PSL 逻辑及验证技术研究进展与展望 *

虞 蕾[1,2], 赵宗涛[2]
(1. 国防科学技术大学 计算机学院, 长沙 410073; 2. 第二炮兵工程学院 计算机系, 西安 710025)

摘 要: 在简要介绍 PSL 的分层结构和语法与语义基础上,综述了 PSL 验证技术的应用研究现状,分析了各种方法、技术的优缺点,最后指出了 PSL 验证技术的未来研究展望。

关键词: 属性规约语言; 基于断言的验证; 形式化验证; 运行时验证
中图分类号: TP301 **文献标志码**: A **文章编号**: 1001-3695(2010)07-2414-07
doi:10.3969/j.issn.1001-3695.2010.07.004

PSL logic and its verification technologies

YU Lei[1,2], ZHAO Zong-tao[2]
(1. *School of Computer, National University of Defense Technology, Changsha* 410073, *China*; 2. *Dept. of Computer Science, Second Artillery Engineering College, Xi'an* 710025, *China*)

Abstract: After the descriptions of the PSL hierarchical structure and its syntax & semantics, this paper reviewed in detail the recent PSL verification application technologies and finally pointed out some issues to be further studied.
Key words: PSL (property specification language); ABV (assertion based verification); formal verification; run-time verification

不同时序逻辑就语法、语义、表达能力和相应验证问题的复杂度四方面都有较大的区别。应用于规约的不同时序逻辑的不兼容性加剧了不同工具之间数据转换的复杂程度。工业领域对验证问题不断增长的研究兴趣,导致了人们对实现规约逻辑标准化目标的更多努力。其中,由 Accellera 组织提出并于 2005 年确定为 IEEE 工业标准的 PSL 就是一种重要的应用于硬件验证的标准化属性规约语言。PSL 主要有三大用途:形式化验证、动态验证仿真和功能规约描述(作为功能规约文档)。由于 PSL 易于读写、语法精简、语义清晰、表达能力强,能直接使用于高效的模拟和形式化验证算法,它作为一种功能描述语言得到了广泛的应用。

1 PSL 的分层结构及语法与语义

1.1 PSL 的分层结构

PSL 是一种分层语言,包括四层(图 1):a)布尔层(boolean layer),用于构造实现单个状态内评估的布尔表达式,是构筑时序表达式的基础;b)时序层(temporal layer),是语言的核心层,用于实现语言的主要功能,即描述设计模块的多个状态的时序属性;c)验证层(verification layer),是与验证工具紧密结合的语言层,用于指示验证工具验证时序层所描述的模型属性规约;d)建模层(modeling layer),为验证过程所需的测试输入模块或辅助硬件模块建模。验证层的主要验证指示词以及建模层的内建函数和建模指示词的功能和定义见文献[1]。图 2 用 PSL 表达的验证实例体现了 PSL 的分层结构。其中,assert 是验证层的验证指示词,用于指示验证工具验证时序层 always (req→next(ack)) 时序属性;assign 是建模层的赋值指示词,为定义的输入变量 req 赋值。

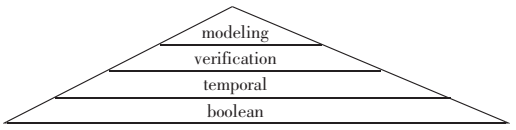

图 1 PSL 的分层结构

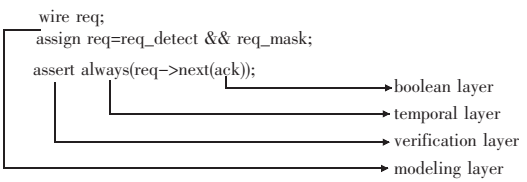

图 2 体现 PSL 分层结构的应用实例

作为 PSL 核心的时序层可分为 FL(foundation language,基础语言)和 OBE(optional branching extension,可选分支扩展)两部分。FL 如同 LTL,是线性时序逻辑,而 OBE 就是分支时序逻辑 CTL(computation tree logic)[2]。时序层的 FL 包含命题操作子、LTL 操作子、定义时间粒度的时钟操作子、连续扩展正规表达式(sequential extended regular expressions,SEREs)以及 abort 操作子。FL 公式还具有时钟声明的功能表示,具有时钟声明的公式很容易改写成非时钟公式(改写规则见文献[1])。

1.2 PSL 的语法与语义[1]

1.2.1 PSL 语法

定义 1 连续扩展正规表达式(sequential extended regular expression,SERE)。每个布尔表达式 b 是一个 SERE;如果 r、r_1、r_2 是 SEREs,则以下各项都是 SERE:

收稿日期: 2009-12-23; **修回日期**: 2010-02-11 **基金项目**: 国家"863"计划资助项目(2007AA010301);中国博士后科学基金资助项目(20080431401)

作者简介:虞蕾(1978-),女,浙江浦江人,讲师,博士后,主要研究方向为模型检验与航迹规划(yuleizj@163.com);赵宗涛(1942-),男,教授,博导.

· $\{r\}$　· $r_1;r_2$　· $r_1:r_2$　· $r_1|r_2$　· $r_1\&\&r_2$　· $[*0]$　· $r[*]$

定义 2　基础语言公式(foundation language formulas, FL formulas)。如果 b 是布尔表达式,则 b 和 $b!$ 都是 FL 公式;如果 φ 和 ψ 是 FL 公式,r 是 SERE,b 是布尔表达式,则以下各项都是 FL 公式:

· (φ)　· $-\varphi$　· $\varphi\wedge\psi$　· $r!$　· r　· $X!\ \varphi$

· $[\varphi\cup\psi]$　· φ abort b　· $r|\rightarrow\varphi$

定义 3　可选分支扩展公式(optional branching extension formulas, OBE formulas)。每个布尔表达式是一个 OBE 公式;如果 f、f_1、f_2 是 OBE 公式,则以下各项都是 OBE 公式:

· (f)　· EXf　· $-f$　· $E[f_1\cup f_2]$　· $f_1\wedge f_2$　· EG f

定义 4　Accellera PSL 公式。每个 FL 公式都是一个 Accellera PSL 公式;每个 OBE 公式都是一个 Accellera PSL 公式。

1.2.2　PSL 的语义

1)非时钟 SEREs 语义

$wL\ r$ 表示 w 紧满足(tightly satisfies)r。其中,w 是定义在 $\Sigma=2^p\cup\{j,b\}$ 上的有穷字,j 表示满足任意命题公式的状态,b 表示不满足任意命题公式的状态。原子命题 $p\in AP$,b 为布尔表达式,r、r_1、r_2 是非时钟 SEREs,则:

(1) $wL\ \{r\}\Leftrightarrow wL\ r$

(2) $wL\ b\Leftrightarrow|w|=1$ 且 $w^0L\ b$

(3) $wL\ r_1;r_2\Leftrightarrow\exists w_1,w_2,$ s.t. $w=w_1w_2,w_1L\ r_1$ 且 $w_2L\ r_2$

(4) $wL\ r_1:r_2\Leftrightarrow\exists w_1,w_2,l,$ s.t. $w=w_1lw_2,w_1lL\ r_1$ 且 $lw_2L\ r_2$

(5) $wL\ r_1|r_2\Leftrightarrow wL\ r_1$ 或 $wL\ r_2$

(6) $wL\ [*0]\Leftrightarrow w=\varepsilon$

(7) $wL\ r_1\&\&r_2\Leftrightarrow wL\ r_1$ 且 $wL\ r_2$

(8) $wL\ r[*]\Leftrightarrow wL\ [*0]$ 或 $\exists\ w_1,w_2,$ s.t. $w_1\neq\varepsilon,w=w_1w_2,w_1L\ r$ 且 $w_2L\ r[*]$

2)非时钟 FL 公式语义

式 $w\models\varphi$ 表示 w 满足 φ。其中,w 是一个有穷或无穷字,b 是布尔表达式,r 是非时钟 SERE,φ 和 ψ 是非时钟 FL 公式,则

(1) $w\models(\varphi)\Leftrightarrow w\models\varphi$

(2) $w\models-\varphi\Leftrightarrow-w\models\varphi$

(3) $w\models\varphi\wedge\psi\Leftrightarrow w\models\varphi$ 且 $w\models\psi$

(4) $w\models b!\Leftrightarrow|w|>0$ 且 $w^0L\ b$

(5) $w\models b\Leftrightarrow|w|=0$ 或 $w^0L\ b$

(6) $w\models r!\Leftrightarrow\exists j<|w|,$ s.t. $w^{0..j}L\ r$

(7) $w\models r\Leftrightarrow\forall j<|w|,$ s.t. $w^{0..j}T^{\omega}\models r!$

(8) $w\models X!\ \varphi\Leftrightarrow|w|>1$ 且 $w^{1..}\models\varphi$

(9) $w\models[\varphi\cup\psi]\Leftrightarrow\exists k<|w|,$ s.t. $w^{k..}\models\psi,$ 且 $\forall j<k,w^{j..}\models\varphi$

(10) $w\models\varphi$ abort $b\Leftrightarrow w\models\varphi$ 或 $\exists j<|w|$ s.t. $w^jL\ b$ 且 $w^{0..j-1}T^{\omega}\models\varphi$

(11) $w\models r|\rightarrow\varphi\Leftrightarrow\forall j<|w|,$ s.t. $w^{0..j}L\ r,w^{j..}\models\varphi$

3)PSL 的分支时序逻辑 OBE

OBE 就是 CTL,采用文献[3]所使用的标准语义。

2　PSL 验证技术的研究现状

PSL 的验证技术大致分为基于断言的验证技术(ABV)、形式化验证技术以及运行时验证技术。PSL 擅长于硬件验证,各种使用 PSL 的能实现硬件动态验证或能实现形式化验证的工具可从 IBM、Cadence、Mentor Graphics 等公司获得;PSL 同样擅长于软件验证,IBM 实现了利用 PSL 和 Sugar 进行软件的模型检验工作[4~6],这些工作还可在基于 C++ 的模拟环境中实现。文献[7]介绍了 PSL 应用于软件的运行时验证过程中的方法。以下给出 PSL 的验证及相关技术的国内外研究现状。

2.1　基于断言的验证技术

断言是对一将要成立的属性的声明,以及要求验证工具验证此属性成立的指示。断言应用于系统设计的一系列任务之中,包括建模、验证和测试过程。基于断言的验证(assertion based verification, ABV)是一种用断言指导设计的验证技术,断言提供的可视化效果和白盒测试能添加到设计的内部状态之中。ABV 分为动态 ABV 和静态 ABV 两种。动态 ABV 是指基于模拟和仿真的 ABV,而静态 ABV 是指基于形式化验证技术(如模型检验)的 ABV。使用 PSL 的基于断言的验证正改变着传统的设计处理过程,因为这种方法学有助于形式化地定义各个抽象层次中的设计需求的特征、引导验证任务、简化测试平台的设计。由于更多的任务使用标准化语言 PSL 描述代替了传统的用户自定义模型,验证任务能被更多的工具识别与执行。因此使用 PSL 的基于断言的验证作为一种主要的功能验证方法最近越来越受到业界的青睐。

2.1.1　实现模拟的断言验证技术

模拟能较早地发现协议或设计中的错误。使用 PSL 断言实现对接口验证和需求规则的编码极大地简化了模拟的实现过程,包括对设计、重启、时钟模块的初始化和借助待执行任务(如写、读)定义测试场景以及将事务转换为低级信号的处理等。使用 PSL 的断言验证技术可直接应用于模拟过程之中。文献[8]提出了一种基于模拟的 PSL 的简单子集 sPSL 的时序断言验证引擎 Temporal Wizard,它主要将 sPSL 转换成一组支持 Verilog 的用户定义的系统任务和函数(USTF),并且与所有支持 PLI 的 Verilog 模拟器兼容。Temporal Wizard 由于引入了能高效处理 forall 操作子的"标签"概念,与多数基于时序检验器的状态机比较,其检测能力有了较大的提高,时空复杂度仅为 $O(n)$。高层决策图(HDLL)是一种广义的 BDD(二进制决策图)结构,文献[9]提出在使用 PSL 断言验证技术的基础上对 HDLL 模型进行扩充形成 THDLL 表示模型应用于模拟,以简化故障诊断和高效执行调试的方法。由于 THDLL 具有图形遍历的快速评估和因果识别机制,该方法具有高效实现模拟和低内存消耗的优势。文献[10]主要研究 PSL 断言验证技术可视化的关键方法以捕捉用户约束,获得关注断言,确保模拟过程中发掘断言允许的所有可能行为,提出了基于 Büchi 自动机和分离范式(separate normal form)的发现模拟路径的方法。分离范式的优势在于其高效性,而 Büchi 自动机方法是覆盖率技术的重要补充。文献还研究了结合两种方法的应用情况。

2.1.2　实现仿真的断言验证技术

在仿真环境中,断言扮演着监控器的角色,检查预期的行为是否满足,当设计的行为和预期的不一致时报告失败。文献[11]提出了一种从 PSL 断言中合成具有 RTL 序列电路形式的属性监控器的方法,设计与监控器通过监控变量相连,且用 PVS 定理证明器验证了该方法的正确性。文献[11]构造监控器的方法适用于 PSL 的 FL 的强、弱操作子,其不足在于监控器只接受能使用"on-the-fly"方法评估的属性。文献[12]提出从 PSL 断言中自动产生异步、可综合监控器的方法以在线检验正常操作方式下的设计,此方法可应用于监控由同步 IP 构造的大型系统以及监控异步事件,保证在事件延迟情况下得到

正确的响应。PSL 公式合成监控器的方法与文献[11]类似,区别在于文献[12]的方法需要采用准延迟非敏感方式(quasi-delay insensitive,QDI)设计异步库,这增强了设计的健壮性、可度量性和可重用性;而且 QDI 方式与标准 FPGAs 兼容,因此 FPGA 原型系统的调试也是文献[12]方法的一个重要应用领域。

文献[13]提出了一种支持 ABV 的硬件仿真架构,开发了实现硬件仿真的检验产生器 MBAC。将断言嵌入到硬件仿真器的实现过程包括将 PSL 声明转换为适于验证下设计(DUV)的硬件描述语言的过程,PSL 文件的每个验证单元(vunit)对应于一个仿真硬件模块(称为断言—电路模块),MBAC 为每个模块产生相应的文件。每个 PSL 表达式都使用两种基本模式解释,即 must 和 if 模式,最后分别利用这两种模式的显式序列算法将 PSL 断言转换为完整的仿真电路。上述方法提高了实现 PSL 断言仿真过程的综合性能,并且简化了断言嵌入到仿真硬件结构的实现过程。

文献[14]是在用 PERL 编写产生的随机测试机器码的基础上,利用 PSL,结合 Cadence 的 ABV 验证工具 IncisiveTM,实现了对具有 MIPS 架构的 RISC 模块进行基于仿真的形式验证,该方法保证验证功能覆盖率达 100%。文献[15]利用基于 PSL 断言的形式与仿真相结合的验证方法验证了宽带电路交换芯片的设计。实验表明,断言的引入降低了验证工作的复杂度,提高了验证的效率,确保了验证的质量。

2.1.3　与 ABV 相关的技术——属性的上下文绑定技术

执行基于断言的验证的最大挑战在于如何将设计的行为检验属性正确地绑定到上下文之中。PSL 具有绑定功能,具体是将验证单元(vunit)绑定到与其相关的设计单元之中。静态验证工具 RuleBase PE 已经实现了通过 PSL 的验证单元将属性绑定到上下文的一种方法。文献[16]提出另一种通过描写内嵌式断言(指在设计执行代码即 RTL 代码中出现的断言)实现 RuleBase 绑定的执行过程。IBM 开发的 FoCs 同样支持上述绑定。文献[17]进一步扩展了 FoCs 的绑定能力,通过内嵌式断言产生模拟监控器以捕捉显式描述的设计属性并将其与正确的上下文相连。内嵌式断言为开发过程中的静态验证断言提供了一种简便描述方法,而且它克服了白盒测试中描写白盒断言需要对验证下设计(design under verification)具有深刻的理解的问题。

2.2　形式化验证

形式化验证方法是对模拟和仿真方法的重要补充,它使用严格的数学推理来证明系统是否满足全部或部分规约(系统属性),主要包括模型检验和定理证明技术。

2.2.1　PSL 与定理证明

定理证明是在验证者的引导下,以形式逻辑的公理和推理规则为基础,不断对公理和已证明的定理施加推理规则,产生新的定理,直至推导出需要的定理的过程。文献[18]介绍了已开发的使用设计断言的半自动定理证明的验证系统 PROVERIFIC,构建了一种统一建模框架 UFM 将命题逻辑形式化地对应设计及其断言,定义了能捕捉 PSL 断言语义和Verilog表示的 UFM 语法子集的一般谓词模板。在验证过程中,PROVERIFIC 使用这些模板自动地提取一个设计(Verilog 表示)的形式化模型以及其属性(PSL 表示),并将它们转换成 PVS 定理证明器的更高阶逻辑,PVS 能进一步验证属性的正确性。

高阶逻辑(higher order logic)是在一阶命题演算逻辑基础上扩展形成的,已开发的 HOL 定理证明器支持高阶逻辑的定理证明。文献[19]详细给出了与 PSL 每个操作算子相关的 HOL 系统的表示形式,即给出了如何将 PSL 嵌入 HOL 系统的过程。PSL 嵌入 HOL 的目的在于:a)调试和证明元定理;b)获得机器可处理的语义,高阶逻辑广泛地应用在各种形式化工具中,一旦具有高阶逻辑的表示形式就能容易地转换为其他工具可执行的表达;c)结合了定理证明和检验功能,PSL 和高阶逻辑结合后具有更强的表达能力,此时的定理证明器可以实现执行、检验和定理证明的功能。在此基础上,文献[20]说明了 HOL 定理证明器是验证 PSL 语义合法性的强有力工具。文献[21]研究了如何利用 HOL 定理证明器元语言 ML 编码证明脚本执行 PSL 形式化语义的方法。HOL 系统提供了一个 ML 函数 Eval 用于证明定理$L\ t=t'$。其中 t'是项 t 的评估结果,Eval 可以调用显示地添加在上下文中的等式和决策断言。若考虑执行带时钟的 PSL 形式语义则首先需要执行时钟删除重写规则,而后再用 Eval 函数评估非时钟 PSL 语义。

2.2.2　PSL 的模型检验实现

模型检验是通过遍历系统所有状态空间,对有穷状态系统进行自动验证,确定系统模型是否具有某种性质,并自动构造不满足验证性质的反例的技术。Tuerk 等人[22]给出了利用 HOL 的定理证明器和模型检验器 SMV 实现 PSL 的模型检验的算法,利用改进的正确性转换定理构筑建立在标准的 PSL 形式化语义基础上的 PSL 执行结构。应用方面,此执行结构借助 IBM 的 RuleBase CTL 模型检验器,实现了一种能检验 PSL 执行正确性的工具,可以检测执行时钟 abort 操作子下的 bug。文献[22]方法的优势在于可以处理时钟 PSL 公式,前提是将时钟视为"语法糖衣"并进一步使用重写规则删除时钟操作子;不足在于只适用于 PSL 的子集,特别是还未扩展到 PSL 的正规表达式部分,因此只能处理一类特殊的 PSL 问题。

Pnueli 等人[23]借助公平离散系统(JDS)构建测试器实现了 PSL 的模型检验。用 JDS 分别构建系统计算模型 D 和待验证属性 φ 的测试器 T_φ。构造 T_φ 的计算复杂度为 $O(2^n)$,n 为 PSL 的公式长度;当 SERE 限定在三种操作子:连接(;)、交叠连接(:)和闭包([*])集合时,T_φ 的变量个数与 φ 长度呈线性关系。由于测试器能用符号表达,可以方便地应用在符号化模型检验中。模型检验过程采用传统的自动机理论方法:a)对 D 和 T_φ 执行同步并发组合计算 $D\|T_\varphi$,即结果的初始状态和结束条件为相应断言的合取形式,结果的迁移关系为系统 D 和测试器 T_φ 迁移关系的合取式;b)检验 $D\|T_\varphi$ 是否有公平计算路径,若存在,则 D 不满足属性 φ。此方法构造的测试器不同于自动机,具有高度的组合性,这对充分利用 PSL 的上下文起到了独到的作用。

文献[24]提出了 PSL 的有界模型检验方法及其算法框架。首先,定义 PSL 逻辑的有界语义,而后将有界语义进一步简化为 SAT(可满足性问题),分别将 PSL 性质规约公式 α、系统 M 的状态迁移关系转换为命题公式$[[\alpha]]_{M_k}$和$[[M,\alpha]]^s_k$,最后验证命题公式$[[M,\alpha]]^s_k \wedge [[\alpha]]_{M_k}$的可满足性,这样就将时序逻辑 PSL 的存在模型检验转换为一个命题公式的可满足性问题,并用一个队列控制电路实例具体解释算法执行过程。

2.2.3　软件的模型检验

随着硬件形式化验证技术的不断成熟,开发软件的形式化验证技术也提上了议事日程。RuleBase 是由 IBM Haifa 研究实验室开发的一种应用 PSL 实现形式化验证的工具。IBM Haifa 研究实验室在 RuleBase 的基础上开发了一种用于跟踪并发 C 程序 bug 的模型检验器 Wolf[4],Wolf 能自动产生模型和规约,在采用 BDD(二进制决策图)的符号化方法基础上显式地完成软件模型检验。Wolf 主要包括三个组成部分:C-to-model 转换器、集成于 RuleBasePE 的软件验证算法以及用于显示 bug 路径的 GUI-vet。其中,C-to-model 转换器接收并发 C 程序并将其转换为有穷模型,产生“hints”和“guides”以引导软件模型检验器发现 bug。软件模行检验器主要采用偏序析取分割的符号化 BDD 算法实现软件验证以及采用动态 BDD 重排序算法缩减 BDD 重排序时间,并且采用偏序精简技术枝剪遍历的分支。Wolf 能处理指针的删除和资源的动态分配等工作,但其不足是很难处理由非确定输入带来的非确定行为。

文献[5]在借助 c2edl 工具的基础上将 ANSI-C 代码转换为 RuleBase 的输入语言 EDL,提出使用 RuleBase 实现 C 描述程序的碎片收集机制的模型检验方法。RuleBase 的核心 SMV 使用 BDD 作为其基本数据结构。若一个将来使用的 BDD 收集在碎片堆中,则结果是一个悬摆参考,也即此 BDD 潜在地包含在碎片集中;若一个将来不使用的 BDD 仍不删除,则结果导致内存泄漏。论文分别应用表达悬摆参考和内存泄漏的 PSL 公式实现 SMV 碎片收集机制的模型检验。文献[5]的软件模型检验方法具有的特点是:a)通过自动产生抽象模型的方式,将模型检验方法直接应用于源代码;b)c2edl 的使用无须人工干预,即实现是完全自动的且为 RuleBase 创建了有穷状态模型;c)只检验时序逻辑表达的属性,对时序逻辑的嵌套层次给予了人工限制。

文献[6]使用 RuleBase 模型检验器建模和形式化验证一个软件高速缓存器的算法,其建模过程采用由 C 代码直接产生的超详细度模型,从而取代先前使用高级抽象模型的方式。超详细模型建立了真实软件结构与实现代码之间的一一对应关系。与文献[5]的抽象模型比较,其优势在于详细建模算法比抽象级建模算法更加简便;其次它排除了建立抽象模型遇到的一些问题,如虚假否定、虚假肯定等。最后,一些研究如文献[25]均实现了超详细模型的成功应用,这都可以作为上述软件模型检验的借鉴方法。

2.2.4　与形式化验证相关的技术

1)PSL 构造自动机技术

自动机是许多形式化规约、验证方法的基础模型,是一种可用于模型检验等验证技术的验证形式,在验证技术领域中有着广泛的应用。Bustan 等人[26]将 PSL 的核心逻辑定义为LTL_WR,即在 LTL 的基础上再定义正规表达式 REs,并且证明每个 LTL_WR 公式 f 都存在一个非确定的 Büchi 自动机(NBA),复杂度是 f 的双指数。在表达时序逻辑方面,交换自动机(AFA)比非确定自动机(NFA)复杂度呈指数形式精简。因此为构造 NFA,Bustan 等人先构造 LTL_WR 的 AFA。但是,由于交换自动机只能产生每个分支都可接受的运行,而事实上,运行遍历的状态既包括可接受也包括不可接受。基于这种情况,文献[26]进一步将 AFA 转换为 NFA。文献[27]介绍了将 PSL 转换为非确定 Büchi 自动机的传统方法:将 PSL 中的 SEREs 首先转换为最小 NFA,而后 NFA 组织在一起,将 PSL 公式转变为交换 Büchi 自动机 ABA,最后使用 Miyano-Hayashi(MH)方法[28]将 ABA 转换为 NBA。文献[29]指出这种方法就是基于 SAT 的有界模型检验,前提是弱交换自动机;文献[30]提出一种基于 MH,由 PSL 属性构造 ABA 而后转换为对应的 NBA 的符号化编码方法。文献[29,30]提出的两种方法都试图限制编码大小,但它们都依赖一个旨在最小化 ABA 并执行优化方法的库,ABA 最小化代价高,即便是对于中等长度的 PSL 规约,这两种方法都不能在可接收时间内完成转换任务。文献[31]提出将 PSL 转换为符号化表示的 NBA 的直接编码,主要基于 SONF(suffix operator normal form,后缀操作符范式),实验表明 SONF 结构是一种快速构造 NBA 的有效方法。文献[32]给出了将 PSL 的一个包含安全属性的子集 safetyPSL^{det} 直接转换为 co-universal 自动机的方法。由于有穷自动机能发现违反安全属性的反例,此时的模型检验可简化成不变式“自动机 A 不处于接收状态”的验证过程。不变式检验不但简单,而且许多模拟工具在不变式中会执行得更好;缺点是虽然 safetyPSL^{det} 的大部分操作子都能建立相应的非确定 CUA,但弱正规表达式 r 建立的自动机必须先确定化,这样明显地增加了构建自动机的复杂性。

2)模型检验的简化设计技术

模型检验为验证逻辑设计是否满足指定属性提供了一种有效的途径。模型检验器穷尽地检验所有可能的输入序列,或穷尽遍历和搜索所有的可达设计状态。为更有效地执行上述过程,人们常借助一些简化手段将设计转换为逻辑上等价的另一简易设计,这种简化后的设计更方便穷尽开发。文献[33]提出两种简化方法,即重定时简化和量词简化,并讨论了各种实现问题,检验了这两种方法的有效性。实验证明,量词简化方法在缩减模型检验时间的情况下显示了突出优势,但其自身的执行过程却相当费时;而重定时简化方法只针对部分问题有效。

3)综合技术

综合是指从指定规约中自动产生设计,同时形式化地验证规约的可实现性并获取实现证据的处理技术。文献[34]描述了基于 PSL 属性的综合技术,在给定 PSL 线性时间片断表达的规约的前提下,此技术自动综合生成硬件设计,主要包括三种不同的方法:a)非确定方法,是一种基于 PSL 模型检验时采用的自动机的方法,此方法只能综合 PSL 的一个子集,但是综合效率很高;不足在于适用面较窄,最适用于部分执行实现系统的综合问题。在合理的消耗下,部分执行实现系统能有效地修补失效系统。b)广义反应规约综合方法,适用于综合更丰富类型的规约,特别是能更精确地综合在使用广义反应接受条件下表达的属性规约。其思想是:将规约转换成能应用三层嵌套的不动点算法的特殊形式,实验证明,在综合大规模规约集合时,其实现复杂度也是可接受的。c)完全方法,应用于采用 PSL 完整线性时间片断的情形,其实现复杂度为双指数时间。上述三种综合方法优势互补,它们能以不同的计算复杂度综合 PSL 不同子集表达的属性规约。PSL 片断涵盖越全面则综合的代价越高,因此在采用综合方法时特别要考虑表达能力与代价之间的折中问题。

电路设计的综合处理方法不需要手写代码,使从规约中快速构造原型系统成为可能。文献[35]应用自动高层综合方法

直接产生 PSL 描述的规约的结构修正(correct by construction)门级实现。首先对 PSL 属性执行综合处理,此时将 PSL 公式的实现问题简化为介于系统与环境之间的 2 选手无穷博弈问题;而后构造系统变量和环境变量的 BDD 结构;最后给出了将 BDD 转换为硬件电路的算法。该方法的优势在于可以获得设计 PSL 规约的一致性和完整性,并且自动综合方法最适用于产生控制电路,未来可考虑将手动编码的数据路径与自动综合产生的控制电路结合的问题;不足在于产生的门级输出较为复杂且不易手动修正,有些电路的规约不能直接描述,并且常常伴随着一个递归处理的过程。文献[36]提出了基于自动机的 PSL 属性断言验证器的综合方法。

2.3　软件的运行时验证

最近几年,嵌入式系统的设计复杂度越来越高,随之带来的验证问题也越困难,在此基础上,人们寻求软件在更高抽象层次上验证规约的方法。文献[7]给出一种将 PSL 绑定到 C 语言上的过程,实现了一个对程序 PSL 逻辑属性执行运行时验证的工具。具体算法如下:

1)数据模型的建立

输入:sPSL 源文件、C 源文件以及 C 源文件的调试信息。

输出:数据模型至评估引擎。

a)用一个脚本处理 sPSL 源文件,为每个属性创建一个标记条目;

b)为 C 源文件的每个变量和函数建立标签;

c)编译 C 源文件,产生详细的调试信息;

d)通过编译工具如 OBJDUMP 将 c)的调试信息抽取到一个文本文件中。

2)数据信息的评估

输入:步骤 1)输出的数据模型以及地址信息、寄存器分配、存储器容量信息。

输出:true 或 false 或 undefined。

a)实时系统通知 Giano 模拟器有关 C 源程序的名字及地址信息;

b)评估引擎依据程序名搜索相应的数据模型文件,作语法分析且创建相应的 PTree;

c)若一个指定属性满足活性(liveness),则评估引擎创建并初始化属性的一棵评估树 ETree(evaluation tree),监控功能开始启动。

其中步骤 2)中 c)的 evaluation tree 通过深度优先和左子树优先的方式实现遍历,每个分支至多带两片叶子,叶子可以是操作算子或变量数值。评估的结果为一个三值逻辑,即 true(T),false(F)和 undefined(Z)。例如评估一个属性 always(a = 1 next b = 1)→until(c = 1),其对应的评估树如图 3 所示。节点 a = 1 是逻辑树的首节点,在下一事件未能判断操作算子 next 是否满足时,逻辑树返回 Z;当 b = 1 满足时,next 操作子返回 T,否则返回 F;一旦 until 操作算子接收到左子数返回的 T 值时,它就开始监控右子树表示的释放节点(release point),即 c = 1;在 c = 1 未满足时,逻辑树返回 Z,一旦 c = 1 和 a = 1 next b = 1 同时满足时,until 操作子向父节点 always 返回 T。此方法的优点是:a)在验证过程中无须改动执行程序和代码,增强了验证处理的可信度;b)原型系统结果产生功能调用和变量变化的执行路径,这对程序员理解程序错误行为的发生原因是大有裨益的;c)工具同时支持实时规约,可应用于性能验证之中;d)sPSL 语言和评估引擎不依赖于使用的特定语言,它们可应用于堆栈—寄存器结果执行的任何块结构语言中。不足是当前的原型系统不支持 SEREs。

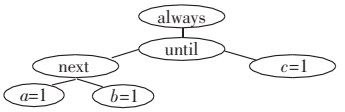

图 3　属性 always(a=1 next b=1)–>until(c=1)的评估树

3　结束语

PSL 是由 EDA 界的标准化组织 Accellera 提出,并于 2005 年确定为 IEEE 工业标准(IEEE-1850)的用于描述并发系统的属性规约语言。本文在简要介绍 PSL 的语法、语义和分层结构基础上,综述了 PSL 的验证技术的研究现状及各种技术、方法的优缺点,今后有待进一步研究和改进的工作包括:

a)基于断言的验证技术。PSL 的 ABV 技术把传统的设计处理从具有粗糙的功能规约文档的非形式化 RTL 编码方式推向了更高层面的处理过程:对设计决策、设计属性和设计假设进行功能规约描述;在模拟过程中简化功能覆盖率分析以检验边角情况;简化用于验证结果正确性的测试平台参考模型的设计等。(a)为方便执行基于 ABV 的模拟技术,以往的文献常采用 PSL 的简单子集 sPSL 描述待验证的断言的方法,这势必影响了对复杂设计行为的表达能力,今后考虑将正规表达式等逻辑描述扩充到 ABV 技术中,不断拓展 ABV 技术的应用范围;(b)将设计故障诊断技术和调试技术集成到 ABV 技术中;(c)对 ABV 的 PSL 断言施加一些重写规则,使用定理证明器如 HOL 或 PVS 形式化地证明规则的正确性;(d)考虑将动态验证与形式化验证结合的混合验证方法实现 ABV 技术,也是今后可以尝试研究的方向。

b)软件的模型检验技术。应用 PSL 的软件模型检验技术的研究目前尚处于研究的初级阶段,已有的一些文献[4~6]都只是实现解决针对特定问题的 PSL 的软件模型检验方法,基本上都借助了 IBM 的 RuleBase 工具。今后需要考虑拓展的内容包括:(a)重点研究如何将 PSL 的软件模型检验技术应用于更一般的实际问题,改进工具的算法,使其对更一般的问题提供技术支持;(b)软件的模型检验的一个重要问题是内存容量,如何应用如无限状态技术[37]解决限定堆栈长度的问题将是一个亟待研究的方向;(c)“影响锥”简化方法是一种针对硬件设计的有效简化方法,针对软件设计的特点,需要开发一种专门适用于软件的简化技术,其中需要结合程序切片和静态分析技术;(d)实现对复杂数据类型如结构、指针和联合的自动编码技术;(e)研究直接支持软件的路径产生技术,RuleBase 的图形化路径显示技术源于硬件设计,对于软件,提供给用户一个类似于图形化调式器的窗口更有意义,以支持单步前进和后退,同时自动产生 readme 文件帮助实现路径的自动解释。

c)PSL 构造自动机。符号化模型验证技术在硬件设计中显示了极其高效的验证能力,为进行模型检验,PSL 公式需要转换为一种可验证形式,通常是自动机。从目前已有的文献[26,27,31,32],PSL 的模型检验的研究成果主要是采用自动机理论,通过构造 PSL 的自动机来实现自动验证。今后的研究主要关注于:(a)完成 PSL 自动机的构造后,通常会采用符

号化模型检验算法,因此可以充分利用自动机结构信息定义一种较好的 BDD 变量排序形式,以提高模型检验算法的效率;(b)考虑将活性简化为安全性检验的应用技术,以产生更短的反例路径,特别是在实现公平环路检测时将显示应用价值;(c)使用一些简化技术如双向模拟最小化技术以产生等价的规模更小的自动机,缩小搜索空间。

d)程序的运行时验证技术。基于程序的时序逻辑 PSL 的安全关键性质运行时,验证需要依据通过监控器收集的程序整个运行过程的信息,以此来验证是否满足时序性质的要求,但可能占用资源多。基于此原因,拟采取以下措施:(a)考虑研究如何把 PSL 时序逻辑公式表示的性质分解为与其等价的一组断言,插装到程序的适当位置,根据断言的违反情况即可判断出性质的满足情况;(b)运行时验证既要考虑可终止的程序也要考虑类似反应式系统这样的非终止程序,此时在任何时刻只能获得当前执行的一个有穷状态前缀,需要对时序逻辑的语义进行适当的修改,变为针对有穷路径,同时与基于无穷路径的时序逻辑公式语义相一致;(c)根据修改后的语义生成接收有穷序列的自动机(验证器),该自动机接收当前执行的有穷前缀,输出相应的验证结果,使得在存在软件缺陷的情况下尽可能早地检测到该缺陷。在基于当前信息能够验证给定性质规约时,及早终止对程序的进一步检测。

e)PSL 的模型检验和可满足性问题的计算复杂度。PSL 的重要应用是对并行系统进行形式化验证,模型检验是其中的技术之一;同时,理论上也已证明许多难解的问题都可从多项式变换为可满足性问题。可满足性问题是研究时序逻辑的核心问题之一,并已成为程序验证的一种有力工具。基于这两方面的考虑,研究时序逻辑 PSL 的模型检验和可满足性问题的计算复杂度将是今后 PSL 理论和实际应用研究的方向,这为解决相关问题提供了难度的标准,且对正确评价解决该类问题的各种算法的效率、进而确定对已有算法的改进余地具有重要的指导意义。文献[38]研究的对象局限于 PSL 线性时间逻辑各片断的计算复杂度,这不适应实际复杂应用问题属性的逻辑表示,今后需要进一步研究 PSL 所有操作符表达的逻辑公式的上述两类问题的复杂度,特别是需要研究如何将这些复杂度理论真正引导到解决实际问题的过程中,这是引导 PSL 研究从理论转向实践的一条途径。

参考文献:

[1] Accellea. Property specification language reference manual version 1.1 [EB/OL]. (2004-06-09) [2008-03-02]. http://www.eda.org/vfv/docs/PSL-v1.1.pdf.

[2] WOLFSTHAL Y. Eu-sponsored deployment of PSL[EB/OL]. (2004-02-04)[2008-05]. http://www.pslsugar.org/psl_meeting.html.

[3] CLARKE E M, GRUMBER O, PELED D A. Model checking[M]. Cambridge: MIT Press, 2000.

[4] BARNER S, GLAZBERG Z, RABINOVITZ I. Wolf-bug hunter for concurrent software using formal methods[EB/OL]. [2009-05-11]. http://www.haifa.ibm.com/projects/verification/RB_Homepage/ps/wolf.pdf.

[5] EISNER C. Model checking the garbage collection mechnism of SMV [EB/OL]. [2009-04-23]. http://www.research.ibm.com/people/e/eisner/papers/sw.ps.

[6] EISNER C. Formal verification of software source code through semi-automatic modeling [EB/OL]. [2009-06-13]. http://www.haifa.ibm.com/dept/svt/papers/swcache.pdf.

[7] CHEUNG P H, FORIN A. A C-language binding for PSL[C]//Proc of the 3rd International Conference on Embedded Software and Systems. Berlin: Springer-Verlag, 2007: 584-591.

[8] CHANG Kai-hui, TU Wei-ting, YEH Yi-jong, *et al*. A simulation-based temporal assertion checkers for PSL[C]//Proc of the 3rd IEEE International Symposium on Micro-Nano Mechatronics and Human Science. 2003: 1528-1531.

[9] JENIHHIN M, RAIK J, CHEPUROV J, *et al*. Temporally extended high-level decision diagrams for PSL assertion simulation[C]//Proc of the 13th European Test Symposium. Washington DC: IEEE Computer Society, 2008: 61-68.

[10] PILL I, JOBSTMANN B, BLOEM R, *et al*. Property simulation, PROSYD D1.2-1[R/OL]. (2005-05-01)[2009-03-22]. http://www.prosyd.org/twiki/pub/Public/DeliverablePageWP1/Prosyd1.2_1.pdf.

[11] MORINALLORY K, BORRIONE D. Proven correct monitors from PSL specification[C]//Proc of Conference on Design, Automation and Test in Europe. 2006: 1246-1251.

[12] MORINALLORY K, FESQUET L, BORRIONE D. Asynchronous assertion monitors for multi-clock domain system verification[C]//Proc of the 17th IEEE International Workshop on Rapid System Prototyping. Washington DC: IEEE Computer Society, 2006: 98-102.

[13] BOULE M, MZILIC Z. Incorporating efficient assertion checkers into hardware emulation[C]//Proc of the 23rd International Conference on Computer Design: VLSI in Computers and Processors. 2005: 221-228.

[14] 许伟坚,周剑扬,吴伟贤,等. 基于 PSL/Sugar 语言的 RISC 模块验证[J]. 厦门大学学报:自然科学版, 2005, 44(3): 337-340.

[15] 张华,郭建,韩俊刚. 基于 PSL 断言的宽带电路交换芯片验证[J]. 计算机工程, 2007, 33(14): 216-235.

[16] NEVO Z, SHAMIS M, PIDAN D. Support for embedded PSL in verilog flavour-static checking, PROSYD D3.2/15[R/OL]. (2006-03-31)[2009-04-13]. http://www.prosyd.org/twiki/pub/Public/DeliverablePageWP3/Prosyd3.2_15.pdf.

[17] PIDAN D, RABINOVICH S. Support for embedded PSL in verilog flavour- oberservers, PROSYD D3.2/16[R/OL]. (2006-06-30)[2009-05-15]. http://www.prosyd.org/twiki/pub/Public/DeliverablePageWP3/Prosyd3.2_16.pdf.

[18] SULE P, KIM Y, MANSOURI N. PROVERIFIC: experiments in embedding(PSL) standard assertions in theroem-proving-based verification[C]//Proc of the 48th IEEE International Midwest Symposium on Circuits and Systems. 2005: 112-115.

[19] GORDON M. PSL semantics in higher logic order logic[EB/OL]. [2009-06-18]. http//: www.cl.cam.ac.uk/~mjcg/Talks/DCC04/paper.ps.

[20] GORDON M. Validating the PSL/Sugar semantics using automated reasoning[EB/OL]. (2003)[2009-02-11]. http://www.cl.cam.ac.uk/~mjcg/Sugar/facpaper/paper.ps.

[21] GORDON M, HURD J, SLIND K. Executing the formal semantics of the accellera property specification language by mechanised theorem proving[M]//Correct Hardware Design and Verification Methods. Berlin: Springer-Verlag, 2003: 200-215.

[22] TUERK T, SCHNEIDER K, GORDON M. Model checking PSL using HOL and SMV [EB/OL]. (2007-05-11) [2009-01-19]. http://www.cl.cam.ac.uk/~tt291/publications/TuSG07.pdf.

[23] PNUELI A, ZAKS A. PSL model checking and run-time verification via testers[C]//Proc of the 14th International Symposium on Formal Methods. Berlin: Springer-Verlag, 2006: 573-586.

[24] 虞蕾,赵宗涛. PSL 的有界模型检验[J]. 电子学报,2009,37(3): 614-621.

[25] EISNER C, SHITSEVALOV I, HOOVER R, *et al*. A methodology for formal design of hardware control with application to cache coherence protocols [C]//Proc of the 37th Annual Design Automation Conference. New York: ACM Press, 2000: 724-729.

[26] BUSTAN D, FISMAN D, HAVLICEK J. Automata construction for PSL [EB/OL]. (2005-05-10) [2009-01]. http://www.wisdom.weizmann.ac.il/~dana/publicat/automata_constructionTR.pdf.

[27] DAVID S B, BLOEM R, FISMAN D, *et al*. Automata construction algorithms optimized for PSL[EB/OL]. (2005-07-10) [2009-03]. http://www.prosyd.org/twiki/pub/Public/DeliverablePageWP3/Deliverable3.2_4.pdf.

[28] MIYANO S, HAYASHI T. Alternating finite automata on omega-words [J]. Theoretical Computer Science, 1984, 32(3): 321-330.

[29] HELJANKO K, JUNTTILA T, KEINÄNEN M, *et al*. Bounded model checking for weak alternating Büchi automata[C]//Proc of the 18th International Conference on Computer Aided Verification. Berlin: Springer-Verlag, 2006: 95-108.

[30] BLOEM R, CIMATTI A, PILL I, *et al*. Symbolic implementation of alternating automata[C]//Proc of the 11th Internationall Conference on Implementation and Application of Automata. Berlin: Springer-Verlag. 2006: 208-218.

[31] CIMATTI A, ROVERI M, SEMPRINI S, *et al*. From PSL to NBA: a modular symbolic encoding[C]//Proc of Formal Methods in Computer. Berlin: Aided Design, 2006: 125-133.

[32] RUAH S, FISMAN D, DAVID S B. Automata construction for on-the-fly model checking PSL safety simpleSubset[R]. Haifa: IBM Haifa Research Lab, 2005.

[33] TZOREF R, BRINKMANN R, NEVO Z. Research report on improved symbolic search strategies and model reduction for static property checking, PROSYD D3.2/2[R/OL]. (2005-03-15)[2009-03]. http://www.prosyd.org/twiki/pub/Public/Public Deliverableold/Prosyd 3.2_2publicversion.pdf.

[34] BLOEM R, JOBSTMANN B, PNUELI A. Property-based logic synthesis for rapid design prototyping, PROSYD D2.2/1[R/OL]. (2005-09-01)[2009-03]. http://www.prosyd.org/twiki/pub/Public/Public Deliverableold/Prosyd2.2_1.pdf.

[35] BLOEM R, GALLER S, JOBSTMANN B, *et al*. Specify, compile, run: hardware from PSL[J]. Electronic Notes in Theoretical Computer Science, 2007, 190(4): 3-16.

[36] BOULE M, ZILIC Z. Efficient automata-based assertion-checker synthesis of PSL properties[J]. ACM Trans on Design Automation of Electronic Systems, 2008, 13(4): 4-1-4-21.

[37] ESPARZA J, HANSEL D, ROSSMANNITH P, *et al*. Efficient algorithms for model checking pushdown systems[C]//Proc of the 12th International Conference on Computer Aided Verification. Berlin: Springer-Verlag, 2000: 299-310.

[38] LANGE M. Linear time logics around PSL: complexity, expressiveness, and a little bit of succinctness[C]//Proc of the 18th International Conference on Concurrency Theory. Berlin: Springer-Verlag, 2007: 90-104.

(上接第 2413 页)

c)会话创建和会话。在会话的创建过程中,调用 getLifeTimeSequenceBase()服务,防止再次可能的重放攻击;同时设置安全会话密钥,对会话内容进行调用,对服务的调用消息进行加密、解密和执行。中间过程只能得到加密过的消息,没有密码无法解密。在安全互连描述中,对频繁进行请求并认证失败的设备不予响应,可以有效防止拒绝服务的攻击。

d)结束工作。调用 expiredSessionKeys()函数删除原有的密钥等一些结尾工作,使得密钥过期,不能够在新的会话中再次使用原来的密钥,进一步增强了密钥的安全性。

5 结束语

在前人关注 IGRS 和 UPnP 设备简单非安全互连的基础上,本文依照 IGRS 和 UPnP 的安全工作原理,参考开源的 UPnP 库对 UPnP 进行改进,实现 IGRS 与 UPnP 的安全互连。原来非安全的互连上存在的设备冒充、重放攻击,互联网络中的设备、用户受到网络干扰等问题,在此方案中得到了十分有效的解决。在泛在设备的互连技术项目中,在已经实现了 IGRS 与 UPnP 互通协议栈的基础上进行了安全扩展,实现了预期的认证、完整性、防止重放、访问控制和保密五大目标。

本系统也存在需要改进的地方,如公钥的维护和更新问题;再者,由于 IGRS 和 UPnP 都存在组播特性,网络上的其他计算机可以"听到",并发现网络中无安全特性的设备,如果这个设备被其他安全设备所信任,那么黑客有可能利用此设备作为跳板来实现对其他设备和服务的攻击。

参考文献:

[1] 闪联工作组. 闪联应用白皮书[R]. 北京:闪联工作组,2003.

[2] UPnP Forum. UPnP device architecture 1.0[EB/OL]. (2008-10-15) [2009-11-10]. http://www.UPnP.org/resources/documents.asp.

[3] UPnP Forum. Security console:1 service template[EB/OL]. (2003-11) [2009-11-10]. http://www.UPnP.org/resources/documents.asp.

[4] UPnP Forum. Device security:1 service template[EB/OL]. (2003-11) [2009-11-10]. http://www.UPnP.org/resources/documents.asp.

[5] 中华人民共和国信息产业部. SJ/T 11310-2005,信息设备资源共享协同服务第一部分:基础协议[S]. 北京:闪联工作组,2005.

[6] UPnP Forum. UPnP security ceremonies design document v1.0[EB/OL]. (2003-10)[2009-11-10]. http://www.UPnP.org/resources/documents.asp.

[7] 周雪凤. 数字家庭 UPnP 标准安全研究[J]. 科技情况开发与经济,2007,17(25):1-2.

[8] 廖国威,杨军,邓中亮. IGRS 基础协议中的安全机制[J]. 计算机安全,2006(3):1-3.

[9] 王佳慧,贺墚. UPnP 中的 DoS 攻击防御方案[J]. 计算机系统应用,2008(8):1-4.