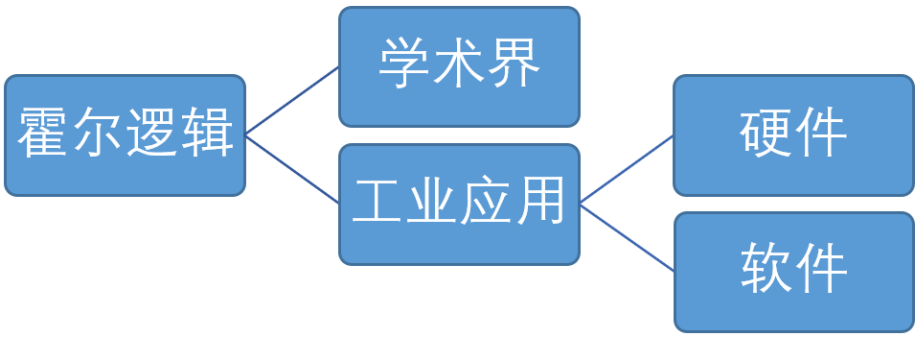
嵌入式系统 | 基于 SCADE 模型的形式化方法

作者：Ansys 中国 姚翔

本文将重点阐述“基于 SCADE 模型的形式化方法”，做个通俗的比喻，形式化方法就是将程序抽象为一个数学公式，然后用严密的数学推理来证实或证伪该公式。在当下软件行业已经有众多测试手段的前提下，为什么还需要形式化方法呢？

1972 年的图灵奖得主 Edsger Wybe Dijkstra 说道：“Program testing can be used to show the presence of bugs, but never to show their absence! ”，即**测试只能表明程序中存在错误，而不能表明程序中没有错误**。除非对程序进行的测试能够穷尽所有可能的场景，否则传统的测试手段无法完全保证系统的安全可靠。**可以说，唯有形式化方法才能从根本上确保系统的安全可靠，而这一点在安全关键的系统中尤为重要。**

1960 年代，霍尔逻辑（Hoare Logic）是第一个关于形式化方法的学说，从那时起的很长时间，形式化方法主要应用于学术界，后来再慢慢地拓展到了工业应用的硬件领域。**客观地说，形式化方法在电子硬件世界中有更多的成功应用——主要是因为硬件工具更标准化和稳定，而软件领域还未达到那样的程度**。软件领域的系统设计、高层需求对应的模型，可能因为不具备足够的细节，无法对一些属性进行有意义的分析，形式化方法的应用效果、实用价值一般。**目前，形式化方法在详细设计层面（Low-level requirement），对于软件行为等模型较为适用。**



形式化方法的发展与应用

/ 形式化方法的定义

机载软件的适航标准第三版 DO-178C 的补充文档《DO-333，DO-178C 和 DO-278A 的形式化方法补充》(下文简称为《DO-333》)中对**形式化方法的定义为：用于构造、开发和推理一个系统行为的数学模型的描述性符号系统和分析方法。简言之，形式化的方法=形式化模型+形式化分析。采用形式化方法的验证手段的最重要优势是其完备性，它能从数学逻辑上完全证明系统设计的正确性。**然而，形式化方法的缺点同样令人印象深刻，它需要对原始设计进行反复提炼、抽象、提取、精化，最终得到其数学模型，这一过程对使用者的数学技巧、项目理解和工程经验都有较高的要求。

/什么是形式化模型？

形式化模型是形式化方法的基础，模型必须有明确的、数学定义的语法和意义，才能成为形式化的模型。形式化方法对任何特定软件开发活动的适用性受到构建适当的形式化模型能力的限制。自然语言，半形式化的语言可能包括不能用形

式化方法推理的属性，在许多情况下，就不能完全模拟软件。因此，**定义模型的限制是有必要的。**

有了这些限制后，形式化模型就可以是一种完整和准确描述重要软件属性的手段。然后，可以用形式化的分析来提供这些属性的保证，并与其他验证技术(例如，测试)相结合，以实现所有验证目标的满意度。

形式化方法可以用于项目开发生命周期的许多阶段，并满足各种验证目标。无论在何处使用形式化的模型，都应该验证它们，以保证它们就是软件的准确表示。这意味着对于被分析的属性，如果它们适用于模型，那么它们也适用于实际的软件。

/什么是形式化分析?

虽然在软件开发周期中创建形式化的模型已经有明显的好处，但是**形式化方法的最大优势在于对这些模型进行的形式化分析。形式化分析可以提供软件属性和软件符合需求的保证(guarantees)或证据(proofs)**。提供保证或证据意味着所有可能的运行情况都被考虑在内，即完备性。为了进行形式化分析，一组软件属性是必要的。这些属性可以被创建出来，也可以被嵌入到要实现形式化分析的特定工具中(例如，最坏运行时间分析)。**形式化分析的属性在许多方面可以被视为类似于可验证的测试需求**。在测试中，测试用例应该很容易从可验证的需求中导出，**对于形式化分析，属性通常也是待验证的测试用例**。可以为一组需求定义一组完整的属性，在这种情况下，该组属性集总是成立的证据，证明了所分析的模型与其需求的等价性。

值得一提的是，仅当对属性的确定性分析是坚实可靠的分析，才可称为形式化分析。坚实可靠的分析意味着：当一个属性可能不为“真”时，形式化方法从不断言它为“真”。相反，断言一个可能是“真”的属性为“假”时，则是可以接受的。因为这是一个可用性的问题，而不是可靠性的问题。通俗地讲，引起“产生错误的警报”是允许的。**此外，在试图证明属性是否成立时，形式化方法的结果返回“不知道”或者不返回结论也是可以接受的。**在这种情况下，就需要额外的验证(精化属性、调整分析引擎等)。

因为形式化的方法=形式化模型+形式化分析，所以软件行业内绝大部分基于模型的开发工具，如果不做专门的定制或转换，由于模型本身不是形式化的，也就无从谈起形式化的方法。而 SCADE 由于其支撑语言发轫于形式化的语言 Lustre 等，**SCADE 是主流商业工具里唯一的、真正的、形式化的模型**，再结合第三方形式化分析引擎(Prover，S3，SafeProver，GATel 等)，在不同行业已经得到较多形式化方法验证的成功案例。

/ 形式化方法的分类

上述提到的《DO-333》文档中标识了三类典型的形式化分析，其定义分别如下：

抽象解释 Abstract Interpretation（也有称之为静态分析 Static Analyzer）：定义构造一个编程语言语义的保守表示，用于“有限状态程序的动态属性的合理确定……它可以被看成一个计算机程序的部分执行，它确定了程序的特定效果(例如，控制结构、信息流、堆栈大小、时钟周期数)，而不实际执行所有的计算”。

抽象解释的方法相对简单，可用静态分析工具检查除 0(division-by-zero)，越界(out of bound array access)，溢出(Overflows)等问题，但可能产生虚假警告。

相关的主流商业工具有 AbsInt 的 Astree，Mathworks 的 Polyspace Code Prover 和 TrustInSoft 的 TrustInSoft Analyzer。

模型检查 Model checking：搜索一个形式化模型的所有行为，以确定一个指定的属性是否满足。

模型检查用于判定系统的形式化模型是否满足基于规约(Specification)的正确属性。模型检查通常有三个结果：1. 属性正确；2 属性错误，并给出反例；3. 分析超时（TO: Time Out）无法确定。相对于抽象解释，模型检查可提供了安全属性的更好的表现形式。然而由于对非线性算法，状态组合空间爆炸等处理能力的不足，其使用范围也受限制。对工程师而言，模型检查最大的价值在于当安全属性违反后，会自动生成反例。当安全属性无法验证时，它能显示导致该错误状态的运行记录。

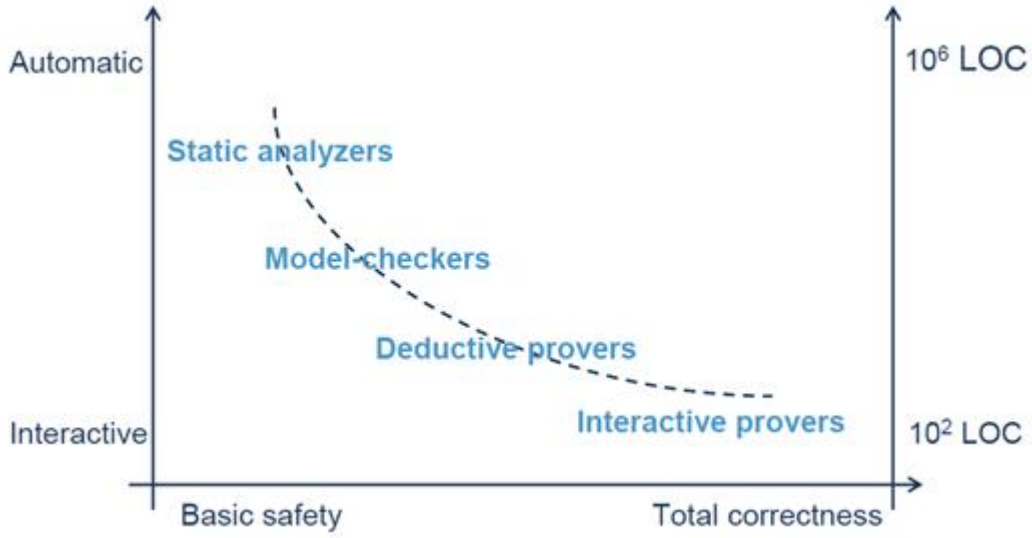
相关的主流商业工具有 Kind2 (University of Iowa) ， JKind (Rockwell Collins) ， CEA 的 GATeL 和 Prover Technologies 的 Design Verifier 等。

演绎法 Deductive proof：使用数学论证来建立一个形式化模型的每个属性。通常使用一个理论证明工具来构造证明，以显示软件属性是成立的。

演绎法是使用数学参数来判定每个形式化模型的属性，最终目标是定理证明(Theorem Proving)，证明通常是基于 Hoare 逻辑和 Dijkstra 先决条件推理。相对于抽象解释和模型检查，演绎法是更有效的。然而该方法通常要求非常专业的人员参与，自动化程度较低，即使很普通的推理验证也经常需要耗费数周甚至上月的时间，难以在大规模的项目上应用，甚至在某些情况下是根本不可能使用的。相关的主流工具有 STeP(Standford Theorem Prover)、ACL2 、HOL 、PVS 、TLV 、Coq 等。

以上三类形式化分析具有一些共性。**首先，它们都依赖于形式化的模型**。非形式化模型，或非形式化模型和形式化模型混合的复合模型都不足用形式化的方法来表明规约和代码之间的符合性，所以要求必须是真正的形式化的模型。**第二，所有的形式化分析通常都是用一个工具实现的。**

下图是形式化分析的全景图，可见三种方法的自动化程度和可处理的代码量由高到低排列，但安全属性证明的完备性由低到高排列。



根据 Xavier Leroy 改写的形式化方法全景图

三种形式化方法中抽象解释和模型检查的工程实践应用较为广泛，SCADE 中也早已经内嵌了基于抽象解释和模型检查两种技术的形式化分析工具(属于 AbsInt 公司和 Prover 公司等)。抽象解释一般可通过简单的按钮操作进行标准化的分析来快速得出结果，对此就不作进一步描述了；而模型检查结果的得出还是有些工作量的，部分复杂项目可能还会耗费工程师们不少功夫才能成功应用。下面就对模型检查进行展开介绍。

/ 模型检查（Model checking）技术的现状

模型检查作为一种包含了一系列面向有限状态系统验证技术的框架，最早由 Clarke、Emerson、Quielle 和 Sifakis 在 1981 年分别提出，他们中的部分人也因此获得了 2007 年的图灵奖。

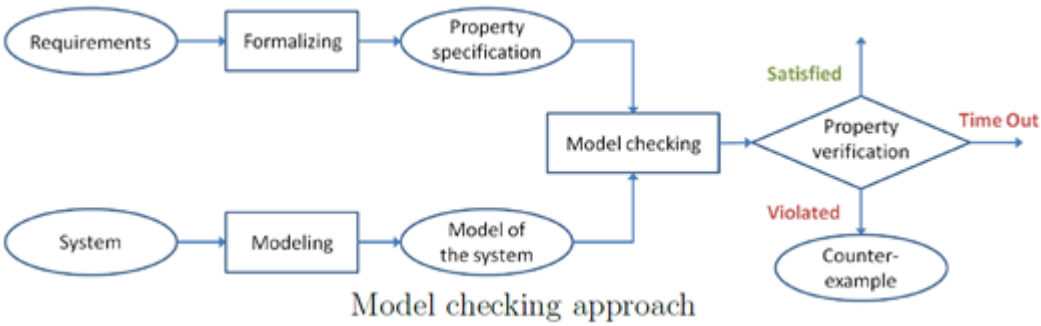
年份	中文译名	姓名	贡献领域
1971 年	约翰·麦卡锡	John McCarthy	计算的理论,LISP
1972 年	艾兹格·迪科斯彻	Edsger Dijkstra	最弱前置条件演算
1974 年	高德纳	Donald E. Knuth	LP Parser
1976 年	迈克尔·拉宾 达纳·斯科特	Michael O. Rabin Dana S. Scott	非确定性自动机、指称语义
1977 年	约翰·巴克斯	John Backus	程序设计语言规范的形式化定义
1978 年	罗伯特·弗洛伊德	Robert W. Floyd	前后断言法
1980 年	东尼·霍尔	C. Antony R. Hoare	Hoare 逻辑公理化方法
1984 年	尼古拉斯·沃斯	Niklaus Wirth	程序设计语言的形式描述
1986 年	约翰·霍普克罗夫特	John Hopcroft	形式语言
1991 年	罗宾·米尔纳	Robin Milner	LCF，ML 语言，CCS
1996 年	阿米尔·伯努利	Amir Pnueli	时序逻辑，程序与系统验证
2005 年	彼得·诺尔	Peter Naur	BNF
2007 年	爱德蒙·克拉克 艾伦·爱默生 约瑟夫·斯发基斯	Edmund M. Clarke Allen Emerson Joseph Sifakis	模型检查 注: 90 年代 Joseph Sifakis 负责的 VERIMAG 联合实验室开发了 SCADE
2008 年	芭芭拉·利斯科夫	Barbara Liskov	抽象数据类型
2013 年	莱斯利·兰伯特	Leslie Lamport	TLA

部分图灵奖获得者在形式化验证方面的研究

模型检查方法主要通过遍历系统的有穷状态空间以验证系统是否满足某些关键的系统属性。模型检查的输入通常为两个，以自动机(automata)模型描述的系统和以时序逻辑公式(temporal logic formula)描述的系统属性。其中模型实现了系统的行为，属性描述了系统应该做什么、或者不应该做什么。

数学上可描述为:设 M 代表含一系列状态的系统，φ表示系统待验证的属性。要检查系统 M 是否满足属性φ，首先将属性φ取非，得$\neg\varphi$。再将$\neg\varphi$转换为对应的自动机$A_{\neg\varphi}$，则该自动机接受的计算都是满足$\neg\varphi$的。然后，再将代表系统的自动机A_M和取非属性的自动机$A_{\neg\varphi}$进行合并，即$A_M \otimes A_{\neg\varphi}$，形成$A_{M,\neg\varphi}$。此时，任何被新自动机$A_{M,\neg\varphi}$接受的运行，即意味着系统 M 中存在违背属性$\varphi$的行为。模型检查的时间复杂度为$|M| \cdot 2^{O(|\varphi|)}$。

对$A_{M,\neg\varphi}$进行非空检测：结果为空，意味着不存在这样的违背属性的行为，系统满足该验证属性；结果非空，则意味着存在违反属性的行为，可以返回所谓的反例(违背属性φ的路径)。如果状态空间太大，也可能运行超时(TO: Time Out)，结果不确定。如下图所示模型检查的输出结果有三种，属性满足，属性违反(提供反例)和超时结果不确定。



模型检查的方法

系统模型和待验证的属性都需要以高度抽象和精确严密的方式进行描述。近年来学术界以混成系统(Hybrid System)的概念来描述工业界安全关键领域的嵌入式系统，其特点是系统的行为里既包含连续的时间行为(可用微分方程描述的流表示)，又包含离散的逻辑控制(可用状态机或自动机描述的跳转表示)，而且连续的时间行为和离散的逻辑控制并不相互独立，而是糅合交织在一起的，共同构成描述真实系统的复杂逻辑。从此前《细数 Ansys SCADE 的前世今生》一文可以看出，SCADE 是从工业界的真实应用推动而发展起来的，它恰是包含了微分方程、状态机和自动机等多种形式化语言的融合产物，可以满足混成系统的要求。因此，SCADE 是形式化方法中实现系统模型的较好工具之一。

而属性描述语言通常是用时序逻辑公式(temporal logic formula)。时序逻辑公式是一种为涉及时间的陈述和推理而量身定制的数学形式，相当接近自然语言，它还附带了形式化语义。常见的时序逻辑公式是线性时间逻辑(LTL: Linear Temporal Logic)和计算树逻辑(CTL: Computation Tree Logic)。LTL 和 CTL 的组合称为 CTL*。

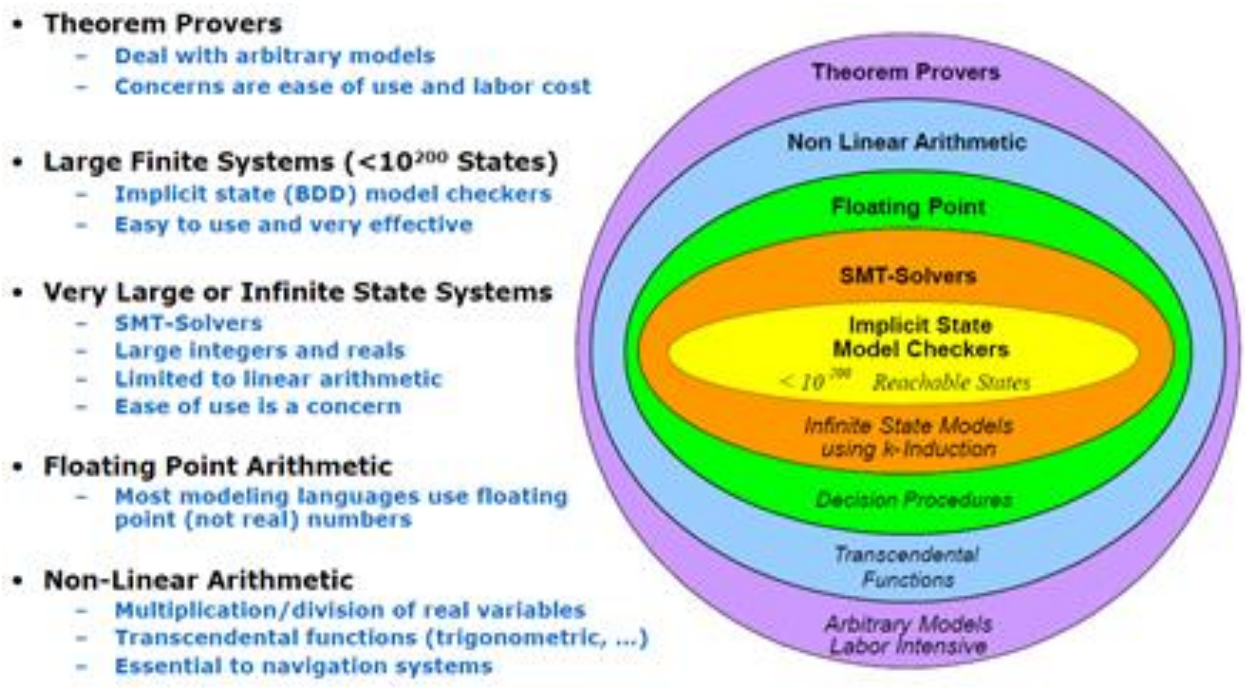
迄今为止，模型检查的算法大致发展了三代。

第一代是显式/状态模型检查(Explicit/state model checking)方法，它可以显式地探索模型的状态空间，并通过正向探索进行搜索，直到发现属性冲突。该方法对于 LTL 和 CTL 提出了不同的思路。在 LTL 模型检查中，探索算法可以是深度优先(depth-first)或广度优先(breadth-first)的：广度优先的方法可以找到最短的可能反例，但对内存使用的要求明显高于深度优先的方法。在 CTL 模型检查中，通过递归地计算每个状态是否满足的子公式，来确定满足给定属性的所有状态集。如果所有的状态都被访问并且没有检测到违反，则判定该模型满足该属性。毫不意外，第一代模型检查方法了就遇到了状态空间爆炸问题。

第二代是符号模型检查(symbolic model checking)方法，该方法使用一种称为二元决策图(BDD：Binary Decision Diagrams)的特殊数据结构，可以更有效地表示状态集和函数关系，从而允许表示更大的状态空间。

第三代是有界模型检查(BMC: Bounded Model Checking)方法，作为基于二元决策图（BDD: Binary Decision Diagram)的符号模型检查的一种补充方法，有界模型检查技术被学者提出并得到了广泛的应用。**其主要思想是通过一个整数 k 来限制模型行为步数，将系统 k 步内的行为编码成一组布尔约束，然后基于 SAT(Boolean Satisfiability)方法对其求解从而高效地找出 k 步内的所有错误**。虽然 BMC 方法因为只能验证阈值内的系统属性而相对缺失了模型检查的完备性，但是通过限定模型的行为步数，其在发现错误的能力上超越了传统模型检查方法。此外，由于 BMC 方法只需要遍历阈值内的系统状态空间，使得其能处理的系统规模得到了提升，这也是 BMC 得到广泛认可的优势所在。

不同于半自动化甚至全手动的定理证明，模型检查可以自动执行并能在系统不满足属性时给出相应的错误信息，因此在工业界比其他形式化方法更受推崇。然而，由于模型检查技术的较高复杂性，时至今日依然难以处理业界的大规模系统。由罗克韦尔柯林斯公司发布的报告可以看到，模型检查技术对于非线性的、可达状态空间小于 10200 的有限情况能作出有意义的分析。

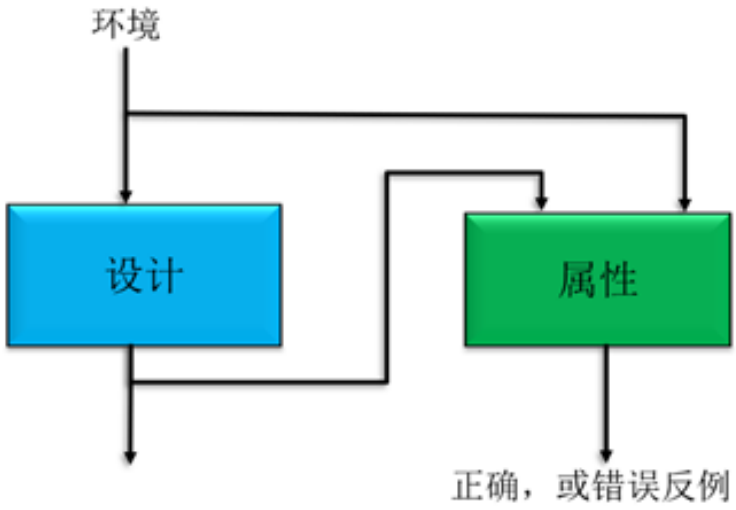


Miller, Steven P 关于模型检查技术的可扩展验证域图

/ 基于 SCADE 的模型检查

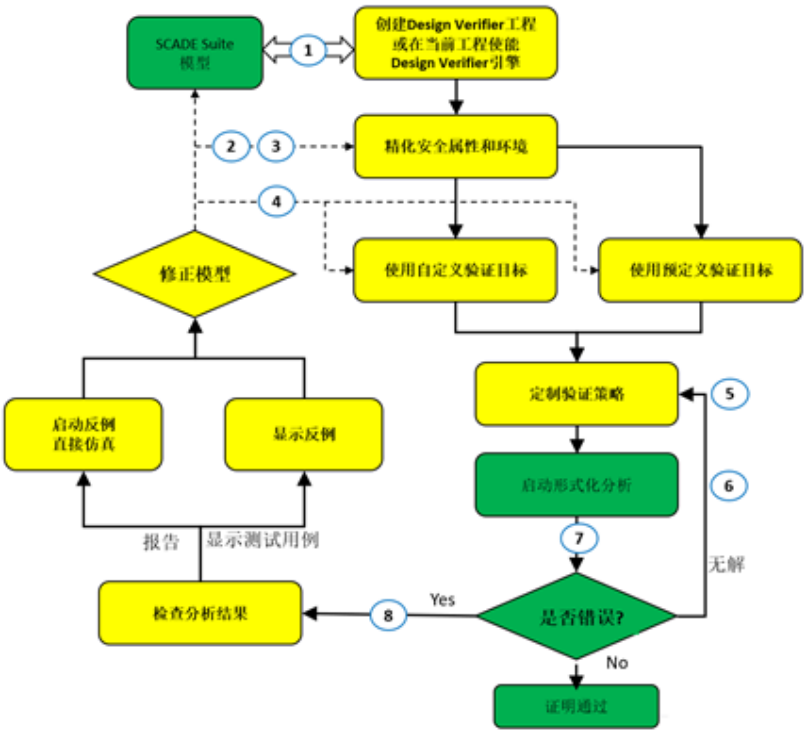
对 SCADE 模型进行形式化分析的绝大部分商用工具采用的就是基于 BMC 方法的模型检查技术。当前 SCADE 在 2020 R1 版本中包含了 Prover Technology 的 Design Verifier 引擎，Design Verifier 引擎里内置了 SAT 解算器，可以对 SCADE 模型进行模型检查，当结果为错误时，自动给出反例的测试用例。

使用 SCADE 模型实现形式化验证的通用方法是：保持原有设计模型不变，根据需求定义含安全属性的观察模型，再设计顶层操作符将设计模型和属性观察模型串联起来，最终分析属性模型的输出即可。



SCADE 模型检查方法简图

基于 SCADE 模型的形式化方法详细工作流如下：

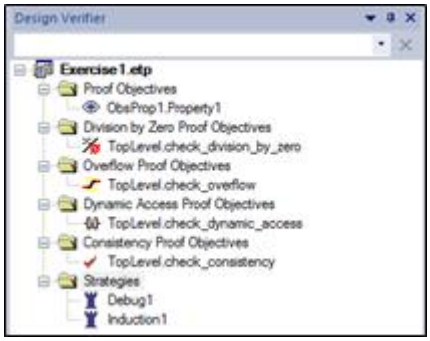


SCADE 模型的形式化方法工作流

1. 创建新的形式化验证工程，或在当前开发工程中使能形式化分析引擎，这样就能自动将形式化验证相关的 Suite 预定义库导入到工程中
2. 根据安全需求和使用场景定义观察者模型，其中包含的安全属性也是用 Suite 设计，该安全属性的输出必须设计为 bool 类型，以 true 值输出代表安全导向，供形式化引擎进行分析
3. 创建顶层操作符，将原设计模型和观察者/属性模型连接起来。可以在顶层操作符中使用 Assert 功能添加约束，排除无关的场景，以加速形式化分析过程
4. 根据安全属性创建自定义的形式化验证目标，或直接启用 Suite 内置预定义的形式化验证目标
5. 定制形式化验证策略，并将策略应用到形式化验证目标上
6. 启动形式化引擎进行分析
7. 输出形式化验证结果
8. 检查结果，为真则得证，为假，则导入给出的反例进行模型仿真，再基于仿真结果修正模型后，迭代进行下一轮形式化分析

步骤 4 中指出的 Suite 内置预定义的形式化验证目标有 4 种：

1. 除 0 检查：Division-by-Zero PO：验证是否会引起除 0 的可能
2. 溢出检查：Overflow PO:验证数值类型数据在赋值、移位或类型转换时是否会溢出等
3. 动态访问检查：Dynamic Access PO：验证数组索引是否越界等
4. 一致性检查：Consistency PO：验证假设和约束是否一致，约束可以由 Assume 功能添加



SCADE 形式化验证面板

步骤 5 中指出的形式化验证策略有如下 3 种:

基础策略 (缺省设置)	Prover引擎 基础策略	描述
Debug	调试	迅速查找反例并不证明安全属性的有效性 该策略查找反例的速度远高于Prove策略
Induction	归纳	使用模型检查中的k-induction技术，在用户指定的k个深度(需要用户自定义start depth和stop depth)内证明安全属性有效性。缺省时: depth start从0到无限
Prove	通用	彻底的证明。 如果不返回任何反例，即证明该安全属性的有效性

SCADE 形式化验证策略

Debug 策略和 Induction 策略中的属性及其含义如下:

标准选项	描述
Dump file	设置转存dump文件的路径: 仅调试使用，由SCADE专家服务团队可识别的内部格式生成
timeout	指定分析安全属性的时间，以秒为单位 (缺省为0,无最大时间限制)
深度选项	**描述**
Start Depth	在Debug策略或Induction策略中，指定k-induction的执行深度开始值（缺省为-1，自定义值从0开始，无最大限制）
Stop Depth	在Debug策略或Induction策略中，指定k-induction的执行深度终止值（缺省为-1，自定义值应该大于执行深度开始值，无最大限制，可无限循环，最终timeout超时时结束，或客户终端结束）

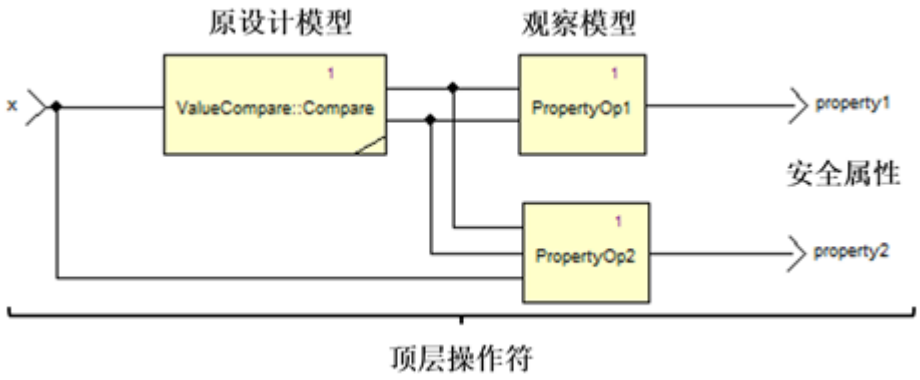
SCADE 策略的属性

SCADE Design Verifier 选项卡与安全属性策略

使用形式化验证策略有助于用户：

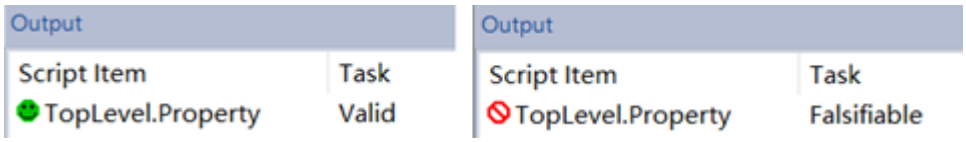
- ✓ 在开发过程早期迅速查找 bug
- ✓ 通过验证某个独立完整的需求对应的安全属性，证明需求是正确的
- ✓ 节省验证成本，提高产品可信度

某项目的形式化分析顶层操作符的设计如图表：



SCADE 某顶层操作符的设计

形式化分析的正确和错误的结果输出如下图所示：

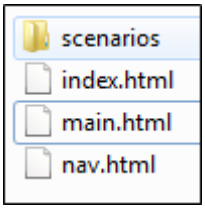


SCADE 形式化正确或错误的分析结果

当形式化分析是错误时，可在详细错误界面单击超链接跳转到该安全属性。也可在配置正确的前提下，直接单击 Load Scenario 直接导入反例给出的测试用例，进行白盒仿真。

SCADE 形式化分析错误后导入反例进行白盒仿真

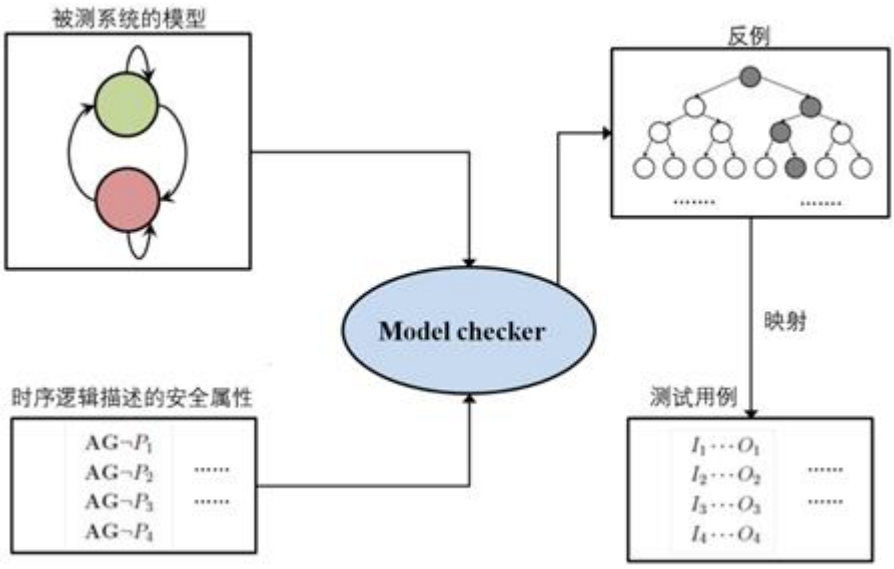
最后，可在 SCADE 程序中，或者工程目录下查看每次分析后的 Html 格式的形式化验证报告。



SCADE 形式化分析的目录级网页格式报告

使用模型检查技术生成覆盖分析相关的测试用例

模型检查用于判定系统的形式化模型是否满足基于规约(Specification)的正确属性。这是模型检查的常规用法。有趣的是，也有不少学者的研究提出，**可以使用模型检查技术生成覆盖分析相关的测试用例。该方法需要反向设计，故意设计一组“缺陷属性”(trap properties)，该缺陷属性声称：某分支无论如何运行，都不可能被覆盖的。**如果模型检查证明该“缺陷属性”为假，就能得到反例，而反例就是可以覆盖该分支的测试用例。数学上就是将上面描述的自动机进行非空检测，且此时结果应该为非空。



模型检查生成测试用例

同模型检查方法的正向使用一样，对“缺陷属性”的检查也会遇到状态空间爆炸的问题。在系统模型已经确定的前提下，**如何在尽可能短的时间内，设计出恰当的“缺陷属性”是提高测试用例生成效率的关键**。在后续文章中将介绍不同行业的客户在项目中结合 SCADE 工具进行的形式化方法应用。

小结

本文结合机载软件的适航标准第三版 DO-178C 的补充文档《DO-333，DO-178C 和 DO-278A 的形式化方法补充》介绍了基于 SCADE 模型的形式化方法。形式化方法=形式化模型+形式化分析。形式化方法主要分三类：抽象解释、模型检查和演绎法。**SCADE 是主流商业工具里唯一的、真正的、形式化的模型，可结合不同的第三方形式化分析引擎实现形式化方法**。SCADE 已内嵌了基于抽象解释和模型检查两种技术的形式化分析工具（AbsInt 公司和 Prover 公司等）。

值得一提的是，Ansys 与 Honeywell 在嵌入式系统领域已合作多年，未来将共同研发把 Honeywell 为航空项目而自研(in-

house)的形式化验证工具链 Hi-Lite 进行优化改进，使其逐步成为 SCADE 的标准模块。

/ 参考文献

[1] Leanna Rierson.安全关键软件开发与审定——DO-178C 标准实践指南[M]. (崔晓峰). 北京：电子工业出版社，2015 年 6 月

[2] Todorov V, Boulanger F, Taha S. Formal verification of automotive embedded software[C]//Proceedings of the 6th Conference on Formal Methods in Software Engineering. ACM, 2018: 84-87.

[3] Xavier Leroy Trust in compilers, code generators, and software verification tools [EB/OL].

[4] 宋俊. LTLNFBA: LTL 公式到 Büchi 自动机的转换[D]. 西安电子科技大学, 2014.

[5] 王瑞. 基于 SAT 的符号化模型检验技术研究[D]. 国防科学技术大学, 2014.

[6] Sun Y. Strengthening fonctional validation of critical system by using Model Checking: Application to Instrumentation and Control systems in nuclear power plants[D]. 2017.

[7] Li W, Kan S, Huang Z. A Better Translation from LTL to Transition-Based Generalized Büchi Automata[J]. IEEE Access, 2017, 5: 27081-27090.

[8] 刘志锋. 模型检测中关键技术的研究及其应用[D]. 南京大学, 2011.

[9] 解定宝. 混成系统有界模型检验优化技术研究[D]. 南京大学, 2016.

[10] Miller S P. Bridging the gap between model-based development and model checking[C]//International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg, 2009: 443-453.

[11] SCSDV_2019R1_M01_Tool. Ansys SCADE Suite Design Verifier and Formal Verification [EB/OL]. Ansys SBU. 2019（内部文档）

https://xavierleroy.org/talks/ERTS2018.pdf. Embedded Real Time Software and Systems, 2018-02-02

[12] Bhatt D, Madl G, Oglesby D, et al. Towards scalable verification of commercial avionics software[M]//AIAA Infotech@ Aerospace 2010. 2010: 3452.

[13] Wang J, Zhan NJ, Feng XY, Liu ZM. Overview of Formal Methods. Journal of Software, 2019, 30(1): 33-61(in Chinese). http://www.jos.org.cn/1000-9825/5652.htm

[14] 形式化方法导论[M]. 清华大学出版社 ，张广泉, 2015