, 2006: 117-127

[85] Beckman N E, Nori A V, Rajamani S K, et al. Proofs from tests [C] //Proc of the ACM/SIGSOFT Int Symp on Software Testing and Analysis (ISSTA'08). New York: ACM, 2008: 3-14

[86] Godefroid P, Nori A V, Rajamani S K, et al. Compositional may-must program analysis: Unleashing the power of alternation [C] //Proc of the 37th ACM SIGPLAN-SIGACT Symp on Principles of Programming Language. New York: ACM, 2010, 45(1): 43-56

[87] Albarghouthi A, Gurfinkel A, Chechik M. From under-approximations to over-approximations and back [G] //LNCS 7214: Proc of the 18th Int Conf on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'12). Berlin: Springer, 2012: 157-172

[88] Albarghouthi A, Gurfinkel A, Chechik M. Whale: An interpolation-based algorithm for inter-procedural verification [G] //LNCS 7148: Proc of the 13th Int Conf on Verification, Model Checking, and Abstract Interpretation (VMCAI'12). Berlin: Springer, 2012: 39-55

[89] Ganty P, Iosif R, Konečný F. Underapproximation of procedure summaries for integer programs [G] //LNCS 7795: Proc of the 19th Int Conf on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13). Berlin: Springer, 2013: 245-259

[90] Pu Fei, Zhang Wenhui. Combining search space partition and abstraction for LTL model checking [J]. Science in China Series: Information Sciences, 2007, 37(12): 1504-1520 (in Chinese)
(蒲飞, 张文辉. 结合搜索空间划分和抽象进行 LTL 模型检测[J]. 中国科学: 技术科学, 2007, 37(12): 1504-1520)

[91] Singh R, Giannakopoulou D, Păsăreanu C. Learning component interfaces with may and must abstractions [G] // LNCS 6174: Proc of the 22nd Int Conf on Computer-Aided Verification(CAV'10). Berlin: Springer, 2010: 527-542

[92] Shoham S, Grumberg O. Compositional verification and 3-valued abstractions join forces [J]. Information and Computation, 2010, 208(2): 178-202

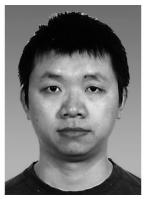
[93] Bauer S S, Fahrenberg U, Juhl L, et al. Quantitative refinement for weighted modal transition systems [G] // LNCS 6907: Proc of the 36th Int Symp on Mathematical Foundations of Computer Science (MSFC'11). Berlin: Springer, 2011: 60-71

[94] Juhl L, Larsen K G, Srba J. Modal transition systems with weight intervals [J]. The Journal of Logic and Algebraic Programming, 2012, 81(4): 408-421

[95] Katoen J P, Klink D, Neuhäußer M R. Compositional abstraction for stochastic systems [G] //LNCS 5813: Proc of the 7th Int Conf on Formal Modeling and Analysis of Timed System (FORMATS'09). Berlin: Springer, 2009: 195-211

[96] Delahaye B, Katoen J P, Larsen K G, et al. Abstract probabilistic automata [G] //LNCS 6538: Proc of the 12th Int Conf on Verification, Model Checking and Abstract Interpretation (VMCAI'11). Berlin: Springer, 2011: 324-339

[97] Bauer K, Gentilini R, Schneider K. A uniform approach to three-valued semantics for μ-calculus on abstractions of hybrid automata [J]. International Journal on Software Tools for Technology Transfer, 2011, 13(3): 273-287



Wei Ou, born in 1974. PhD, associate professor at Nanjing University of Aeronautics and Astronautics. Member of China Computer Federation. His main research interests include software verification and model checking.

Shi Yufeng, born in 1990. Master candidate at Nanjing University of Aeronautics and Astronautics. Student member of China Computer Fderation. Her main research interest is model checking.

Xu Bingfeng, born in 1986. PhD, lecturer at Nanjing Forestry University. Member of China Computer Federation. Her main research interests include embedded systems and formal verification.

Huang Zhiqiu, born in 1965. PhD, professor at Nanjing University of Aeronautics and Astronautics. Senior member of China Computer Federation. His main research interests include software engineering and formal methods.

Chen Zhe, born in 1981. PhD, associate professor at Nanjing University of Aeronautics and Astronautics. Member of China Computer Federation. His main research interests include formal verification and model checking.