模型检验综述

王蓁蓁
(金陵科技学院信息技术学院 南京 211169)

摘 要 在软硬件验证里,模型检验是一个重要手段,至今它已经形成一个庞大的方法论体系。现在我们把模型检验内容从标准方法、抽象解释方法、综合方法3个范畴加以介绍,旨在形成人们对模型检验总的印象,从而全面理解和掌握模型检验各个方法的精神实质和具体情况,有助于将这些方法运用到实际软硬件验证中并从中受到启发,以便进一步发展模型检验理论或开发模型检验新的方法和工具。

关键词 时态逻辑,模型检验,抽象解释,抽象模型检验
中图法分类号 TP311 **文献标识码** A

Survey of Model Checking

WANG Zhen-zhen
(School of Information Technology, Jinling Institute of Technology, Nanjing 211169, China)

Abstract Model checking is an important technique in software/hardware verification, and it has become an enormous system of methodologies. Now we investigated the contents of model checking from three categories which are standard methods, abstract interpretation methods and integrate methods respectively. The aim of this survey is making one to have a complete impression about model checking. In this way one may fully understand and grasp the spirits and concrete contents of every method in model checking. The studies help in applying these methods to the piratical software/hardware verification, and furthermore, inspired of this, one may develop new model checking theories or new methods or tools of model checking.

Keywords Temporal logic, Model checking, Abstract interpretation, Abstract model checking

1 引言

迄今为止,软、硬件的验证技术主要有四大方法体系:模拟、测试、形式证明和模型检验[1]。模拟方法也许是人类最传统的一种验证自己创造的产品功效的方法。即使在最新的验证技术里,仍然看到它渗透其中并发挥它自己的作用。例如在产品寿命试验里,加速寿命试验方法的基本思想是用加大应力(诸如热应力、电应力、机械应力等)的办法,加快产品失效,缩短试验时间;运用加速寿命模型,估计出产品在正常工作应力下的可靠性特征。在软件测验中,驱动程序和桩等技术也是应用模拟思想的典型方法。模拟方法具有很多优点,特别是它们为产品创造了用其他方法难以创造的环境条件。然而它的结论通常具有统计特性,并且依赖于一些假设和理论模型。例如软件可靠性的测量和预测经典地依赖参数或可靠性增长模型(Jelinski-Moranda模型),因此它的准确性需要处理。

测试是一个重要的方法体系,直观地说,测试是预先设计出一些策略、方法、工具和用例,对产品进行试验,看其性能是否符合要求。现今软件已经渗透到我们的日常生活,它的安全性和实用可靠性越来越重要,因为人们越来越依靠计算机。与之发展的软件测试技术也越来越丰富,并且获得很大的成功。但是完全测试是不可能的,通常测试也不能直接指出缺陷存在的位置。

形式证明,例如在软件中的正确性证明,是显示产品正确的一种数学技术,它的一个重要方面是应与设计和编程结合进行,Dijkstra把它表达为"程序员应让程序证明和程序一起发展"。形式证明系统比较著名的是霍尔逻辑,其中证明规则是语法制导的,其中主要精神是把证明一条复合命令的部分正确性断言简化成证明它的直接子命令的部分正确性断言。虽然形式证明系统的优点是它能够自动证明程序的某些性质,但是利用霍尔逻辑即使证明很小的程序的正确性也不是那么容易,例如证明"循环出口条件"和循环语句的不变式。因此,许多软件工程实践者提出,正确性证明不能看成是标准的软件工程技术。他们认为除了证明太难以外,证明也太昂贵了,没有实用性。

模型检验是基于有限状态空间来保证软硬件设计正确性的形式化自动验证技术,基本思想是预先设定软硬件应该具有的规格即属性,而在模型不满足规格时,给出反例,如果满足,通常都给出肯定。上述优点比其他传统方法更为突出,因此日益得到人们的重视,并应用在许多领域中获得了成功。

可惜的是模型检验面临的主要问题是状态空间爆炸问题。可以这样说模型检验技术的演变基本上与解决上述问题

本文受国家自然科学基金重大研究计划项目(91018005),国家自然科学基金项目(61170071),金陵科技学院科研基金资助(jit-b-201207)资助。
王蓁蓁(1975—),女,博士后,主要研究领域为软件测试、程序分析、人工智能,E-mail:wangzhenzhen@seu.edu.cn。

有关。本文主要基于这个线索，对模型检验方法体系进行考察，主要是：

(1)标准方法及对状态空间爆炸问题的解决措施。

(2)抽象方法及对状态空间爆炸问题的解决措施。

(3)综合方法及对状态空间爆炸问题的解决措施。

它们分别构成本文的第 2 节、第 3 节和第 4 节，第 5 节是算法和其他重要方法概览，最后是总结。在以下叙述中，为了不增加读者查阅文献时的负担，当引用文献内容时尽量使符号与所引的文献一致。另外，由于篇幅所限，参考文献只列出本文里较详细介绍的文献，其它文献很容易从所列的文献索引里找到。

2 标准方法

2.1 基本概念

模型检验是一个形式方法，它验证一个系统是否符合它应该要满足的属性。因此必须建立一个能描述系统行为的系统的形式模型，传统的方法是选择 Kripke 结构。至于系统要满足的属性，传统的方法是用时态逻辑的公式。

Kripke 结构 $\mathcal{K}=(\Sigma,R,AP,L)$ 是一个四元组，它由转换系统(Σ,R)、原子命题集合 AP 和标签函数 $L:\Sigma\to\mathcal{P}(AP)$ 组成。其中转换系统(Σ,R)，Σ 表示状态集，$R\subseteq\Sigma\times\Sigma$ 表示转换(关系)，并也用 $\xrightarrow{R}$ 表示，通常假设 R 是“全”关系，即对任意的状态 s，存在状态 t 使得 $s\xrightarrow{R}t$。

• $\mathcal{K}$ 的路径定义为 $path(\mathcal{K})\triangleq\{\pi,\mathbf{N}\to\Sigma\mid\forall i\in\mathbf{N},\pi_i\xrightarrow{R}\pi_{i+1}\}$。

• 转换关系 R 在 $\mathcal{P}(\Sigma)$ 上导出前向/后向转换算子。pre_R，$\widetilde{pre_R}$，$post_R$，$\widetilde{post_R}$。

例如：$pre_R\triangleq\lambda Y.\{s\in\Sigma\mid\exists t\in y.\ s\xrightarrow{R}t\}$

$\widetilde{pre_R}=\lambda Y\cdot\{s\in\Sigma\mid\forall t\in\Sigma.\ s\xrightarrow{R}t\Rightarrow t\in\gamma\}$

$post_R=\lambda Y\cdot\{t\in\Sigma\mid\exists s\in y.\ s\xrightarrow{R}t\}$

$\widetilde{post_R}=\lambda Y\cdot\{t\in\Sigma\mid\forall s\in\Sigma.\ s\to t\Rightarrow s\in\gamma\}$

时态逻辑 CTL*，CTL，LTL。

CTL* 是一个有很强表达力的逻辑。CTL* 的公式是由原子命题、布尔运算符、时态算子和路径量词构成。时态算子包括 G，F，R，U 和 X，路径量词包括 A 和 E。CTL* 公式主要有两种类型：状态公式(其意义是它在某些特定状态下为真)；路径公式(其意义是它沿着某些特定轨道时为真)。状态公式，也可以看作路径公式，而对路径公式 f，Ef，Af 等也是状态公式。总之根据语法规则，CTL* 是状态公式的集合，它是描述反应式系统在状态之间转换序列的形式化方法。

计算树逻辑(也称 branching-time logic)CTL 是 CTL* 的子逻辑，主要限制是时态算子 X，F，G，U 和 R 必须在它们的前面直接加上路径量词。

线性时态逻辑 LTL 也是 CTL* 的另一个子逻辑。它的公式具有形式 Af，其中 f 是路径公式，并且包含在 f 里的子状态公式仅仅是原子命题。

CTL* 逻辑是 CTL 和 LTL 的组合。这些逻辑在模型检验里是最经常运用的。除此以外，还有其他一些变形，例如 ACTL，LTL_X，它们分别是对 CTL、LTL 施加更特殊限制而得到的逻辑。

2.2 标准模型检验步骤

• 首先，将被测系统模型化，即从系统里提取一个 Kripke 结构，并在这个 Kripke 结构增添初始状态集和/或终止状态集。详细地说，$M=(\Sigma,R,AP,L,H,T)$，其中 $H\subset\Sigma$ 是系统的初始状态；(如果有必要的话设置) T 为系统的终止状态集。(Σ,R,AP,L) 就是前面所述的 Kripke 结构。M 称为模型。

• 选定逻辑，并确定系统要满足的公式。通常设计一个系统，总是期望该系统满足一定的属性，即系统应该或不应该具有的行为。通常这些属性和行为由 CTL* 等等时态逻辑的若干公式所表达。这些公式在模型检验里称为指派(specification)或说明或干脆称为属性。今后不加区别地互用这些术语。

• 模型检验。给定模型 M 和指派 f，令

$\{s\in\Sigma\mid M,s\vdash f\}$

一般地说，若 M 的初始状态集 H 包含在上述集合里，则称系统满足指派 f。

• 设计算法。要实现上述模型检验，必须针对逻辑和指派 f 设计算法。Clarke 和 Emerson 在 20 世纪 80 年代初引进了时态逻辑模型检验算法。例如他们开发的 CTL 公式验证算法在由程序决定的模型规模和时态逻辑指派公式的长度上都是多项式时间的(见文献[1] P. 5)。文献[2]提出的 LTL 指派依赖于“Tableau”算法也很有效。正如 CTL* 逻辑是 CTL 和 LTL 的结合，基于 CTL 和 LTL 模型检验结合的“state labeling technique”算法也相应开发出来[1,3,4]。

• 公平约束(fairness constraints)。通常选定的时态逻辑及其指派都无法满足我们对系统的预期要求。因此必须对公式作一些限制，而这些限制是无法用指派的逻辑来表达的。例如 s_0 满足 $E[f_1\ \mathrm{U}\ f_2]$ 的意义是存在一条轨道，$\pi=s_0,s_1,\cdots,s_u,\cdots$，它从 s_0 出发，且在某个 j，$j\geqslant0$，使 $\pi_j\vdash f_2$，$0\leqslant i<j$ 时，$\pi_i\vdash f_1$。但这个公式并不要求“最终”f_2 无限次成立，所以必须增加约束，例如令 $\inf(\pi)=\{s\mid s=s_i\text{ 无限次}\}$，我们要求轨道是公平(fair)，当且仅当要求 $\inf(\pi)\cap\{s\mid s\vdash f_2\}\neq\emptyset$，这里 $\{s\mid s\vdash f_2\}$ 集合就是一个 fairness constraints。因此，把原先的 Kripke 结构提升为 fair Kripke 结构，即在模型 M 里增加 $F=\{P_1,\cdots,P_k\}$ 所谓公平约束的集合(也称 generalized Büchi acceptance conditions)即 $M=(\Sigma,R,AP,L,H,T,F)$，其中 $F\subseteq\mathcal{P}(\Sigma)$ 是公平约束集合，其它符号定义如前。我们说轨道 π 是 fair，当且仅当对 $\forall P\in F$，$\inf(\pi)\cap P\neq\emptyset$。直观上，在 M 里的图里一个强连结成份(strongly connected component) C 是 fair(关于 F)，当且仅当 $\forall P_i\in F$，都存在一个状态 $t\in(C\cap P_i)$。这样就对 M(所形成的图)的轨道属性添加了更多的要求，而这些要求是系统所希望的(见文献[1]P. 33，P. 40)。

• 决策：如果模型满足指派，则做出肯定回答，如果模型不满足指派，则找出反例。

2.3 符号模型检验

符号模型检验是解决状态空间爆炸问题的首次重要突破。对于许多具有并发组件(concurrent parts)的系统，在全局状态转换图里的状态(模型检验主要考察这样的图)往往过于庞大以致难以处理。1987 年 McMillan 认识到用符号表达状态转换图，许多大型系统能够有效验证，为此他开发了称为 SMV 的模型检验系统。自那以后，基于符号模型检验能够解

决的状态数量级已达到 10^{120}（见文献[1] P. 7）。符号模型检验主要概念如下。

• OBDDs(ordered binary decision diagrams)。Bryant's OBDDs 是表达布尔公式的经典形式。在 OBDDs 中，每一个节点都表示一个布尔表达式。该图是由初始节点、中间节点和终止节点构成。对于决定系统状态的变量选择次序，然后依次标记在节点上。而每个不是终点的节点都有两个分支，例如节点 v 中左边分支 low(v)表示 $v=0$ 时的后代，右边分支 high(v)表示 $v=1$ 时的后代。对于终止状态，它表达该变量的值。Bryant 给了一个称为 Apply 的统一算法来计算全部逻辑运算[5]。

• 用 OBDDs 表达 Kripke 结构。

通常都是给系统简洁的高层次的描述，然后直接构造 OBDDs 来表达它，其详细描述见文献[1] P. 58。

• 模型检验算法的设计是基于像集计算(image computation)和谓词转换算子(predicate transformers)的不动点计算。不动点是状态的集合，它们表达时态逻辑的属性。谓词转换算子(从转换系统获得)和不动点两者都由 OBDDs 表达。

• SMV(symbolic model verifier)是一个用来检验有限状态系统满足由 CTL 给出的指派的工具。SMV 的语言具有许多特点，例如 modules(模块性)。并且这些模块可以同步和异步交替组成，也可以确定性或非确定性进行转换，因此它可以有效地描述复杂的有限状态系统(见文献[1] P. 99)。从这种语言写出的程序里，模型检查器抽取转换系统并用 OBDD 表示，并基于 OBDD 的搜索算法去决定这个系统是否满足它所宣称的用时态逻辑表达的指派(见文献[1] P. 9)。另外，VERILOG 也是有限状态语言，用它写的描述系统的程序也能编译为等价的 Kripke 结构。

• 著名的第一次运用形式方式去寻找 IEEE 标准协议非平凡错误的例子。在 IEEE Futurebus + standard (IEEE Standard 896. 1－1991)里描述了 cache coherence protocol，1992 年 Carnegie Mellon 的学者运用 SMV 语言构造了关于上述协议的精确模型，然后运用 SMV 去证明模型产生的转换系统满足 Cache coherence 的形式指派。他们发现了许多以前未发现的错误以及在协议设计中的潜在错误(见文献[1] P. 8)。

其他许多学者在运用符号模型检验方面，都做了大量工作，开发了许多算法并且运用到许多实际应用领域。

2.4 其他减轻问题复杂性的方法

虽然符号模型检验使模型检验能够运用到大型状态转换系统。然而状态空间爆炸问题并未彻底解决，因此学者们又从多个方面去解决这个问题。其中最主要的技术如下。

• 偏序简化(partial order reduction)。例如在 LTL_X 逻辑指派的验证中，从每个状态 s 的 enabled(s)(即在 s 状态可以执行的行动集合)里选择 ample(s)(即在 s 状态挑选出代表性行动)，从而根据 ample(s)把原先的转换图简化，而简化的图与原来系统的转换图在相应路径上满足"stuttering equivalent"。所谓两个路径"丛快"等价，是指两个路径可以分别划出许多段，这些段之间顺序等价。

• 结构间的等价关系和前序关系。若两个结构可以相互模拟，则称两个结构等价，即 bisimulation equivalent。设 $M=(\Sigma,R,AP,L,H)$ 和 $M'=(\Sigma',R',AP,L',H')$ 是两个结构，若存在关系 $B\subseteq\Sigma\times\Sigma'$，满足：$\forall s,s'$，若 $B(s,s')$ 有：

(i) $L(s)=L'(s')$；

(ii) $\forall s_1$，若 $R(s,s_1)$，则存在 s_1'，使得 $R'(s',s_1')$ 且 $B(s_1,s_1')$；

(iii) $\forall s_1'$，若 $R'(s',s_1')$，则存在 s_1，使得 $R(s,s_1)$ 且 $B(s_1,s_1')$；

则称 B 为相互模拟关系。如果 B 对 M 和 M' 的初始集合 H 和 H' 的元素也成立，即 $\forall s_0\in H$，则有 $s_0'\in H'$，使得 $B(s_0,s_0')$ 反过来对 $\forall s_0'\in H'$，也有 $s_0\in H$，使得 $B(s_0,s_0')$，则称 M 和 M' 模拟等价。

比模拟等价关系 B 较弱的关系 H 称为 simulation 关系，即在 B 的条件中第(iii)条件未必成立。这时若对 M 中任一初始状态 s_0，有 M' 中初始状态 s_0'，使得 $H(s_0,s_0')$ 成立，则称 M' 模拟 M，并记之为 $M\leqslant M'$。这样可以通过模拟关系建立结构间的前序关系。利用这些关系往往可以简化原系统，有许多算法是关于等价关系和前序关系的。相互模拟关系适合 CTL* (CTL)等公式的验证，模拟关系适合 ACTL(LTL)等公式的验证。

• 抽象：不涉及抽象解释的抽象，我们也把它归为标准方法。主要是数据结构简化和影响锥简化技术。所谓 cone of influence reduction 技术是指通过选择对于指派(要验证的属性)有关系的变量来简化状态空间。数据简化是指对于指派而言所关心的数据特征对数据结构进行简化。但是许多标准方法的抽象及其改良技术，例如 CEGAR 方法，我们放在抽象解释方法一节里去讨论。

• 组合推理。根据组合系统的构造，例如并行系统，可以把整个系统的验证分割为 n 个子系统的验证。这些部分功能是相互独立的，而且所有部分系统的功能联合起来构成整个系统的功能。其中，最著名的是假设-证明技术。即根据假设去证明(部分)系统应满足的指派，然后去验证这些假设在原系统中成立，即消除假设。关于这个方法在抽象解释里也有介绍。

• 对称：通常系统都具有某种性质的对称性，利用对称性产生的等价关系可以简化原系统到所谓根据对称关系产生的商结构，从而缩减状态空间。

• 归纳方法：往往系统之间的差异只是(例如是组件的数目)"非本质"的属性，即这些系统是具有相似特征的一个家族的成员，甚至这个家族有无限个成员。时态逻辑表达这个家族时往往运用正则表达式。对于有限状态结构的无限成员的家族来说，其验证技术通常依赖于"不变式"。给予家族 $\mathcal{F}=\{M_1,M_2,\cdots\}$，在这些结构之间定义一个前序关系(即反射、传递)$\leqslant$，所谓不变式 I 是如此的结构，它对 $\mathcal{F}$ 中所有的 M，都有 $I\geqslant M$。于是验证了 I，就可以对具体模型做出推断。

2.5 其他标准方法

• μ-calculus。命题 μ-calculus 是强有力的语言，运用最小和最大不动点算子表达转换系统属性。可以运用 OBDDs 表达 μ-calculus 公式，并且能把(例如)CTL 转换为 μ-calculus。

• 运用自动机进行模型检验。有限状态自动机 $\mathcal{A}=(\Sigma,Q,\Delta,Q^0,F)$ 是一个五元组，其中 Σ 是有限字母表，Q 是有限状态集合，$\Delta\subseteq Q\times\Sigma\times Q$ 是状态转换关系。例如 $\Delta(s,q,s')$ 表

示位于状态 s，接受字母 q，转换到状态 s'；Q^0 是初始状态，F 是终止状态。设 v 是一个 word(即 string)，即 Σ^* 的一个序列(sequence)。令 $\mathscr{A}$ 的语言 $L(\mathscr{A})\subseteq\Sigma^*$，它由所有的词所组成。一个 Büchi 自动机是特殊的自动机，它接受的语句是无限语句，即语句来自 Σ^ω，上标 ω 表示无限。一个 Kripke 结构直接相应于一个 ω-正则(regular)Büchi 自动机 $\mathscr{A}$，而系统要满足的指派也能用自动机给出，设为 S，令 $L(\mathscr{A})$ 和 $L(S)$ 分别表示两个自动机能够接受的语言。$L(\mathscr{A})\subseteq L(S)$，或 $L(\mathscr{A})\cap\overline{L(S)}=\emptyset$ 表示系统的行为符合指派。

• 和定理证明相结合的模型检验。我们把它结合到综合方法一节里讨论。

3 抽象解释方法

抽象解释首先由 Cousot 和 Cousot 作为程序设计和程序静态(static 即 compile-time)分析的统一框架于 1977 年提出，现在已经成为描述具体系统近似语义的一个基本的方法理论，应用于许多计算机科学领域。抽象解释简单但是严格的定义以及它能够在不同的抽象层次上指定系统的行为，使它成为研究近似语义的适合工具。抽象简化在于利用有限转换系统去近似无限或者大型有限转换系统，以致于原先存在的适用于有限转换系统的算法能够在抽象系统上运用，这种 semi-verification 思想首先由 Clarke 等人在 1992 年引进。

在模型检验中，抽象早就是解决状态空间爆炸的一种重要方法，其目标是构造足够小的抽象模型，使之能进行有效的分析和验证。现在基于抽象解释理论的模型检验和抽象简化思想的发展，更能有效地解决状态空间爆炸的问题。

3.1 根据存在(或经验)的抽象

文献[6]系统地描述了一个称为根据存在的(existential)抽象框架，虽然它不同于基于抽象解释的抽象框架，但是它比标准抽象(即基于数据结构抽象和变量影响锥等方法)更具一般性，也和基于抽象解释方法一样具有更大功能，所以我们把它也归并到抽象解释这一节里。

直观地说，根据存在(或经验)的抽象是将一个 Kripke 结构 $M=(S,I,R,L)$ 的状态划分为群集(clusters)，然后将这些群集处理为新的抽象状态。形式上，用满射 $h:S\to\hat{S}$ 表达抽象函数，这里 $\hat{S}$ 是抽象状态集合。h 在具体状态空间 S 产生一个等价关系 $\equiv_h$，即如果 $d,e\in S$，则 $d\equiv_h e$ iff $h(d)=h(e)$。抽象能够运用满射 h 或等价关系 $\equiv_h$ 来表达，因此今后这两种表达可以互用。由 h 产生抽象 Kripke 结构 $\hat{M}=(\hat{S},\hat{I},\hat{R},\hat{L})$，为了强调 h，有时记为 $\hat{M}_h=(\hat{S}_h,\hat{I}_h,\hat{R}_h,\hat{L}_h)$。$\hat{M}$ 定义如下：

(1) $\hat{I}(\hat{d})$ iff $\exists d(h(d)=\hat{d}\wedge I(d))$。

(2) $\hat{R}(\hat{d}_1,\hat{d}_2)$ iff $\exists d_1\exists d_2(h(d_1)=\hat{d}_1\wedge h(d_2)=\hat{d}_2\wedge R(d_1,d_2))$。

(3) $\hat{L}(\hat{d})=\bigcup_{h(d)=\hat{d}}L(d)$。

抽象框架里最重要的概念是适合性概念：关于指定属性 φ 称抽象函数 h 是适合的，如果对所有 φ 里的原子命题 f 以及所有在 S 域中的状态，下面的条件成立：

若 $d\equiv_h e$，则 $d\vdash f\Leftrightarrow e\vdash f$。

更一般地，若对公式集合 F 里每一个公式 φ，h 是适合的，则称 h 对 F 是适合的。

我们有下面重要定理：

定理 1[6] 设 h 对 ACTL* 里的公式 φ 是适合的，若 $\hat{M}_h\vdash\varphi$，则 $M\vdash\varphi$。

文献[6]系统地描述了构造抽象转换 $\hat{R}_h$ 的近似方法(近似转换记为 $\tilde{R}$)，特别是过早近似方法、变量群集方法。它们的严格描述见文献[6]。由此得到近似 Kripke 结构 $\tilde{M}=(\hat{S}_h,\tilde{I},\tilde{R},\hat{L}_h)$，其中 $\tilde{I}\supseteq\hat{I}_h$，$\tilde{R}\supseteq\hat{R}_h$，$\tilde{M}$ 减少了 $\hat{M}_h$ 的复杂性，且有 $\hat{M}_h\leqslant\tilde{M}$ 模拟关系。

文献[6]运用 BDDS 直接实行他们的算法，除了应用一些重要启发法，例如两阶段改良算法、近似转换和利用变量依赖图进行抽象以外，他们的算法主要步骤是：首先构造初始抽象模型，继而反覆运用(如果需要的话)反例引导精化抽象模型的各种算法，最后得到抽象模型。该模型具有重要功能，其总结见以下定理。

定理 2[6] 给定一个模型 M 和一个 ACTL* 指派公式 φ，它的反例或者是一个路径反例或者是一个循环反例。我们(即文献[6])提出的算法最终找到模型 $\tilde{M}$，使得

$\tilde{M}\vdash\varphi\Leftrightarrow M\vdash\varphi$

一般来说，如果抽象模型框架“保留”了转换关系，习惯上有时称它为标准抽象模型，现在根据存在的抽象模型框架“保留”了 Kripke 结构，所以它是标准抽象模型。根据存在的抽象模型框架与基于解释的抽象模型框架存在一定的关系。这一点在后面 3.4 节关于 CEGAR 和 EGAS 的方法里加以介绍。

3.2 抽象解释理论框架

1. 闭包操作和 Galois 连接(GC)(Closure operators and Galois Connections)

在抽象解释理论中，抽象域可通过闭包操作或 Galois 连接从具体域来获取，两者是等价的。

具体域 C 和抽象域 A 通过 Golois 连接(GC) (α,C,A,γ) 联系起来，其中 $\alpha:C\to A$ 是抽象映射(或函数)，而 $\gamma:A\to C$ 是具体化映射。一般来说，设具体域 C 和抽象域 A 为完备格(当然也可以放松条件为 cpo)，即 $C=\langle C,\leqslant,\vee,\wedge,T,\perp\rangle$ 和 $A=\langle A,\leq,\vee^\#,\wedge^\#,T^\#,\perp^\#\rangle$，而 α 和 γ 满足：$\forall c\in C,\forall a\in A:\alpha(c)\leq a\Leftrightarrow c\leqslant\gamma(a)$。在 GC 中，如果 α 是满射，或者 γ 是一一映射，则称 (α,C,A,γ) 是 Galois 入射(GI)。具体域 C 上的所有 GI $\mathscr{L}_C$ 对于精确性是拟序的。$\mathcal{G}_1=(\alpha_1,C,A_1,\gamma_1)\sqsubseteq\mathcal{G}_2=(\alpha_2,C,A_2,\gamma_2)$，当且仅当 $\gamma_1\circ\alpha_1\sqsubseteq\gamma_2\circ\alpha_2$，这时称 A_1 比 A_2 更精确，或者 A_2 比 A_1 更抽象。当 $\mathcal{G}_1\sqsubseteq\mathcal{G}_2$ 且 $\mathcal{G}_1\sqsubseteq\mathcal{G}_2$，则称 $\mathcal{G}_1$ 与 $\mathcal{G}_2$ 等价。

在具体域 C 上通过向上闭包操作获得的抽象域简称为 uco 或闭包。ρ 是一个闭包，它可以从两个等价角度定义。(1) ρ 是 C 上单调、幂等和扩展的操作；(2) ρ 也可以用通过它在 C 上的不动点集合来唯一确定，并且与它的像集一致，即 $\rho(C)\triangleq\{x\in C|\rho(x)=x\}$。今后我们用 C 上的函数或用 C 上的子集合来表示一个闭包。令 $\langle uco(C),\sqsubseteq\rangle$ 表示 C 上的所有 uco 的偏序集，因为我们规定 C 是完备格，所以可以推出 $\langle uco(C),\sqsubseteq,\sqcup,\sqcap,\lambda x.T,\lambda x.x\rangle$ 也是完备格，它表示 C 的所有可能抽象域的完备格。如果对 $\rho,\eta\in uco(C)$，当且仅当 $\eta(C)\subseteq\rho$

(C)时，$\rho \sqsubseteq \eta$，即 ρ 比 η 精确(或者说 η 比 ρ 抽象)。

闭包和 GI 的等价性。自文献[7]起人们就知道抽象域能够用 GI 或具体域上的闭包等价表示。因此这两个结构是相互"可逆"的。特别是已知一个 $GI(\alpha, C, A, \gamma)$，与它相联在 C 上的闭包 $\gamma \circ \alpha$ 可以看作 A 在 C 上的逻辑表示，所以当对抽象域的属性的推理独立于抽象域对象的表示时，运用闭包操作方法特别方便。因此今后认为 $uco(C)$ 与 C 上抽象解释形成的格 $\mathscr{L}_C$ 等同[8]。

2. 近似性和完备性

近似性和完备性是抽象解释里最重要的两个概念[8]。

设 $f: C^n \xrightarrow{m} D (n \geqslant 1)$ 是具体语义操作定义在具体域 C 和 D 上。又设抽象解释运用 $GIs(\alpha_{C,A}, C, A, \gamma_{A,C})$ 和 $(\alpha_{D,B}, D, B, \gamma_{B,D})$ 指定抽象域 A 和 B，且运用 $f^{\#}: A_n \xrightarrow{m} B$，指定相应的抽象语义操作。如果 $\alpha_{D,B} \circ f \sqsubseteq f^{\#} \circ \langle \alpha_{C,A}, \cdots, \alpha_{C,A} \rangle$，则 $f^{\#}$ 是 f 的正确近似(以后简称近似)。令 $f^{A,B} \triangleq \alpha_{D,B} \circ f \circ \langle \gamma_{A,C}, \cdots, \gamma_{A,C} \rangle: A^n \xrightarrow{m} B$。$f^{A,B}$ 称为 f 的最正确近似，显然若 $f^{\#}$ 是 f 的近似，则关系 $f^{A,B} \sqsubseteq f^{\#}$ 成立，即按函数逐点序，$f^{A,B}$ 是最准确的近似。同样设 $f: C \xrightarrow{m} C$ 和 $f^{\#}: A \xrightarrow{m} A$，如果 $\alpha_{C,A}(lfp(f)) \leqslant_A lfp(f^{\#})$，则称 $f^{\#}$ 是 f 的不动点近似。这时对于 f 在 A 上最正确近似 $f^{A,A}$ 也有：$lfp(f^{A,A}) \leqslant_A lfp(f^{\#})$。由近似性，显然只讨论单调函数，即"$\xrightarrow{m}$"表示单调性。

沿用上面的符号。若 $\alpha_{D,B} \circ f = f^{\#} \circ \langle \alpha_{C,A}, \cdots, \alpha_{C,A} \rangle$，则称 $f^{\#}$ 关于 f 完备。这时，可证 $f^{A,B}$ 关于 f 也是完备的，而且 $f^{A,B}$ 与 $f^{\#}$ 一致。同样，若 $f: C \xrightarrow{m} C$ 和 $f^{\#}: A \xrightarrow{m} A$，若 $\alpha_{C,A}(lfp(f)) = lfp(f^{\#})$，则称 $f^{\#}$ 关于 f 不动点完备，并且可证 $f^{A,A}$ 也是不动点完备。上述概念不难推广到关于函数族的完备性和不动点的完备性上。

完备性和不动点完备性都是抽象域的性质。因此今后称一个抽象域是完备的或是不动点完备的就是指与之联系的最正确近似相应的完备属性。

根据许多作者的研究(例如文献[7] Theorems 7.10.4)得出完备性蕴涵不动点完备性，但反之未必成立。

3. 抽象解释完备化

文献[8]对抽象解释的完备性作了系统的研究。主要工作包括研究完备性(包括不动点完备性)构造性特征。对于完备性，引入相对完备核和相对完备壳等重要概念，直观地说，当我们讨论函数族 F 在 $\langle \rho, \eta \rangle \in uco(C) \times uco(D)$ (C, D 是两个具体域)的完备性时，相对完备核指的是对 η"从抽象度的提升"，相对完备壳指的是对 ρ"从精确度的改进"。下面绝对完备核和绝对完备壳与此类似，不在赘述。在此基础上，得到几个重要定理和事实，例如，当 F 是连续函数族时，对于 $\langle \rho, \eta \rangle \in uco(C) \times uco(D)$，我们可以改良 ρ 或者 η，得到关于 F 的完备域。但是当 F 是单调函数族时，则相对完备壳未必存在，不过相对完备核总是存在的。另外，当 $F \subseteq C^n \xrightarrow{m} C$，文献[8]引入了有关的绝对完备核和绝对完备壳等概念，简称完备核和完备壳，并用它讨论了不动点，得到了一些重要事实，例如设 $G \subseteq C \xrightarrow{m} C$，$\rho \in uco(C)$，则 ρ 关于 G 不动点完备核存在，是 $uco(C) \to uco(C)$ 上算子 $\mathscr{T}_G$ 的最小不动点。同样，不动点完备壳未必存在，详情参见文献[8]。值得注意的是文献[8]还讨论了并非一切完备化改良在模型检验里都是最好的举措。给定具体解释 C 及其(不动点)完备抽象解释 I，如果 J 是 I 的最进一步抽象，则 J 是 C 的(不动点)完备当且仅当 J 是 I 的(不动点)完备。这个性质使得前面叙述的完备核和完备壳的计算更容易，因为抽象域较具体域简单，所以计算 J 关于 I 完备比计算 J 关于 C 完备更容易些。特别地上述性质可以引入"智能"改良技术，详情参见文献[8]。

4. 加速收敛算子[9,10]

通常语义函数牵涉到迭代计算。为使迭代计算最终收敛或者加速收敛，抽象解释理论中往往采用两个算子。

· 扩展算子 ∇。$\nabla \in \wp(\mathscr{D}^{\#}) \to \mathscr{D}^{\#}$，它满足性质

$\forall x, y \in \mathscr{D}^{\#}, x \sqsubseteq_x x \nabla y, y \sqsubseteq_x x \nabla y$。

并且对所有上升链 $x^0 \sqsubseteq x^1 \sqsubseteq \cdots \sqsubseteq x^n \sqsubseteq \cdots$，运用算子 ∇ 得到的上升链 $y^0 = x^0, \cdots, y^{i+1} = y^i \nabla x^{i+1}, \cdots$，不是严格上升的。

· 收缩算子 Δ。$\Delta \in \wp(\mathscr{D}^{\#}) \to \mathscr{D}^{\#}$。它满足性质

$\forall x, y \in \mathscr{D}^{\#}, x \Delta y \sqsubseteq_x x, x \Delta y \sqsubseteq_y$。

并且对所有下降链 $x^0 \sqsupseteq x^1 \sqsupseteq \cdots$，运用 Δ 算子得到的 $y^0 = x^0, \cdots, y^{i+1} = y^i \Delta x^{i+1}, \cdots$，的序列不是严格下降的。

当然设计 ∇、Δ 算子，还可以作别的要求，例如，文献[10]还把 ∇ 和 Δ 作为合理性选择函数使用。

5. 抽象解释的一般化理论

文献[10]提出的抽象解释框架认为"从理论角度看，可以在不同的水平层次上理解程序在运行时的行为，它们形成语义相互联系的体系。我们称描述程序在具体执行时的可能行为的语义为标准语义，于是相应于抽象'聚焦点'的不同可以得出不同层次的抽象语义。例如忽略程序运行时不相关的细节，聚集于程序运行时一类"本质"属性，这样得到的抽象解释可以是 collecting 语义。更进一步，如果只考虑能够有效计算的程序属性，那么这样得到的语义便是人们通常称之的抽象语义。显然，抽象是相对的，例如集合语义相对于标准语义是抽象语义，而相对于抽象语义，它却是一个具体语义"。文献[10]为建立抽象解释提出必须要考虑的 5 个基本选择。具体描述参见文献[10]。

实际上，文献[10]提出的框架是比较一般的形式，许多抽象解释框架都可以统一到文献[10]的框架下，例如它并不要求像标准抽象解释里的许多条件：域的格性或 CPO 性，甚至也不必需要 GC 或 GI 存在。然而利用这种框架却可以讨论其它框架，例如文献[10]运用上述基本选择建立 Galois 联结以及运用 widenings 和 narrowings 等算子提出实际中可以应用的迭代算法。文献[10]认为在实际中通常 GC 中的抽象关系(或函数)α 与具体化关系(或函数)γ 并不是"同等地"一样可以容易获得的，因此例如文献[10]分别讨论了运用抽象函数引导的抽象解释算法的收敛性和运用具体化函数引导的抽象解释算法的收敛性。

3.3 基本抽象解释理论的模型检验

首先概述一下抽象模型检验。

抽象模型检验的主要思想是通过抽象状态近似具体状态的某些性质并在抽象状态之间定义抽象转换关系，来构造出抽象模型。然后在抽象模型中检验时序性质，通过时序性质在抽象模型的结果，推导出其在具体模型中是否满足。抽象模型检验已经在硬件验证领域取得巨大成功，特定情况下能验证有 10^{1300} 可达状态的 ALD 电路。通常抽象主要有两种方式[11]：弱保留抽象和强保留抽象。对于性质弱保留的抽象，

可分为两类：向下近似抽象和向上近似抽象。给定具体模型和抽象模型以及性质 ϕ，如果是按照向下近似抽象方式获得的抽象模型，则 $\models\phi\Rightarrow\models\phi$，即性质在抽象模型中不满足，由此推导出其在具体模型中也不满足；如果是按照向上近似抽象方式获得的抽象模型，则 $\vdash\phi\Rightarrow\vdash\phi$，即时序性质在抽象模型中满足，由此推导出其在具体模型中也满足。Clarke 采用向上近似抽象方式，基于反例精化的思想进行抽象模型检验，即通过模型检验自动验证抽象模型是否满足所期望的性质，如果满足，则报告正确；否则给出抽象反例，检查抽象反例是否合理，若合理，则生成具体反例；否则依据不合理反例精化抽象模型；重复迭代精化直到给出满足性质的肯定回答或给出不满足性质的具体反例。对于性质强保留的抽象，时序性质在抽象模型中满足当且仅当其在具体模型中满足，即性质在抽象模型中满足（或不满足），则在具体模型中也满足（或不满足）。性质强保留是高期望的，且要获得最优的强保留抽象模型是困难的[11]。一般来说，根据抽象解释理论，完备抽象解释和性质强保留之间存在密切关系，Cousot 和 Giacobazzi 的研究表明抽象解释的完备性仅仅依赖于抽象域，是抽象域的性质。例如抽象域对 CTL 标准算子（语义函数）是完备的，那么相对应的抽象划分强保留 CTL 性质。因此抽象解释理论包括了抽象域完备性研究，讨论构造完备抽象解释的方法，即通过抽象域的完备化，构造一个抽象状态划分且强保留时序性质的抽象模型[11]。以上内容是基本抽象解释模型检验的一些典型思想。

下面用例题来阐述典型的抽象模型检验方法。

例 1　我们采用文献[9]介绍的运用抽象解释改良模型检验的基本做法。它显示利用抽象简化在于通过有限转换系统或加速收敛来近似无限或者大型转换系统，使已经存在的为有限系统设计的算法能够实际应用。

1. 假设（实时）并发系统已经由转换系统 $\langle S,t,I,F\rangle$ 模型表达，其中 S 是状态空间，$t\subseteq S\times S$ 是转换关系，$I\subseteq S$ 是初始状态集合，$F\subseteq S$ 是终止状态集合。

现在我们讨论系统的不变式（或者是安全性）属性。于是由转换关系 t 可以确定 $\wp(S)$ 上的前像算子 pre-image $pre[t]$ 及其对偶 $\widetilde{pre}[t]$ 和系统后像算子 $post[t]$ 及其对偶 $\widetilde{post}[t]$。于是不变式安全性性质 P 可以用下面关系表达：

$$post[t^*]I\subseteq P,\text{或者 } pre[t^*]\neg p\subseteq\neg I \qquad (*)$$

其中 $t^*\triangleq\bigcup_{n\geqslant 0}t^n$。众所周知，前像和后像算子的计算可以通过不动点表达，因此关于不变式属性能够运用不动点计算程序去检查，例如检查 $post[t^*]I\subseteq P$，我们有算法（Cousot，P and Cousot，R）。具体算法参见文献[9]。

2. 为了从上面近似前向集合语义 $post[t^*]I$，运用 Galois 连接

$$\langle\wp(S),\subseteq\rangle\underset{\gamma}{\overset{\alpha}{\rightleftarrows}}\langle L,\sqsubseteq\rangle$$

$$\forall p\in\wp(S):\forall Q\in L:\alpha(p)\sqsubseteq Q\Leftrightarrow p\subseteq\gamma(Q)$$

获得抽象领域 $\langle L,\sqsubseteq\rangle$，其中包含抽象初始状态 $I^\#$ 和抽象函数 $F^\#\in L\xrightarrow{m}L$，满足：$\alpha(I)\sqsubseteq I^\#$，即 $I\subseteq\gamma(I^\#)$ 和

$$\alpha\circ(\lambda X\cdot X\cup post[t]X)\circ\gamma\sqsubseteq F^\#$$

在抽象领域 $\langle L,\sqsubseteq\rangle$ 上，我们获得上述 Cousot，P and Cousot，R 算法的类似算法（详情见文献[9]），由它去检查有关不动点的属性。直观上，它是抽象模型检验基于抽象解释的一般化。

3. 更典型的是，如果从抽象函数 α 导入抽象转换系统，例如 α 由（满射）$h\in s\to s^\#$ 产生，即 $\alpha[h](X)\triangleq\{h(x)\mid x\in X\}$，$\gamma[h](Y)=\{x\mid h(x)\in Y\}$，那么，我们得到抽象转换系统 $\langle S^\#,t^\#,I^\#,F^\#\rangle$，$\alpha(I)\subseteq I^\#$，$\forall s,s'\in S:\langle s,s'\rangle\in t\Rightarrow\langle h(s),h(s')\rangle\in t^\#$，由此可以得到 Cleaveland 等算法（详情见文献[9]），它也类似于 Cousot，P and Cousot R 算法，可以看作是（前面说的）标准抽象模型检验，因为它“保留”了转换系统。

4. 如果抽象域不满足 ACC 条件（即无限上升链必有限终止）或者为了尽可能地加速收敛并且尽可能获得精确结果，可以引入算子 ∇ 和 Δ。由这些算子对上述算法，例如 Cleaveland 等算法进行改进，可以保证序列最终稳定，并且是不动点的合理（从上方）近似，甚至是较优的精确近似。具体的改进算法参见文献[9]。

5. 文献[9]还考虑基于后向抽象解释的模型检验以及前向和后向抽象解释相结合的模型检验算法。关于后者还可以得到更精确的结果。特别是当抽象解释模型检验也不能胜任时（正如在讨论最小/最大路径长度问题可能发生的情况），文献[9]提出的算法基于经典的前向和后向抽象解释相结合的部分（partial results）结果去减少（甚至是在线运行中）对具体状态空间的搜索规模。例如利用抽象解释分别改进传统的最小延迟（Halbwachs）和最大延迟（Campos et al.）算法，甚至文献[9]提出的一些算法并行（parallel）运行抽象解释和模型检验来提高效率。

总之，标准符号模型检验和基于抽象解释的模型检验的区别只在于后者可能丢失具体模型的一些信息，以致于有时不能得到确定的结果。无论是基于抽象解释的标准抽象模型检验（例如抽象模型和具体系统都是转换系统）或是基于抽象解释一般化后的抽象模型检验，文献[9]指出在其上也可以运用已有的具体模型检验算法或者对它们进行改进使之更有效。即使是用抽象解释模型检验也不能解决问题时，我们也可以将基于抽象解释的模型检验方法或者对系统的抽象解释分析与模型检验方法结合起来，以便能运用抽象解释自动推出的关于系统的属性（即抽象解释模型检验的部分结果）去提高符号模型检验算法的效率。

例 2　基于抽象解释完备性的强保留。以下叙述主要根据文献[11,12]。

本例显示怎样利用抽象解释完备性的方法得到强保留抽象模型。

1. 设 Q 是任意（可能无限）的状态集合。令 $\mu\in uco(\wp(Q)_\subseteq)$ 是任意一个闭包（具体域 $\wp(Q)$ 上的抽象域）。在 Q 中引入状态等价关系 $\equiv_\mu$：

$s\equiv_\mu s'\Leftrightarrow\forall S\in\mu.(s\in S\Leftrightarrow s'\in S)$，即 $s\equiv_\mu s'$，iff $\mu(\{s\})=(\mu\{s'\})$

由 $\equiv_\mu$ 可以得到 Q 的一个分割。

定义 1　If $\mu=\mathcal{R}(\mu)\triangleq\sqcap\{\eta\in uco(\wp(Q))\mid\equiv_\eta=\equiv_\mu\}$，则称 μ 是可分的闭包。并用 $uco^P(\wp(Q))$ 记所有可分闭包的集合。

分割也是抽象。用 $Part(Q)$ 表示 Q 上所有分割集合。并在其上引入序 $\leqslant$，$\forall P,P'\in par(Q)$，$P\leqslant P'$ 表示 $\forall B\in P$，存在 $B'\in P'$，使得 $B\subseteq B'$。现在我们在具体域 $\wp(Q)$ 上得到两个抽象层次。定义两个映射 par 和 pcl 如下：

$par: uco^P(\wp(Q)) \to Part(Q)$

$par(\mu) = \{\mu(\{q\}) \mid q \in Q\}$

$pcl: par(Q) \to uco^P(\wp(Q))$

$pcl(P) \triangleq \mathbb{P}(M(P)) = \lambda X \in \wp(Q) \cup \{B \in P \mid X \cap B \neq \Phi\}$

于是$\langle uco^P(\wp(Q)), \sqsubseteq \rangle$和$\langle par(Q), \leqslant \rangle$通过 par 和 pcl 两个映射同构。

2. 现在假设模型检验 CTL 公式。CTL 公式是由原子命题，布尔运算符、时态算子及路径量词构成。即它的(状态)公式 φ 可以归纳定义，其语法为：

$\varphi ::= P \mid f(\varphi_1, \cdots, \varphi_i)$

其中，$P \in AP$，AP 是有限原子命题集合；$f \in OP$，OP 是有限操作算子集合。$OP = \{\neg, \vee, EX, EU, EG\}$。因为任何 CTL 公式都可以用 OP 中操作算子表达。假设具体语义结构是 Kripke 结构 $\mathscr{K} = (Q, R, AP, L)$，$I$ 是 CTL 公式在具体语义域 $\wp(Q)$ 上的解释函数，于是 $\forall P \in AP, I(P) \in \wp(Q)$，而 OP 中算子便是定义在 $\mathscr{K}$ 上路径和标准算子的逻辑操作，例如 $I(\vee) = \cup$，$I(\neg) = C$(求余运算)，$I(EX) = pre_R$，$I(EU) =$ "Existential until"，$EU(\varphi, \psi)$ 能通过最小动点计算达到，$I(EG) =$ "Existential Always"，$EG\varphi$ 能通过最大不动点计算达到。也就是说，我们定义了语义函数$[\![\cdot]\!]_I$: CTL$\to \wp(Q)$，即表示语义结构中 $\varphi \in CTL$ 为真的状态集。

由具体语义域引入抽象域 $\mu \in uco(\wp(Q))$，由此可导出抽象语义函数$[\![\cdot]\!]_\mu$: CTL$\to \mu$。对于任意公式 $\varphi \in$ CTL，抽象值$[\![\cdot]\!]_\mu$ 属于 μ，它是通过最正确近似来定义的。例如$[\![P]\!]_\mu = \mu(I(P))$，$P \in AP$。同样对于 $\forall P \in par(Q)$，可以定义抽象域 P 上的语义函数$[\![\cdot]\!]_I^P = [\![\cdot]\!]_I^{pcl(P)}$。

3. 闭包完备性与强保留

我们有以下几个重要结论：

(1)$\mu \in uco(\wp(Q))$，μ 对 CTL 是完备的$\Leftrightarrow \mu$ 对 AP 和 OP 是完备的。

(2)假定算子集 $OP = \{\neg, \vee, EX, EU, EG\}$，$\mu \in uco(\wp(Q))$，$\mu$ 是 OP 完备的当且仅当 μ 是$\{C, pre_R\}$是完备的。

(3)假定 $\mu \in uco(\wp(Q))$，$[\![\cdot]\!]_\mu$: CTL$\to \mu$ 是对应的抽象语义函数，则抽象语义结构对 CTL 性质强保留当且仅当$(\mu, [\![\cdot]\!]_\mu)$是完备的。

如果 μ 不完备，且想得到高期望的结果，就必须改良 μ，使其完备化。因为有如下重要事实：$F \subseteq Fun(\wp(Q))$，如果 $\mu \in uco(\wp(Q))$，μ 的最小 F-完备精化 $\varepsilon_F(\mu) \triangleq \sqcup\{\rho \in uco(\wp(Q)) \mid \rho \sqsubseteq \mu, \rho$ 关于 F 是完备的$\}$总是存在。再根据上述定理，只要精化 μ 使其对$\{C, pre_R\}$-完备，就可以得到对 CTL 完备的闭包，这样得到的 $\varepsilon_{\{C, pre_R\}}(\mu)$便对 CTL 性质强保留。又由于 $\mu \in uco^P(\wp(Q))$，当且仅当 μ 是 C-完备的，因此精化的闭包 $\varepsilon_{\{C, pre_R\}}(\mu)$还是一个最优可分割闭包，能够证明它是一个算子最大不动点且能通过 Kleene 迭代序列计算。最后运用算子 par 于 $\varepsilon_{\{C, pre_R\}}(\mu)$便得到最优抽象划分。

4. 具体算法：

(1)初始闭包 $\mu_{AP} = M(\{I(P) \mid P \in AP\})$，其中 M 是 Moore 封闭算子。

(2)通过 Kleene 不动点计算 $\varepsilon_{\{C, pre_R\}}(\mu_{AP})$得到最优可划分闭包 μ_{CTL}。

(3)通过 $par(\mu_{CTL})$便得到最优抽象划分 P_{CTL}。

(4)令 $R^\# = R^{\exists\exists}$，$L^\#(B) = \bigcup_{s \in B} L(s) = L^\exists(B)$。于是便得到抽象 Kriple 结构。$\mathscr{A} = (P_{CTL}, R^\#, AP, L^\#)$。它是一个对 CTL 性质强保留的 Kripke 结构。

3.4 运用抽象解释理论对(抽象模型)标准方法的改良

在文献[13]里 Ranzato 和 Tapparo 叙述了用抽象解释理论一般化标准抽象模型(例如抽象 Kripke 结构)关于指定语言的强保留问题并把它化归为一般抽象领域(可以作为抽象模型)前向完备性的属性的研究。现在介绍几个方法，以显示抽象解释一般化及对标准方法的改进。

1. CEGAR 和 EGAS 方法

运用伪反例(spurious counterexample)引导精化抽象，有如文献[6]里所描述的 Counterexample-Guided Abstraction Refinement(简称为 CEGAR)方法和运用例子引导简化抽象，如文献[14]里所描述的 Example-Guided Abstraction Simplification(简称为 EGAS)方法，它们是相互对偶方法，前者精化抽象以致于消除假的反例，后者的目标是简化抽象模型 A 得到抽象模型 A_S，它简化后和 A 一样具有相同的近似行为。换言之，如果 π_{A_S} 是简化后抽象模型 A_S 的一条伪例路径，则在抽象模型 A 里也存在一条伪例路径 π_A，使得 π_{A_S} 是 π_A 的抽象。

· CEGAR 方法

给定程序 P 和一个 ACTL* 公式 φ，目标是检查与 P 对应的 Kripke 结构 M 是否满足 φ。CEGAR 方法由下述步骤组成。

(1)产生初始抽象。例如利用(满足一定条件的)抽象函数 h 从 M 得到抽象 Kripke 结构$\hat{M}$，$\hat{M}$是 M 的一种划分。

(2)模型检查抽象结构。检查是否 $\hat{M} \models \varphi$。如果肯定，则我们给出结论 $M \models \varphi$。如果检查揭示出反例$\hat{T}$，则要检查$\hat{T}$是否合理，若合理则生成具体反例，并返回给用户。若$\hat{T}$是伪反例，则进入第(3)步。

(3)改良抽象模型。为了简单起见，不讨论循环结构反例，设$\hat{T} = \langle \hat{s}_1, \cdots, \hat{s}_n \rangle$，计算 $h^{-1}(\hat{T})$如下：

$$\{\langle s_1, \cdots, s_n \rangle \mid \bigwedge_{i=1}^{n} h(s_i) = \hat{s}_i \wedge I(s_1) \bigwedge_{i=1}^{n-1} R(s_i, s_{i+1})\}$$

分析上述集合，找到失败抽象状态$\hat{s}_i$，以及与它相应的具体状态集即$\{s \mid h(s) = \hat{s}_i\}$，并把这个具体状态集的状态分离为 3 个不同的类型，然后基于这个划分，改良抽象函数 h，使得新的抽象模型里不再包含伪反例$\hat{T}$。但是最优即最抽象精化是 NP-困难问题。所以文献[6]采用启发式策略，得到新的抽象函数和新的划分抽象结构。然后返回第 2 步。

EGAS 方法：先给出一个定义，设 C 是具体域。用记号 $Abs(C)$表示 C 上抽象域构成的格$\langle Abs(C), \sqsubseteq \rangle$，它与 C 上的格闭包$\langle uco(C), \sqsubseteq \rangle$同构。给定具体语义函数 $f: C \to C$，对任意 $A, B \in Abs(C)$，用 f^A 和 f^B 分别表示 f 在 A 上和 B 上最正确近似(简记为 $b.c.a$)。当 f^A 和 f^B 对于它们的抽象值相同到它们的同构表示，即若用 μ_A 和 μ_B 分别表示 A 和 B 相应的闭包，则 $\mu_A \circ f \circ \mu_A = \mu_B \circ f \circ \mu_B$，这时称 f^A 和 f^B 是 f 的相同的最正确近似，并记之为 $f^A = f^B$。对于具体语义函数集合 $F \subseteq C \to C$，则 $F^A = F^B$ 表示 $\forall f \in F$，有 $f^A = f^B$。因此，对于给定的 $A \in Abs(C)$，定义

$A_S \triangleq \{B \in Abs(C) \mid F^B = F^A\}$

于是,我们问:$F^{A_S} = F^A$ 是否成立?如果是,则称 A_S 是 A 关于 F 的最优简化。

·为了讨论上面的问题,文献[14]引入 A 关于 F 的正确核概念:令 $\mathscr{K}_F(A) \triangleq \sqcup\{B \in Abs(C) \mid F^B = F^A\}$。若有 $F^{\mathscr{K}_F(A)} = F^A$,则称 $\mathscr{K}_F(A)$ 是 A 关于 F 的正确核。

·如果 $\forall f \in F, f \circ \mu_A$ 连续,则 A 关于 F 的正确核存在,并且文献[14]给出它的计算公式(参见文献[14]),显然它就是最优简化 A_S。

·在模型检验中运用正确核的主要精神是:例如正确核对状态空间 Σ 的一个抽象分割 P 产生的抽象域 $\wp(P)$ 提供一个简化,使得它保持关于前向(pre)和后向(post)两个语义操作的最正确近似,从而在最简化的模型上讨论检验问题。由正确核引入的抽象简化并不增加伪反例路径。

因此在 CEGAR 方法里可以结合 EGAS 方法。这样可以利用 EGAS 方法改良 CEGAR 方法。例如在抽象域 A,利用 CEGAR 方法发现伪反例路径 π,如果找到失败抽象板块 B_K,就把它分解为相应的 3 个子板块。文献[14]提供了新的启发式,通常它比文献[6]提供的启发式更有效,即能找到更有效的改良方法来精化抽象 $\mathscr{A}$ 以便消除伪反例 π。

·在基于谓词抽象的模型检验里,上述正确核也能得到应用。在第 4 节里,我们将介绍另一个特殊谓词抽象模型。文献[15]利用谓词 $P_1, P_2, \cdots, P_n$ 得到布尔抽象域 $B \triangleq \langle \wp(\{0,1\}^n), \subseteq \rangle$,为了检验可达到属性,文献[15]通过 Cartesian abstraction 以及对它的改良来进行模型检验,这是由于 B 过于昂贵。文献[14]通过一个简单实例,表明 $\mathscr{K}_F(B)$ 比 Cartesian abstraction 要好,其中 F 是具体域上关于变量的赋值操作。总之,类似上述 EGAS 方法在一般文献中还是少见的,关于它的进一步研究还是很有意义的。

最后值得一提的是,文献[6]提出的 CEGAR 方法是基于存在抽象框架的,而文献[14]提出的 EGAR 方法是基于抽象解释框架的,因此可以看出这两个抽象框架(例如根据文献[10]的观点)在某种意义上是相通的。

另外,还有许多文献是有关抽象改良技术的(包括不运用反例改良技术),在运用反例改良抽象模型方法里,Govindaraju 和 Dill(在 1998 和 2000 年)分别将随机和重叠投影技术运用到基本近似体系里(参见文献[6])。

2. 基于抽象解释的关于一般强保留模型检验框架

基于抽象解释可以设计比抽象 Kripke 结构更为一般的抽象模型。文献[16]叙述怎样利用基于抽象解释模型来指定一般强保留抽象模型检验框架。特别对包含标准时态算子在内的规范语言,叙述了不动点强保留特征。

这里首先介绍几个概念

·(C, α, γ, A) 是一个 GI,关于一般语言 $\mathscr{L}$,设 $[\![\cdot]\!]: \mathscr{L} \to C$ 和 $[\![\cdot]\!]^\# : \mathscr{L} \to A$ 分别是具体和抽象语义,如果对任意 $\varphi \in \mathscr{L}, \alpha([\![\varphi]\!])_A \leqslant [\![\varphi]\!]^\#$,则称抽象语义 $[\![\cdot]\!]^\#$ 是合理的,如果 $\forall \varphi \in \mathscr{L}, \alpha([\![\varphi]\!]) = [\![\varphi]\!]^\#$,则称抽象语义 $[\![\cdot]\!]^\#$ 是后向完备的。如果 $\forall \varphi \in \mathscr{L}, [\![\varphi]\!] = \gamma([\![\varphi]\!]^\#)$,则称抽象语义 $\langle A, [\![\cdot]\!]^\# \rangle$ 是前向完备的。

·仍然沿用上段符号,如果对 $\forall S \subset C, \forall \varphi \in \mathscr{L}$,我们有等价关系 $S \subseteq [\![\varphi]\!] \Leftrightarrow \alpha(S) \leqslant_A [\![\varphi]\!]^\#$,则称抽象语义 $[\![\cdot]\!]^\#$ 相对具体语义 $[\![]\!]$ 强保留 $\mathscr{L}$。特别地,设 $\delta = (\Sigma, I)$ 和 $\delta^\# = (A, I^\#)$ 分别是 $\mathscr{L}$ 的具体和抽象语义结构,即由它们可以分别得到具体语义域 $\wp(\Sigma) = C$ 及抽象语义域 A 的具体语义 $[\![\cdot]\!]_\delta$ 函数和抽象域语义 $[\![\cdot]\!]_{\delta^\#}$ 函数,这时若有上面等价关系,则称 $\delta^\#$ 相对 δ 关于 $\mathscr{L}$ 强保留。

我们下面的定理表明 $\delta^\#$ 关于 $\mathscr{L}$ 是强保留,当且仅当抽象语义 $\langle A, [\![\cdot]\!]_{\delta^\#} \rangle$ 是前向完备的[16]。

在某种意义上强保留也是抽象域的属性,因此当上述定理成立时,也称抽象域 A 是语言前向完备的。然而为了验语言前向完备性是困难,文献[16]引入抽象域 A 上运算子方式前向完备性概念(精确意义见文献[16]),这时一般的运算子方式前向完备性可以推出语言前向完备(即强保留),但反之未必成立。然而抽象域 A 被具体语义 $[\![\cdot]\!]_\delta \mathscr{L}$-覆盖时(精确意义见文献[16]),$A$ 关于 $\mathscr{L}$ 语言前向完备就与 A 是运算子方式前向完备性等价。运算子方式前向完备性相对语言前向完备性较容易检验,只要证实了运算子方式前向完备性就可以得到强保留性质。

在上面理论框架下,文献[16]讨论了基于抽象解释的抽象模型检验问题,其详细叙述请查看原文。

值得注意的是,对于析取抽象模型,文献[17]还举例说明对于不动点计算,若用标准抽象模型(抽象 Kripke 结构)相对于基于抽象解释一般方法所需要的迭代次数更多,则说明基于抽象解释的一般方法更有效。

3. 标准强保留算法的基于抽象解释的分析和推广

文献[17]首先叙述了一个 GI:$(par, uco(\wp(\Sigma))_\supseteq, par(\Sigma)_\geqslant, pcl)$,其中 $par(\mu) = \{[s]_\mu \mid s \in \Sigma\}$,$pcl(P) = \{\cup_i B_i \mid \{B_i\} \subseteq P\}$。这样分割作为抽象领域,并且如果定义 $puco(\wp(\Sigma))$ 为所有可分割的 uco 集合,则 $puco(\wp(\Sigma))_\subseteq \cong par(\Sigma)_\leqslant$ 通过 par 和 pcl 两个映射。

如果语言 $\mathscr{L}$ 的公式运用语义结构 $\Im = (\Sigma, I, AP, l)$ 加以解释,则通过上述等价性可得:P 关于 $\mathscr{L}$ 强保留当且仅当 $pcl(P)$ 关于 L 强保留。这样便为把强保留看作前向完备性属性提供了一个合理框架。文献[17]的目标是在 $par(\Sigma)$ 上实行前向完备壳 $\mathscr{S}_F(\mu) = gfp(F_\mu)$ 的完备抽象计算,其中 $F_\mu : uco(\wp(\Sigma)) \to uco(\wp(\Sigma))$,$F_\mu(\rho) = M(\mu \sqcup F(\rho))$。$F(\rho)$ 是 F 在 ρ 上的像集。

基于上述思想,文献[17]分析了标准 Paige and Tarjan 算法(以下简称 PT):$PT(P)$ 是 PT 算法在输入 $P \in par(\Sigma)$ 上的输出,在上述抽象解释理论的框架下它可以看作是 $pcl(P)$ 关于 $\{pre, C\}$ 前向完备壳通过映射 par 的抽象,即:$PT(P) = par(\mathscr{S}_{\{pre,c\}}(pcl(P)))$。

通过分析 PT 算法的基本步骤,文献[17]从抽象解释角度推广它,得到一个一般的前向完备壳的一般抽象算法,称为 GPT(a generalized Paige-Tarjan refinement algorithm)。简略地说,当抽象域 A 和语义算子 F(关于 $\mathscr{L}$ 语言)满足一定的条件,以 F 和 A 为参数的一般 GPT_F^A 算法终止而且对任意 $a \in A, GPT_F^A(a) = \alpha(\mathscr{S}_F(\gamma(a)))$,其中 $(\alpha, uco(\wp(\Sigma))_\supseteq, A_\geqslant, \gamma)$ 是一个 GI。

文献[17]还利用 GPT 算法分析了标准模型检验算法:

(i) simulation 等价的 Henzinger et al.'s 算法。

(ii) Stuttering 等价的 Groote-Vaandrager 算法。

(iii) 强保留表达可达到性的(即包括算子 EF)语言 $\mathscr{L}$ 的算法。

上述分析把标准基于分割模型检验方法中求关于语言 $\mathcal{L}$ 的最优(即最粗分割)$P_{\mathcal{L}}$ 问题统一在广义 GPT 算法中。

4. 基于抽象解释的一个有效的模拟算法

设 $K=(\Sigma,\rightarrow,l)$ 是原子命题集合 AP 上的 Kripke 结构,其中 Σ 是状态空间,$\rightarrow$ 是 Σ 上转换关系,l 是标签函数(labeling function)。如果 $R\subseteq\Sigma\times\Sigma$,$\forall s,s'\in\Sigma$,$(s,s')\in R$ 满足下述两个条件:(a)$l(s)=l(s')$;(b)对任意的 t,若 $s\rightarrow t$,则存在 $t'\in\Sigma$,有 $s'\rightarrow t'$ 且 $(t,t')\in R$,则称 R 是 simulation 模拟关系。若 $(s,s')\in R$,则称 s' 模拟 s。显然当 $R=\emptyset$ 时,R 也是一个模拟关系,而且模拟关系的并也是模拟关系。所以最大的模拟关系存在。最大模拟关系是一个 Preorder relation,称为 K 上的 simulation preorder,并记为 R_{sim}。模拟等价 $\sim_{sim}\subseteq\Sigma\times\Sigma$ 是 R_{sim} 的对称归约,即 $\sim_{sim}=R_{sim}\cap R_{sim}^{-1}$,由 $\sim_{sim}$ 产生的分割 $P_{sim}\in par(\Sigma)$,称为模拟分割。

根据模型检验的熟知的结果,关于 $\sim_{\mathrm{sim}}$ 把 K 归约,可以定义抽象 Kripke 结构。$\mathcal{A}_{\mathrm{sim}}=\langle P_{\mathrm{sim}},\rightarrow^{\exists},l^{\exists}\rangle$。它强保留时态逻辑 ACTL*。其中 P_{sim} 是抽象状态空间,$\rightarrow^{\exists}$ 是模拟等价(集合)块之间的抽象转换关系,$\forall B\in P_{\mathrm{sim}}$,$l^{\exists}(B)\triangleq l(s)$,$s\in B$ 是任意的代表。$\forall\varphi\in\mathrm{ACTL}^*$,$\forall s\in\Sigma$,$s\models^K\varphi$ iff $P_{\mathrm{sim}}(s)\models^{\mathcal{A}_{\mathrm{sim}}}\varphi$,至于怎样求 P_{sim} 和 R_{sim},许多文献都提出了算法,其中 HHK 算法和 Gentilini 等人提出的算法都是很好的例子(参见[18])。但是,文献[18]在抽象解释理论框架下,证明了 simulation preorder 能够作为集合并和前像转换算子(predecessor transformer)两种运算的前向完备壳。回到 $K=(\Sigma,\rightarrow,l)$,设 l 产生状态分割 $P_l=\{[s]_l\mid s\in\Sigma\}$。运用 Moore-closure 算子于 P_l,得抽象域 $\mu_l\triangleq M(P_l)\in uco(\wp(\Sigma))$。实际上,简单地增加空集 $\emptyset$ 和整个状态空间 Σ 到 P_l 里便形成 μ_l。文献[18]提出定理:设 $\mu_K=S_{\cup,pre}(\mu_l)$ 是 μ_l 关于 $\{\cup,pre\}$ 操作的前向完备壳,则 $R_{\mathrm{sim}}\triangleq\{(s,s')\in\Sigma\times\Sigma\mid s'\in\mu_K(\{s\})\}$ 和 $P_{\mathrm{sim}}=par(\mu_K)$。

在上述定理基础上,文献[18]提出新的模拟算法,称为 SA。它的时间复杂度 $O(|P_{\mathrm{sim}}||\rightarrow|)$,空间复杂度 $O(|P_{\mathrm{sim}}||\Sigma|\log|\Sigma|)$,因此在时间、空间复杂性方面(与其他算法相比)都是有效的。例如 HHK 的空间复杂度是 $O(|\Sigma|^2\log|\Sigma|)$。

SA 是对 HHK 算法的修改,它主要是基于分割—关系序对概念(精确定义见文献[18])和基本程序(称之为 BasicSA,精确意义见文献[18])而得到的。这里值得一提的是,虽然分割——关系序对也在一些文献例如 Henzinger 等和 Gentilini 等里得到运用(参见文献[18])。但它们和文献[18]的应用具有不同的特征,文献[18]把 PRs 逻辑地看待为抽象域并运用抽象解释的理论改进 HHK 算法以及证实自己算法 SA 的正确性。因此文献[18]表明基于抽象解释理论对标准算法的改进是有效的,并且还通过实验证实 SA 算法是有效的。

4 综合方法

在引言里,我们提过验证软硬件主要有 4 个基本方法体系:(物理)模拟、测试、模型检验和定理证明,因此在模型检验里结合其他方法特别是定理证明将是非常重要的。但问题是,怎样结合这两种具有不同风格推理方式形成一个框架,以便人们能够自然地整合利用每种方法已获得的结果。也许这个框架也要基于抽象解释理论。实际上现今许多学者对抽象模型检验的基础和原则进行了研究,出现特殊的抽象解释形式,例如 Graf 和 Saidi 的谓词抽象,实现了抽象的自动化。基于谓词抽象的技术已应用到软件系统的模型检验中,诸如验证 Java 程序的 Bandera 和 Java PathFinder 以及验证 C 语言的 SLAM、MAGIC 和 BLAST(参见文献[11])。现在我们把人们在这个方面的努力,总结为"综合方法"加以介绍。

4.1 谓词抽象

文献[19]运用 PVS 为无限系统构造抽象状态图。

1. 基本定义(Processes):Name:P;Declarations:$x_1:T_1$,$\cdots$,$x_n:T_n$;Transitions:$\mathcal{T}_1,\cdots,\mathcal{T}_P$;Initial states:$init$。

其中 P 是过程名;$\forall i\in[1,n]x_i$ 是具有 Type T_i(可以是 PVS 定义的任意类型)的变量;它们(实质上)是全局变量;每一个转换 $\mathcal{T}_i$ 是一个具有下述形式被保护的赋值操作:

$g_i(\vec{x})\perp\rightarrow\vec{x}:=ass_i(\vec{x})$

其中,$g_i(\vec{x})$ 是 PVS 布尔表达式,$ass_i(\vec{x})$ 是由类型 T_i 的 PVS 表达式 ass_{ij} 组成的元组。$Init$ 是初始状态谓词。

2. 具体(过程)语义:关于分析由许多过程组成的平行系统,在某种意义上仅考虑单一过程 P 并不丧失一般性。P 定义一个状态图。$S_P=(Q_P,R_P,I_P)$,其中

- $Q_P=T_1\times\cdots\times T_n$
- $R_P=\bigcup_{i=1}^{P}\mathcal{T}_i$,其中 $\mathcal{T}_i(q)=\begin{cases}\perp, & \text{if } g_i(q)=\text{false}\\ ass_i(q), & \text{otherwise}\end{cases}$
- $I_P=\{q\mid init(q)\equiv\text{true}\}$ 是初始状态集合。

3. (具体语义操作)谓词转换算子。设 R 是 Q 上二元关系,谓词 $\varphi\in\mathcal{P}(Q)$ 表达 Q 上的一个子集,则定义

- $post[R](\varphi)=\exists q'\cdot R(q',q)\wedge\varphi(q')$
- $\widetilde{pre}[R](\varphi)=\forall q'\cdot(R(q,q')\Rightarrow\varphi(q'))$

$post[R](\varphi)$ 表达强后置条件;$\widetilde{pre}[R](\varphi)$ 是弱前置条件。

4. 抽象状态图。设 $S=(Q,R_P=\bigcup\mathcal{T}_i,I)$ 是程序 P 的状态图。$\boldsymbol{Q}^A$ 是抽象状态空间(结构是格),并且 $\alpha:\mathcal{P}(Q)\rightarrow\boldsymbol{Q}^A$,$\gamma:\boldsymbol{Q}^A\rightarrow\mathcal{P}(Q)$ 是 Galois connection。称 $S^A=(\boldsymbol{Q}^A,\bigcup\mathcal{T}_i^A,I^A)$ 是 S 的一个抽象状态图,当且仅当

- $I\subseteq\gamma(I^A)$
- $\forall I,\forall Q^A\in\boldsymbol{Q}^A$,$post[\mathcal{T}_i](\gamma(Q^A))\subseteq\gamma(\mathcal{T}_i^A(Q^A))$

在上述基本概念基础上,由具体程序 P 的变量决定 l 个谓词 $\{\varphi_1,\cdots,\varphi_l\}$,可以选择由 l 个布尔变量 $B_1,\cdots,B_l$ 决定的谓词集合作为抽象状态空间,其中每一个变量 B_i 表达所有满足谓词 φ_i 的具体状态集合。于是

$\gamma(\exp^A(B_1,\cdots,B_l)=\exp^A[\vec{\varphi}/\vec{B}]$

由此推出抽象函数 α 的定义:

$\alpha(\varphi)=\wedge\{\exp^A(B_1,\cdots,B_l)\mid\varphi\Rightarrow\exp^A[\vec{\varphi};/\vec{B}]\}$

随后对每一个具体转换 $\mathcal{T}_i$,定义一个抽象转换函数 $\mathcal{T}_i^A$ 为:

$\mathcal{T}_i^A(\exp^A)\triangleq\alpha(post[\mathcal{T}_i](\gamma(\exp^A)))$

以及确定初始抽象状态。

然而上述抽象函数 α 和抽象转换函数 $\mathcal{T}_i^A$(一般来说)都比较难算,所以大都采用近似方法,并且在计算过程中,运用 PVS theorem prover 和 PVS-interface 等工具,具体计算方法参见文献[19]。

最后关于抽象状态空间搜索方法和抽象状态图实现算法大都也是采用近似方法,例如对于可达到状态(不变式)可以定义不同的(向上)近似,详情参见文献[19]。

文献[19]还提出抽象状态图构造算法以及有关它们的改良以及谓词 $\varphi_1,\cdots,\varphi_l$ 的选择方法。其中抽象状态图构造中最昂贵的部分是一个抽象后代的计算,因为它要求若干个有效性检查,这可以递交给 PVS 定理证明器。但是仅仅相对较小的状态图才能够被构造,虽然为转换关系增加的存储代价是可以忽略的。在这个意义上,文献[19]提出的方法也是整合 PVS 定理证明器和决策程序的应用。

一个抽象状态图能够在模型检验中验证由一些时态逻辑公式(但不包含有存在量词的路径公式)表达的属性。它还能运用到其他领域。特别是它引进系统的守卫命令,使抽象状态图构造更有效,并使其成为控制图而得到应用。最后文献[19]运用这个方法,自动验证了一个有约束重复发送协议(bounded retransmission protocol)的正确性。

4.2 模型检验和定理证明

1. 抽象模型检验通过假设-承诺与定理证明结合

为证实无限状态反应系统的任意线性时间时态逻辑(例如 LTL 和 ACTL)属性,文献[20]提出一个方法,该方法结合数据抽象、模型检验和定理证明,其关键是通过假设-承诺和解除假设的推理方式把抽象模型检验与定理证明结合起来。文献[20]的基本思想总结在下面定理里。

定理 3[20] 设 M 和 M' 是两个转换系统,A_E 是环境假设(environment assumption),A_A 是抽象系统假设,C 是用 LTL 公式给出的承诺(commitment)。

如果:(1)$M' \models (A_A \wedge A_E) \rightarrow C$

(2)$M \models A_A$

(3)h 是 M 和 M' 之间同态

都满足,则 $M \models A_E \rightarrow C$。

非形式地说,如果抽象系统 M' 在假定 A_A 和 A_E 条件下满足承诺 C,并且抽象系统假设 A_A 在具体系统 M 里能够被解除以及抽象系统 M' 模拟具体系统 M(通过从 M 到 M' 的同态映射 h),则具体系统 M 在环境假设 A_E 成立时也满足承诺 C。如果抽象系统 M' 是有限系统(这正是我们抽象的目标),则第 1 个条件可以运用抽象模型检验证实。因为一般地,具体系统 M 都是无限的,所以后两个条件必须依赖定理证明去检验。

现在把文献[20]的方法概括为下面 4 个步骤。

(1)建立抽象转换系统。给定一个无限状态反应系统,由于它往往是面向控制的,故数据往往并不占据重要地位。所以通过状态空间每一个领域(例如整数,集合,队列)的数据抽象可以形成抽象有限状态空间。一个领域数据抽象包括从具体数值到抽象数值的映射和相应于具体数值运算的抽象数值运算。于是从原先具体系统的描述可以得到抽象系统的描述。由此产生抽象转换系统。通常要求它是有限的,并且可以由用户给出。

(2)确定系统应该满足的承诺。待检验的性质即承诺由 LTL 公式指定。由 ACTL 公式指定的性质也可以类似处理。

(3)抽象模型检验。假设模型检验支持假定-承诺风格的推理并能产生反例。我们初始化当前假设集合为空集,迭代进行下述过程:如果抽象状态系统在当前假设集合下满足承诺,我们就能够继续在具体系统里解除假设。如果承诺不满足,我们分析由此产生的反例,看它是否是相应于具体系统真实的程序错误或者是强加于原程序不真实的反例。对于前者我们调整具体系统并且从头再开始。对于后者,我们寻找一个假设以便排除反例,并增加它到原先的假设集合形成当前的假设集合并检查现在系统在已增加的假设下是否满足承诺。在这个过程中,人们当然也可以作出决策:是否需要改变数据抽象,并从头开始整个过程,即使用传统的利用反例改良抽象模型的方法。但这个方法并不是文献[20]提出的方法的主要精神。文献[20]只注重提出假设去消除反例。简言之,这里是抽象模型检验—伪反例—提出新假设合并到假设集里的循环,即使原来的抽象模型是非常粗糙的。

(4)假设解除证明。假设集中的每一个假设限制环境或是系统本身的行为。为了保证系统假设仅仅排除了由于近似抽象而增加在抽象系统的行为,人们必须证明具体系统满足这些假设而环境假设也必须在具体系统运行时加以解除。由于系统是无限的,必须运用定理证明器去证明具体系统满足假设。同样定理证明器也用来建立具体系统和抽象系统的同态对应。不过定理证明器典型地需要和用户相互作用以及用户的专业知识。

文献[20]最富有特色的部分是它并不依赖抽象的"颗粒状"(granularity),它主要采用通过伪反例增加假设的方法消除由于抽象强加于具体系统的"伪"行为。这样给用户产生抽象模型提供了很大的自由度。然而过于粗糙的抽象会导致过强的假设,这样解除它的证明将更困难。因此该方法也是在寻找合适抽象和自动性方面的一种权衡。另外在执行语言里定义称谓属性变量。在某种意义上,它是程序注释虚变量,该变量在抽象模型的(形式)假设的构造中是有用的。并且在具体系统里证明假设时,利用归纳法可以分解证明并且归并到 Hoare-triples 简单范式,即前置条件成立并且经过一步程序行动则后置条件成立。总之在一定程度上,文献[20]实现了模型检验和定理证明有效的自动结合。

2. 通过 PVS 集成模型检验和定理证明

文献[21]的目标是在 PVS 的内容里集成模型检验和定理证明。现在把文献[21]的主要想法和基本做法叙述如下。

设 Σ 是有限符号集合,每一个符号或者是命题变量或者是谓词变量(具有 n-ary,$n>0$)。语法项分为公式和关系两个范畴。关于公式项和关系项的语法定义不再详细写出。实际上如此定义的命题 mu-clalculus 是命题 calculus 的扩展,它包含形如 $\exists z.f$ 或 $\forall z.f$ 公式(其中 f 是公式),以及形如 $\mu z.P[z]$ 或 $\nu z.P[z]$ 的分别由最小最大不动点运算定义的谓词,其中 z 是一个 n-ary 谓词变量,$P[z]$ 是一个关系项,它正规单调于 z(即在 $P[z]$ 里,z 出现在符号¬下,是偶数次)。命题 mu-calculus 能够一般化为 mu-calculus。粗糙地说,即在命题 mu-calculas 里引入关于类型的表达和计算。实际上,mu-calculus 提供了一个框架来表达时态逻辑,例如 CTL 和它的 fairness 延展 fairCTL 以及其他时态。对于时态逻辑,例如 CTL 运算子,运用 mu-calculus 给出它们通常的(不动点)定义,例如(其中 N 表示 next-state 关系):

$(EXP)(x)=\exists z.P(z)\wedge N(x,z)$

$(EGP)(x)=(\nu Z.(\lambda z.P(z)\wedge(EXZ)))(x)$

$(E(pUq))(x)=(\mu Z.\lambda z.q(z)\vee(p(z)\vee(EXZ)(z)))(x)$

运用 PVS 简单类型高阶逻辑描述语言,表达和推广 mu-calculus。例如[$S\rightarrow T$]表达从类型 S 到类型 T 的函数类型;

[T→*bool*]简写为 PRED[T],表达在类型 T 上谓词类型,其中 *bool* 是由 TRUE 和 FALSE 组成的布尔类型。并且谓词逐点序<=定义为:如果 P_1,P_2 是类型 PRED[T],则定义(P_1<=P_2)=(FORALL(x:T):$P_1(x)$ IMPLIES $P_2(x)$)。这时称 P_1 强于 P_2,或者 P_2 弱于 P_1。

文献[21]提升逻辑运算到谓词运算。定义在 T 上的谓词转换为类型[PRED[T]→PRED[T]]。一个谓词转换是单调的,是指它在谓词上保持<=序。如果 PP 是单调的谓词转换,则 mu(PP)和 nu(PP)分别表示 PP 的最小和最大不动点。于是 CTL 运算子也能够定义。它们参数化在 next-state 关系 N 上,N 的类型是[T,T→*bool*]。例如

EX(N,f)(u):*bool*=(EXISTS v:($f(v)$ AND $N(u,v)$))

EG(N,f):PRED[T]=*nu*(LAMBDA Q:(f AND EX(N,Q)))

EU(N,f,g):PRED[T]=*mu*(LAMBDA Q:(g OR (f AND EX(N,Q)))

关于 fairness 的定义,最简单最有用的是 fair paths 概念,它不能在 CTL 表达,却很容易在 mu-calculus 中定义。设 fairEG(N,f)(h)(u)断定一个 fair path 存在,它从 u 出发沿着该路径 f 成立且 h 无限次经常成立。用 mu-calculus 在 PVS 里将它定义为:

fair EG(N,f)(h)=*nu*(LAMBDA P. EU(N,f,f AND h AND EX(N,P)))

对于 fairness 的清晰表达使得我们能够检验 fair paths 的存在性。可以定义 PVS 这样的片断(fragment):它能够转换到有限状态类型以及从基本类型归纳定义的结构形成的 mu-calculus。

同样,我们也能从 PVS 转换到命题 mu-calculus,这是必须要做的,因为低层次模型检验只能接受命题 mu-calculus 语言,并且这些转换在 PVS 里是自动实现的。通过 mu-calculus,模型检验和定理证明已经"光滑地"集成于 PVS 里,模型检验器作为 PVS 的一个良定义片断部分的判定程序,是实现文献[21]的目标最基本的一步。

值得注意的是,以上的叙述都是基于有限情况。

运用模型检验器验证有限状态系统的 CTL 或其他 mu-calculus 性质,现在已经是直接的事情了。系统的状态在 PVS 里表达为有限类型。系统用一个初始谓词和一个 next-state 关系加以描述。系统的性质能够在 CTL 或其他能够运用 mu-calculus 定义的操作项中表达,并且运用一个称为 model-check 的命令就可以证明。

实际上,一个复杂系统的验证,牵涉到抽象、归纳和组合推理。文献[21]运用 PVS 的基本想法是找到表达系统属性的一种形式化方法,即用 PVS 的"片断"语言,形式刻画时态逻辑及其运算,以及抽象、归纳、分解和组合方法,以便把模型检验(适用于有限转换系统)方法作为一个决策过程融于 PVS 证明体系之中。虽然如此,该方法仍然存在缺陷,例如最小和最大不动点计算交替时,状态呈指数增长。

定理证明和模型检验的结合的运用基础是抽象的运用。文献[21]还用一个小型实例说明在抽象基础上上述方法的使用。

3. 其他结合方法简介

因为定理证明检验和模型检验是相互补充的技术方法,因此结合它们是合理的。

实际上,Hol/Voss 系统[22]是这个方向的早期尝试。Hol 证明器[23]和 Voss 符号模型检验器连接在 HOL/Voss 系统里,Voss 建立(输入到 HOl)一些常量的属性,并把这些论断结果作为引理反馈给 HOL 证明器,而 HOL 对它们进行处理。

基于 HOL-Voss 系统,文献[22]为形式硬件验证提供符号计算与定理证明相结合的方法。符号计算是基于 BDD 关于状态轨迹的论断证实,从而获得有关 circuit 行为和同步调速的精确模型和高程度的自动化。定理证明是基于与用户互动和用户的专业知识,使得我们能够运用强有力的数学工具和手段,例如归纳和抽象。为了发展这种混合方法,他们花了很多精力开发"证明"的基础结构,例如数学和逻辑运算、HCL(higner-level constraint language)以及一般化证明程序,并且用一个实例说明他们两个层次证明系统的优越性。除此之外,文献[22]混合方法的优点是,适用于不同水平的用户,即对系统的掌握技能不同,使用系统完成的任务也不同。缺点是两个系统的联系较松散,Voss 建立的属性不能直接被 HOL 证明。

Kurshan 和 Lamport 也在 TLP 定理证明器和自动推理(automata-theoretic)模型检验器 COSPAN 之间建立了一个类似连结[24]。TLP 验证用行为时态逻辑 TLA(Temporal logic of actions)编写的模型,COSPAN 验证用 S/R 语言编写的模型,COSPAN 是语言容纳系统。可以容纳符号模型检验器(language containment verifier based on BDDS)。在原则上,模型在上述两个语言里都可以编写且能相互转换。这两个语言都很简单,且有类似的基本语义基础。

证实一个复杂系统的关键是分解它的证明。在 TLA 编写的模型里,一个系统的说明能够分解为它的各个成份的说明的联合。例如我们可以用 TLA 公式 $E\wedge B_1\wedge\cdots\wedge B_n$ 表达由 n 个成份组成的系统,其中 E 是环境说明,B_i 是第 i 个成份的说明。如果要求系统满足某个性质 F,我们(例如利用抽象)可以为每一个成份 i,写出它的高阶(high-level)说明 M_i,然后证明:

$\vdash E\wedge M_1\wedge\cdots\wedge M_n\Rightarrow F$

$\vdash E\wedge B_1\wedge\cdots\wedge B_n\Rightarrow E\wedge M_1\wedge\cdots\wedge M_n$

前式用标准的 TLA 推理即用 TLP 工具证明,后式可以用一个分解定理(Decomposition Theorem)得到。而分解定理的假设可以用不同的方法加以证实,或者用 TLP 证明,或者用模型检验(此时 TLA 公式应转换 S/R 公式)加以证实(特别是牵涉到低层次的模型),而且分解定理的结构表明(因为它的结论和它的部分假设具有相同的构造)它还可以适用于递归方法。这样,文献[24]提出的方法就在定理证明中尽可能使用了模型检验方法,从而使定理证明简化。虽然如此,但通过以上叙述,可以看出这两个系统并没有真正结合在一起。文献[24]证明,8-bit 乘法器能够运用 COSPAN 证实,而由 8-bit 乘法器组成的 N-bit 乘法器能够运用 TLP 证实,并通过 64 Bits 的乘法器的验证说明问题。

Hungar 描述了一个类似方法,其中运用模型检验去证实过程的属性,而用语法形式化 MCTL 去证实各个过程的组合。还有其他相关工作,例如 Müller 和 Nipkow 在 I/O 自动系统里关于模型检验和演绎的结合[25]等,不再赘述。

4.3 其他方面的努力

由于模型检验越来越受到人们的关注，人们还从不同的方面对模型检验的原理和方法进行了探索。下面尽我们所知介绍若干这方面工作。

1. 时态抽象解释。文献[26]提出基于轨道时间对称时态模型：

S：states　$\mathbb{P} \triangleq \mathbb{Z} \mapsto S$ paths

$T \triangleq \mathbb{Z} \times \mathbb{P}$ trances　$M \triangleq \wp(T)$ temporal model

即把时态公式解释为无限时间对称轨道的集合，从而定义由转换系统产生的程序基于轨道的语义。运用上述模型，推广 μ-calculus 为 $\overset{\frown}{\mu}^*$-calculus，使它具有可逆和抽象模态和新的时间对称轨道语义。文献[26]研究了基于轨道模型，诸如原点封闭、前向封闭、后向封闭和状态封闭等时态模型概念。并开发了一般的具体和抽象语义结构，即它们参数化适用于一般的语义域和语义操作算子。为此特别处理单调和反序，即引入肯定性(positiveness)符号 $P::=+\mid -$ 于语义转换算子和抽象解释，这样抽象解释的合理性就可以按通常熟知的方式表达而且对抽象语义类型而言一次就可以将所有情况进行处理，抽象解释的合理性和完备性、不完备性也可以一起进行讨论。在最一般的经典的具体语义域上，基于集合的语义能够作为基于轨道的语义的抽象解释，这样便使我们能够讨论模型检验和数据流分析问题，从而讨论时态抽象解释的合理性、完备性和不完备性。文献[26]指出产生不完备性的根源。但是按 CTL 方式定义的子逻辑(sublogics)却是相对完备的。

时态抽象解释模型具有很多优点。至少在抽象解释性质的理论证明时具有一般性的优点：

i. 由于时态模型关于时间是对称的，因此关于前向、反向性质的讨论可以统一处理。

ii. 抽象解释理论中的合理性和完备性的讨论更具有一般的格式，并对检验抽象(checking abstraction)这一概念作了深刻阐述。

iii. 推广了 μ-calculus 和一些其他逻辑。

2. (counterexamples)反例研究

违反指定性质的反例为系统调试工作提供了重要信息，并且许多抽象模型检验都借助伪反例去改进模型，因此反例的提供是模型检验的一个重要优点。

然而一般文献很少把精力投入在反例研究上，或者只局限于相对简单的反例。例如，由 Clarke 和他的同事以及 Hojati 和他的同事提出的生成线性反例的算法广泛应用于实际以后，很少有人专门从事这方面的研究。现在，文献[27]为反例研究提供了一般的框架。

假设 Kripke 结构 K 违反了一个 ACTL 公式 φ，即 $K \not\models \varphi$。我们对反例 C 的要求是：

(i)C 违反 φ；

(ii)φ 在 C 上的违反应该能够说明 K 对 φ 的违反；

(iii)C 是切实可行的。

根据上面 3 个合理要求，可以得到反例的形式定义。

也就是把上面 3 个假设分别形式化叙述如下：

I. $C \not\models \varphi$，或者等价地说 $C \vdash \neg\varphi$。

注意 $\neg\varphi$ 是 ECTL(ACTL 的对偶(dual))里的公式，于是 C 是 $\neg\varphi$ 的见证(witness)。

II. $K \geqslant C$。注意 $\geqslant$ 是模型检验里的有关模拟关系符号，模拟关系保留 ACTL 公式。即对 ACTL 公式 φ，若 $K \geqslant C$ 且 $K \vdash \psi$，则 $C \vdash \psi$。于是，若 C 违反 ACTL 公式 $\varphi = \neg\psi$，则 K 也违反 φ。

通过 II 也可以看出在一般文献里也只产生线性反例并通过 Kripke 结构模拟执行。

III. C 是树型反例。所谓树型反例是特殊的 Kripke 结构[27]。转换关系 SCCs(强连接成份 strongly connected components)在树型反例里形成一个有限树。并且每一个 SCCs 是有向循环的。详细定义见文献[27]第 4 节。简言之，它是以 SCCs 作为“节点”的有限树。要求如此结构是有效的且切实可行的，即要求它满足下面 3 个条件：

• 完备性(completeness)：C 对于包括 ACTL 在内的时态语言很大的一类应该是完备的，即对 ACTL 每一个指定公式违反，在 C 里都有一个反例成为见证。

• 智能性(intelligibility)：C 里每一个元素都应该简单和明确，致使人们容易分析。

• 有效性(effectiveness)：应该存在(可能是符号)有效算法使得人们在 C 里产生并操作反例。

文献[27]证明了一个重要定理：ACTL 有树型反例。由此得到推论：ECTL 有树型模型性质。

并且文献[27]给出 CEX 算法，给定 K, s 和 φ，如果 $K, s \not\models \varphi$，则调用 $\mathrm{CEX}(K, s^0, \varphi)$ 为 $K, s \vdash \varphi$ 计算树型反例。

文献[27]分析了 ACTL 和 ACTL* 一个微妙细节即线性时间运算的单调性，以及一些重要概念，例如 w-正则式，详情参见文献[27]。在上述概念基础上，文献[27]提出 $A\Omega$ 逻辑，其中 Ω 是时态运算子的集合(可能是无限的)，它是全称分枝线性时间逻辑，ACTL 和 ACTL* 可以定义为 $A\Omega$ 具有有限联合(conjunction)的子逻辑。直观上，$A\Omega$ 是带有 w-正则线性时间运算子关于 ACTL 等逻辑的扩展，文献[27]证明了它有树形反例且能有效计算。

3. 接口组合模型检验

为了解决由许多并行过程组成的系统关于时态逻辑模型检验的复杂性问题，一般的文献都是从检验系统的组成成份的属性，然后推出全局的属性这样的方法论着手。这种方法论的主要困难在于局部的属性未必在全局上也能够保持。文献[28]提出，运用接口过程(interface processes)去模拟系统成份的环境，组合成份和接口过程，然后检验这种组合的性质，就能保证它们局部属性也在全局上保持。

设 P_1 和 P_2 是两个过程，系统是 P_1 和 P_2 的并行组合，记为 $P_1 \parallel P_2 \triangleq \mathscr{P}$，$A_1$ 和 A_2 是分别附加于 P_1 和 P_2 关于 P_2 和 P_1 的接口过程。

文献[28]的思想完全包括在下述接口规则里：

$$P_1 \downarrow \Sigma P_2 \equiv A_1 \quad P_2 \downarrow \Sigma P_1 \equiv A_2$$

$$\varphi \in \mathscr{L}(\Sigma_{P_2}) \quad \psi \in \mathscr{L}(\Sigma_{P_1})$$

$$\frac{A_1 \parallel P_2 \vdash \varphi \quad P_1 \parallel A_2 \vdash \varphi}{P_1 \parallel P_2 \vdash \varphi}$$

其中，φ 是逻辑 $\mathscr{L}$ 的公式(表达系统要检验的性质)，每一公式都是从某些原子命题集合构造出来的，例如 $\varphi \in \mathscr{L}(\Sigma_{P_2})$ 表示包含在 φ 里的原子命题都是 Σ_{P_2} 的子集，而 Σ_{P_2} 表示联系于过程 P_2 的原子命题集合。记号 $P \downarrow \Sigma$，例如 $P_1 \downarrow \Sigma_{P_2}$ 表示 P_1 过程约束到 Σ_{P_2} 形成的过程，它由 Σ_{P_1} 里隐藏不属于 Σ_{P_2} 的符号所形成。

上述规则的精神是，A_1 联系于 P_1，但它却是 P_2 的接口，同样 A_2 联系于 P_2，但它却是 P_1 的接口。直觉上，A_1 是所有

通过线路由 P_2 能够观察到的 P_1，它是 P_2 的环境，故 $A_1 \equiv P_1 \downarrow \Sigma_{P_2}$，因而由 $A_1 \| P_2 \vdash \varphi$，就能够推出 $P_1 \| P_2 \vdash \varphi$（注意 $\varphi \in \mathcal{L}(\Sigma_{P_2})$）。类似的关系在 A_2 与 P_1 之间也成立。由于接口过程通常要比（一个组成成分）整个环境小得多，这样就减少了检验复杂性。关于接口规则的合理性，文献[28]提出 4 个充分条件。

文献[28]在 CTL* 逻辑上给出两个组合系统模型检验的例子：异步过程模型和逻辑、同步模型和逻辑，并分别证明它们满足可以运用接口规则的 4 个条件。

4. 离散时间-连续时间实时系统模型检验

文献[1]讨论两种形式的实时系统模型检验。对于离散时间实时系统，在 CTL 时态算子中引入界限（bounds）。例如 $f \sqcup_{[a,b]} q$ 公式，它就是由有界限制的 U（until）算子产生的。$[a,b]$ 定义一个时间区间，在这个区间内性质必须成立。于是我们说 $f \sqcup_{[a,b]} q$ 在轨道 $\pi = s_0, s_1, \cdots$ 上是成立的，就是指在轨道某一个将来状态 s 上 q 成立，并且在所有从 s_0 到 s 之间的状态上 f 成立，而且还要求从 s_0 到 s 之间的距离在区间 $[a,b]$ 内，这种扩展的逻辑称为 RTCTL，于是离散时间系统的模型检验就是关于 RTCTL 的模型检验。除此之外，还介绍了其他处理方法以及关于最小延迟和最大延迟两个算法。

关于连续时间实时系统，文献[1]介绍了定时自动机（Timed Automata）模型，并把连续时间实时系统模型化为定时自动机的并行组合系统。为了解决状态空间无限问题，文献[1]介绍了两种方法：Clock Regions 和 Clock Zones，后者还能用差分有界矩阵（Difference Bound Matrices）简洁表达。

5 应用和其他重要方法概览

前面几节，我们沿着标准方法-抽象（解释）方法-综合方法线索，介绍了几十年来人们在模型检验方面的主要工作。本节着重介绍一些和它们有关的重要应用以及一些其他重要方法。由于篇幅限制，这些介绍知识是梗概性的。

5.1 模型检验理论在程序分析中的应用简介

1. Mycroft 严格性分析（Mycroft's strictness analysis）

Mycroft 描述了关于函数程序的严格性分析，所谓严格性分析在于确定当它的项没有定义时，函数的结果是否没有定义，该分析对加速序列或平行执行“懒”（lazy）函数语言是有用的。抽象解释方法能运用到该分析中（参见文献[10]），例如文献[10]介绍了 G. L. Burn 和他的同事关于高价函数严格性分析的抽象解释并指出从语义角度推出抽象解释的一种实际可行的途径。一般来说，抽象语义主要关心严格性和终止性分析。

2. 逻辑程序的“基础”（groundness）分析

基础分析是基于逻辑程序语言最重要的分析之一，主要思想是静态（statically）检查逻辑变量和谓词项在运行时期是否被约束到基础项。它是逻辑变量和谓词项之间基础依赖性的一般化。抽象解释方法也能运用到该分析中，例如文献[8]提出的“intelligent disjunctive refinement of ground-dependency analysis”方法。

3. 数据流分析：Cousot 等人提出基于布尔抽象解释的数据流分析，Schmidt，Steffen 和 Cousot 从不同方面对数据流分析进行阐述（参见文献[26]），此外文献[26]也从时态规范的抽象解释角度对它进行讨论。

5.2 其他重要方法

1. 不用 BDDS 的符号模型检验

Biere 等提出用 SAT 程序替代 BDDS 进行符号模型检验（参见文献[6]）。Marques-Silva 等运用 GRASP 搜索算法，也是出于同样目的[29]。

2. 其他逻辑和语法

McMillan；Burns and Godefroid；Sagiv 等提出在模型检验里运用 3 值逻辑，它能给出正面和反面正确论断，但是可能用未知或近似结论终止（参见文献[6]）。

Maidl 还研究了用 ACTL∩LTL 逻辑公式描述系统属性，例如当这些属性违背时仅能产生线性反例；Kupferman 和 Grumberg 研究了 Buy one Get one Free 逻辑；Thomas 研究了具有 ω-正则算子的计算树时态逻辑；Sistla 研究以量词限定的时态逻辑（quantified temporal logic or SIS）等等，参见文献[27]。此外 Shtadler 和 Grumberg 提出网络语法，Vigna 和 Ghezzi 提出图语法，等等，参见文献[1]。这些逻辑和语法都具有各自特点，或者针对特定目的而设计的，例如它们之中有许多是为了解决无限系统验证问题而设计的。

3. 向上、向下近似处理 CTL

Lind-Nielsen 和 Andersen 提出向上、向下近似处理整个 CTL。他们的方法可以在每一步改良以后避免重新检查整个模型而保证完备性，参见文献[6]。

4. 变量依赖图和最小函数图

例如 1993 年 Balarin 和 Sangiovanni-Vincentelli 在初始抽象和改良过程中运用变量依赖图，抽象运用变量距离概念（参见文献[6]）；Jones 和 Mycroft 在数据流分析中运用最小函数图（参见文献[10]）。

5. 局部改善方法

Granger 提出局部减少迭代方法去改善程序静态分析结果，其要旨是改善抽象简化算子的逻辑合取（参见文献[10]）。Kurshan 提出的局部简化实际上是基于反例改良抽象模型的最早研究之一，基本概念是 L-过程（参见文献[6]）。他把并发系统用 L-过程 $L_1, \cdots, L_n$ 的组合模型来表示，局部简化是一个迭代技术，它开始于有重大关系的 L-过程的一个较小子集，这些子集在变量依赖图里与性质指派说明较接近，所有其他程序变量用非确定性指派抽象掉。如果发现伪反例，则增加额外变量去消除这个反例。至于选择额外变量也是根据变量依赖图的信息，例如根据变量接近性质指派的距离进行决策。

6. Polyhedrul 方法

基于 Polyhedrul 的抽象解释，可以运用到（线性）实时系统和混合系统的抽象分析里（参见文献[19]）。

由于篇幅所限，还有许多模型检验方法和应用，例如 Giacobazzi 和 Mastroeni 在他们理论里有关抽象解释领域的简化和压缩方法的应用[30]；Cousot，P. 和 Cousot. R 基于 PER（直观解释为部分等价关系）的抽象解释框架及其应用[7]；Classen 和 Legay 研究模型检验软件生产线（SPL）行为，提出了表达 SPL 行为的特征转换系统，并给出相关算法[31]；Fillieri 等人发展了实时概率模型检验[32]；等等。在此就不介绍了。

结束语 模型检验是一个重要的软硬件验证手段，至今已经形成一个庞大的方法体系，我们分为标准方法、抽象解释方法和综合方法 3 个部分加以介绍，由于抽象解释最初引入是作为对标准方法的一种近似，因此也把它称为非标准方法。在这种分类框架下，我们试图总结模型检验在各个方面的成果，我们认为这样介绍容易形成人们对模型检验总的印象，从

而全面地掌握模型检验的方法。当然由于文献太多，这样做也可能遗漏一些方法，但是我们相信，模型检验主要方面已经包括在其中了，这对于今后的研究是有帮助的。

参 考 文 献

[1] Clarke E M Jr, Grumberg O, Peled D A. Model checking [M]. Cambridge, Massachusetts, London, England, The MIT Press, 1999

[2] Lichtenstein O, Pnueli A. Checking that finite state concurrent programs satisfy their linear specification [C]// Proceedings of the 12th Annual ACM Symposium on Principles of Programming Language. ACM, 1985: 97-107

[3] Clarke E M, Emerson E A, Sistla A P. Automatic verification of finite-state concurrent systems using temporal logic specifications [C]// Proceedings of the 10th Annual ACM Symposium on Principles of Programming Language. January 1983

[4] Emerson E A, Lei C-L. Modalities for model checking: Branching time strikes back [C]// Twelfth Symposium on Principles of Programming Languages. New Orleans, La., ACM Press, January 1985: 84-96

[5] Bryant R E. Graph-based algorithms for boolean function manipulation [J]. IEEE Transactions on Computers C, 1986, 35(8): 677-691

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

[7] Cousot P, Cousot R. Systematic design of program and analysis frameworks [C]// Conference Record of the 6th ACM Symp. on Principles of Programming Languages (POPL'79). New York: ACM Press, 1979: 269-282

[8] Giacobazzi R, Ranzato F, Scozzari F. Making abstract interpretations complete [J]. Journal of the ACM, 2000, 47(2): 361-416

[9] Cousot P, Cousot R. Refining model checking by abstract interpretation [J]. Automated Software Engineering Journal, special issue on Automated Software Analysis, 1999, 6(1): 69-95

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

[11] 钱俊彦，徐宝文. 基于完备抽象解释的模型检验 CTL 公式研究 [J]. 计算机学报，2009，32(5)：992-1001

[12] Ranzato F, Tapparo F. Strong preservation as completeness in abstract interpretation [C]// Proceedings of the 13th European Symposium on Programming (ESOP). LNCS 2986. Barcelona, Spain, 2004: 18-32

[13] Ranzato F, Tapparo F. Generalized strong preservation by abstract interpretation [J]. Journal of Logic and Computation, 2007, 17(1): 157-197

[14] Giacobazzi R, Ranzato F. Example-guided abstraction simplification [C]// Proceedings of the 37th International colloquium conference on Automata, Languages and Programming ICALP 2010. Part II, LNCS 6199. 2010: 211-222

[15] Ball T, Podelski A, Rajamani S K. Boolean and cartesian abstraction for model checking C programs [J]. International Journal on Software Tools for Technology Transfer, 2003, 5(1): 49-58

[16] Ranzato F, Tapparo F. Strong preservation of temporal fixpoint-based operators by abstract interpretation [C]// Proceedings of the 7th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI). LNCS 3855. Charleston, SC, USA, 2006: 332-347

[17] Ranzato F, Tapparo F. An abstract interpretation-based refinement algorithm for strong preservation [C]// Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS 3440. Edinburgh, UK, 2005: 140- 156

[18] Ranzato F, Tapparo F. An efficient simulation algorithm based on abstract interpretation [J]. Information and Computation, 2010, 208(1): 1-22

[19] Graf S, Saidi H. Construction of abstract state graphs with PVS [C]// Proceedings of the 9th International Conference on Computer Aided Verification (CAV), LNCS 1254. Haifa, Israel, 1997: 72-83

[20] Dingel J, Filkorn T. Model checking for infinite state systems using data abstraction, assumption-commitment style reasoning and theorem proving [J]. Lecture Notes in Computer Science, 1995, 939: 54-69

[21] Rajan S, Shankar N, Srivas M K. An integration of model checking with automated proof checking [J]. LNCS, 1995, 939: 84-97

[22] Joyce J J, Seger C-J H. Linking Bdd-based symbolic evaluation to interactive theorem proving [C]// Proceedings of the 30th Design Automation Conference. Association for Computing Machinery, 1993

[23] Gordon M J C, Melham T F, et al. Introduction to HOL: a theorem proving environment for higher-order logic [M]. Cambridge, UK, Cambridge University Press, 1993

[24] Kurshan R P, Lamport L. Verification of a multiplier: 64 bits and beyond [C]// Proceedings of the 5th International Conference on Computer-Aided Verification. London, UK, Springer-Verlag, 1993: 166-179

[25] Müller O, Nipkow T. Combining model checking and deduction for I/O automata. Draft manuscript, 1995

[26] Cousot P, Cousot R. Temporal abstract interpretation [C]// Proceedings of the 27th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Programming Languages (POPL). Boston, USA, 2000: 12-25

[27] Clarke E, Jha S, Lu Yuan, et al. Tree-like counterexamples in model checking [C]// Proceedings of the IEEE Symposium on Logic in Computer Science (LICS). Copenhagen, 2002: 19-29

[28] Clarke E M, Long D E, McMillan K L. Compositional model checking [C]// Proceedings of the 4th IEEE Symposium on Logic in Computer Science. Asilomar, CA., June 1989: 353-362

[29] Marques-Silva J, Sakallah K A. GRASP: A search algorithm for propositional satisfiability [J]. IEEE Transactions on Computers, 1999, 48(5): 506-521

[30] Giacobazzi R, Mastroeni I. Transforming abstract interpretations by abstract interpretation (Invited Lecutre) [C]// Alpuente M, Vidal G, eds. SAS 2008. LNCS, 2008, 5079: 1-17

[31] Classen A, Legay A. Symbolic model checking of software product lines [C]// Proceedings of the 33th International Conference on Software Engineering. Waikiki, Honolulu, HI, USA, 2011: 321-330

[32] Filieri A, Ghezzi C, Tamburrelli G. Run-time efficient probabilistic model checking [C]// Proceedings of the 33th International Conference on Software Engineering. Waikiki, Honolulu, HI, USA, 2011: 341-350