软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(5):1144−1166 [doi: 10.13328/j.cnki.jos.005214]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种面向 CPS 的控制应用程序协同验证方法*

张 雨[1,2], 董云卫[3], 冯文龙[1,2], 黄梦醒[1,2]

[1](南海海洋资源利用国家重点实验室(海南大学),海南 海口 570228)

[2](海南大学 信息科学技术学院,海南 海口 570228)

[3](西北工业大学 计算机学院,陕西 西安 710129)

通讯作者: 黄梦醒, E-mail: huangmx09@163.com

摘 要: 信息-物理融合系统是一种新型嵌入式系统计算模式.它集成了控制计算过程和受控对象,二者相互影响并有机结合.随着信息技术在现实世界中更加广泛、深入的应用,智能化程度不断提升,在具有信息-物理紧密耦合特点的嵌入式系统中,嵌入式控制软件的功能比重急剧上升,作用更加突出.作为安全攸关的系统,需要引入形式化验证方法来保证嵌入式控制应用软件的安全性.基于自动机理论建立统一的系统验证模型,并针对系统的可达性、安全性(safety)和活性(liveness)等属性要求,提出了对该模型进行形式化验证的算法:基于有界模型检验方法,基于可达性将对系统模型的相关属性验证问题转换为可满足性判定问题.将活性转换为 Büchi 自动机,并基于四值语义进行判断.在求解过程中,通过偏序规约等手段化简了问题求解的规模,提高了可验证系统的规模.另外,结合协同仿真技术,灵活配置验证的场景,提高验证的可用性.实验结果表明,结合仿真,形式化协同验证方法可以有效地对系统进行验证.

关键词: 信息-物理融合系统;嵌入式控制应用程序;自动机理论;协同验证;有界模型检验

中图法分类号: TP311

中文引用格式: 张雨,董云卫,冯文龙,黄梦醒.一种面向 CPS 的控制应用程序协同验证方法.软件学报,2017,28(5):1144−1166. http://www.jos.org.cn/1000-9825/5214.htm

英文引用格式: Zhang Y, Dong YW, Feng WL, Huang MX. Co-Verification approach to control software program for CPS. Ruan Jian Xue Bao/Journal of Software, 2017,28(5):1144−1166 (in Chinese). http://www.jos.org.cn/1000-9825/5214.htm

Co-Verification Approach to Control Software Program for CPS

ZHANG Yu[1,2], DONG Yun-Wei[3], FENG Wen-Long[1,2], HUANG Meng-Xing[1,2]

[1](State Key Laboratory of Marine Resource Utilization in South China Sea (Hainan University), Haikou 570228, China)

[2](College of Information Science and Technology, Hainan University, Haikou 570228, China)

[3](School of Computer Science, Northwestern Polytechnical University, Xi'an 710129, China)

Abstract: Cyber-Physical System (CPS) tightly integrates control software and embedded components, incorporating software with control domains. CPSs are pervasive and often mission-critical, therefore, they must have high level of security. With the extensive use of

* 基金项目: 国家自然科学基金(61462022, 61363071); 国家科技支撑计划(2014BAD10B04, 2015BAH55F04); 海南省重大科技计划(ZDKJ2016015); 海南省自然科学基金(614232, 614220); 海南省产学研一体化专项(cxy20150025); 海南大学科研启动基金(kyqd1610)

Foundation item: National Natural Science Foundation of China (61462022, 61363071); National Key Technology R&D Program of China (2014BAD10B04, 2015BAH55F04); Major Science and Technology Project of Hainan Province (ZDKJ2016015); Natural Science Foundation of Hainan Province (614232, 614220); Enterprises-Universities-Researches Integration Project of Hainan (cxy20150025); Scientific Research Staring Foundation of Hainan University (kyqd1610)

收稿时间: 2016-07-15; 修改时间: 2016-09-25; 采用时间: 2016-12-07; jos 在线出版时间: 2017-01-20

CNKI 网络优先出版: 2017-01-20 16:06:33, http://www.cnki.net/kcms/detail/11.2560.TP.20170120.1606.005.html

information technology, embedded control software plays a greater role in such systems. The close interactions between control software and embedded components demand co-verification. In this paper, an automata-theoretic approach is presented to co-verification. Co-verification, which verifies control software and embedded components together, is essential to establish the correctness of a complete system. The foundation of this approach is a unified model for co-verification and reachability analysis of the model. The LTL formula is converted into a Büchi automata, which is interleaved with the execution of the unified model under analysis. An online-capture offline-replay approach is proposed to improve the usefulness for formal verification. Case studies on a suite of realistic examples show that the presented approach has major potential in verifying system level properties, therefore improving the high-assurance of system.

Key words: cyber-physical system; embedded control software program; automata theoretic; co-verification; bounded model checking

信息-物理融合系统(cyber-physical system,简称 CPS)[1]是一种新型的复杂嵌入式系统计算方式,其运行融合了计算过程和物理过程,两者之间具有密切的交互与影响.与传统的信息系统或物理系统相比,CPS 系统具有显著的不同特征.例如,与嵌入式系统相比,CPS 需要更多对物理实体的可控性;与物联网/传感网相比,CPS 更强调物理节点中融入复杂的计算过程以及对物理环境变化的适应性;与混成系统(hybrid system)相比,CPS 不仅仅是简单包含离散过程和连续过程的系统,而是需要二者更加深入的融合和可控,以及更高的精确性和适应性.

随着信息技术在现实世界中更加广泛、深入的应用,智能化程度不断提升,系统规模和复杂度日趋增大,在具有信息-物理紧密耦合特点的新型嵌入式系统中,嵌入式软件的功能比重急剧上升,作用更加突出.国际上在军事、航空、航天、医疗、交通等关键应用领域,由核心控制应用软件设计缺陷所带来的系统失效造成了巨大的损失,已经引起了人们的高度重视,我国在卫星、飞机等重点型号研制过程以及投入使用后所发生的异常事件或重大事故中有不少也与其嵌入式软件相关.不断发生的控制软件错误或失效所引发的事故使人们逐渐认识到,在系统复杂性较高的情况下,面对 CPS 这一新型体系结构,常规的软件工程方法和软件评测手段不能解决嵌入式控制软件安全性设计深层次的问题.

因此,作为安全攸关的系统,需要引入形式化验证方法来保证嵌入式控制应用程序的安全性.模型检验作为一种形式化验证技术,已经在硬件设计和协议实现方面取得成功的应用.随着抽象、符号化等技术的发展,模型检验逐渐发展到对源代码进行验证.人们研制了数个验证工具原型,包括 CBMC[2],CPAchecker[3],微软的 SLAM[4]项目,加州大学伯克利分校的 BLAST[5],美国国家航空航天局开发的 JPF[6],美国贝尔实验室的 SPIN[7]等.目前将模型检验技术应用于软件源程序的验证是当今嵌入式系统前沿领域的热门研究内容.

面向 CPS 的自主控制应用程序的运行一般包含感知(sensing)、控制应用(controlling)和实施(actuating)这 3 个过程.这 3 个过程形成了控制器软件的控制环路(control loop)或反馈环路(feedback loop).而控制器应用软件是这类系统的核心.它依据传感器信号来调整发送至作动器的输出信号,以改变受控对象(plant)状况.由控制器软件构成的闭环控制系统如图 1 所示,其运行既包含连续动态行为(受控对象)又包含离散动态行为(控制应用软件),系统的状态空间由离散变量和连续变量共同刻画.

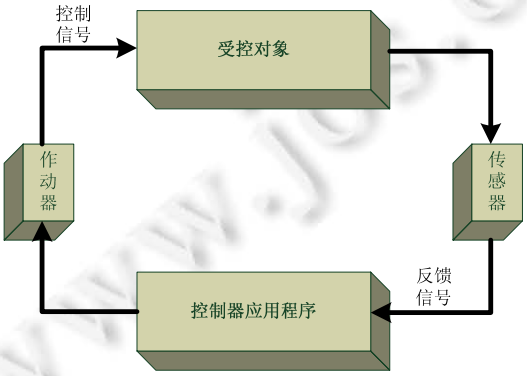


Fig.1 Control loop between control software program and plant

图 1 由控制应用程序和受控对象组成的控制环路

目前相关研究如下:文献[8,9]提出构造系统的时间自动机模型使用 UPPAAL 间接对控制软件进行了应用验证;文献[10,11]只是针对系统模型利用模型检验和定理证明技术来验证;文献[12–14]只是针对软件实现的控制器的进行了分析,基于符号执行技术提出了一种计算软件控制性能的方法;文献[15]提出了一种控制闭环分析方法,通过集成一个显示的模型检验工具检查控制器软件和 Simulink 去仿真运行受控对象;文献[16]提出了一个验证框架,通过模型转换将 Simulink/Stateflow 转换为统一的中间表示,然后基于中间表示做符号执行.文献[17,18]都对整个控制闭环做了静态分析,包括控制器程序和受控对象,基于可达性分析对系统某些属性进行了验证,并且只能处理定点计算.

目前,控制应用类软件模型检测主要存在以下两个问题.

(1) 如何构建系统属性的规约.目前的源程序验证工具主要采用基于断言的属性规约描述,在所需验证的程序位置加入 assert(expression)语句,基于可达性判断该断言在所有执行路径下是否成立.这种基于断言的属性规约具有一定的局限性,它无法表达时序、活性(livness)等属性.

(2) 如何有效构建控制环路验证模型以支持各种变化的模型检验.系统中控制应用程序和受控对象之间具有时间交互特性,它们消耗资源并同时影响整个系统的行为.然而控制系统中的大规模复杂异构性、不确定性以及计算过程同步或异步管理需求导致系统的行为越来越复杂,问题的规模随之急剧增加,从而影响了可验证系统的规模,而这也正是著名的组合状态空间爆炸问题.

针对验证这类需要不断地与物理过程交互从而共同完成系统功能的控制应用程序,本文在已有研究的基础上[19],基于自动机理论提出了控制环路的验证模型及其优化验证算法:将控制应用 C 程序抽象成下推系统,受控对象被抽象成混成自动机,基于控制任务周期分析,在各类自动机的变迁关系添加同步标记,通过将各自成员进行笛卡尔乘积构造出系统组合后的乘积自动机.基于有界模型检验(bounded model checking,简称 BMC)[20]技术,将验证模型 k 步内行为采用布尔约束编码,然后利用 SAT/SMT 方法来寻找此布尔约束集的可行解,从而判定系统在 k 步内行为是否有不满足规约的情况.针对组合状态空间爆炸问题,通过偏序规约等手段化简模型规模,并结合虚拟仿真提出了综合虚拟仿真和形式化验证相结合的方法.

本文采用线性时序逻辑 LTL(linear temporal logic)[21]作为模型属性的规约语言,通过定义在物理连续变量上的布尔表达式描述物理过程,而后将 LTL 性质规约公式转换为 Büchi 自动机,即将性质分解为与其等价的一组断言,然后再将这组断言插装到控制应用程序程序的适当位置,根据断言的违反情况即可判断出性质的满足情况.运行时验证考虑控制应用这样的非终止程序,在任何时刻只能获得当前执行的一个有穷状态前缀,针对有穷路径基于四值语义[22]进行判断.

本文将探讨控制应用程序协同验证方法.本文第 1 节简述控制应用程序的特点并分析控制任务的时间周期.第 2 节描述系统验证属性.第 3 节描述系统验证模型.第 4 节描述验证算法.第 5 节简述工具原型的设计.第 6 节给出实验结果,并对实验结果加以分析.最后,第 7 节对本文加以总结,并指出进一步的工作方向.

1 面向 CPS 的控制应用程序分析

1.1 控制应用程序特点

C 语言是目前控制应用程序通常使用的编程语言,因此本文主要针对使用 C 语言编写的控制应用程序(如图 2 所示).由于运行环境、实现功能等因素,嵌入式自主控制应用软件具有一些固有特性,与通用软件相比,它需要更多的对物理实体的可控性.这就增加了系统设计与功能、性能验证的难度,其中时间和并发操作是系统复杂性的主要来源.控制应用程序的主要特点.

(1) 物理交互性.控制应用程序能够从物理环境中监测相关数据,并对数据进行计算,然后根据控制逻辑通过作动器作用于物理环境.此类系统通过计算过程和物理过程相互影响的反馈循环实现这两者的深度融合和实时交互,以安全、可靠、高效和实时的方式监测或者控制一个物理实体.这使得嵌入式系统软件的验证必须考虑物理过程的影响,而计算过程和物理过程的同步过程和异步过程的交织不仅增加了软件设计的复杂度,也极大地增加了验证的难度.

(2) 时间约束性.在嵌入式系统中,某一个任务的正确性不仅取决于其功能和行为特性,还依赖于时间约束特性.这种严格的实时性要求使得诸如航空航天、武器控制、医疗设备等具有高安全可靠要求的嵌入式软件开发从需求分析、设计实现、验证测试均必须显式地考虑时间约束.不仅要验证软件的功能,而且要验证任务的执行时间.

(3) 存在位操作、float/double 等底层数据类型.针对浮点等底层数据类型,主要是计算精度问题,目前主要的做法有两种:近似和精确求解.近似目前有过近似(over-approximate)和近似(under-approximate)两种方法.常用的过近似方法是区间计算(interval arithmetic),区间计算将每个变量 x 定义在一个区间 $[\underline{x},\overline{x}]$,然后在变量的算术运算过程中将区间加入运算,在不断地迭代过程中有可能是最后结果的区间变得过大而失去意义.如 $x\in[1,3]$, $y\in[5,7]$, $x+y$ 通过区间计算后 $x+y\in[6,10]$.仿射计算(affine arithmetic)是近似方法,它将每个变量 x 定义为一个多项式 $x=x_0+x_1\cdot\varepsilon_1+x_2\cdot\varepsilon_2+...+x_k\cdot\varepsilon_k$, $\varepsilon_1,\varepsilon_2,...,\varepsilon_k\in[-1,1]$ 是误差.同样的例子,$x=3+\varepsilon_1$,$y=5+\varepsilon_1$,$x+y\in 8+\omega\cdot\varepsilon_2$,相对于区间计算,这样的结果更有价值,但是计算开销也比较大.精确求解是通过建立位模型(bit-vector model)和按位复写(bit-blasting)来实现,这样产生的表达式非常精确,但是计算开销相对于近似方法会大很多.本文是采用精确求解方法,求解过程参考 Clarke 等人的工作[1].

如图 2 所示是一个控制应用程序的典型结构.控制应用程序中存在控制循环,在每次控制循环中,首先从传感器读取数据完成 A/D 转换,然后基于采样数据计算控制指令,最后将控制指令写入作动器完成 D/A 转换.因为对程序中所有的循环加了上界,所以程序循环边界(loop bound)是可知的,程序必然是终止的,这是和其他类型程序相比的一个特点.以 while 循环为例来演示如何将其展开,其他循环的展开方式类似.while 循环的一次展开过程如下:while(s) L;⇒if(s){L;while(s) L;}.在已经设定上界为 n 的情况下,我们将上述循环展开 n 次,并在最后一次循环展开加入一个断言来表示,即 assert(!s).

```
01 int main(int argc, char **argv, char **envp){
02     initial();
03     while(TSrunning){
04         waitClock();
05         /* Read the raw sensor values */
06         if(!readSensors(&SensorReading)) {
07               printf("ReadSensor error \n");
08               return;
09         }
10         calculateOutput();
11          /* Actually fire off the motors */
12          commandMotor(Actuator.FanVoltage);
13          updateState();
14     }
15     Return;
16 }
17 int ReadSensors(SensorReading_t *sensor)
18 {    ...
19     if((result = dscADScan(dscb, &dscadscan, samples))!= DE_NONE){
20       ...}
21 }
22 void commandMotor(double *v)
23 {    ...
24     if((result = dscDAConvert(dscb, pos_channel, pos_counts)) != DE_NONE){
25         ...}
26 }
```

Fig.2 Excerpt of a control program

图 2 控制应用程序结构示例

1.2 控制任务周期分析

采样器通过一定时间间隔(控制周期)的采样把连续的偏差信号转换成离散信号,由控制应用程序对它进行适当的变换,以满足控制的需要,最后通过保持器再将数字控制器输出的离散控制信号转换成连续的控制信号去控制被控对象.周期性任务的时序关系[23]如图 3 所示,采用固定控制周期,控制周期时刻 $t_k=T\times k$,其中,T 是控制时间间隔,k 是控制次数.t_i 和 t_o 分别表示 A/D 和 D/A 转换完成的周期时刻.

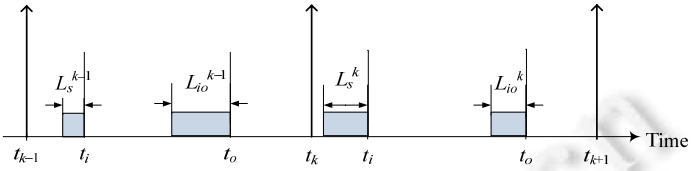


Fig.3 Excerpt of a control task

图 3 控制任务的周期时间

由于操作系统任务调度的阻塞和抢占,所以用 L_s 表示采样延迟时间,即实际任务的采样时刻可能会被延迟 L_s 一段时间.由于调度策略的动态变化使得采样延迟时间并不固定,我们用 J_s 表示采样抖动,并用最小延迟时间和最大延迟时间定义抖动的范围,即 $J_s\in[\min L_s,\max L_s]$.

在完成计算和可能的任务抢占之后,控制应用程序开始通过 D/A 转换发出控制信号,我们用 L_{io} 表示输入/输出延迟时间.由于优先级抢占的调度策略和控制器计算执行时间的不定使得输入/输出延迟时间并不固定,我们用 J_{io} 表示输入/输出抖动,并用最小延迟时间和最大延迟时间定义抖动的范围,即 $J_{io}\in[\min L_{io},\max L_{io}]$.

我们假设 t_i' 和 t_o' 分别表示 A/D 和 D/A 转换的设计完成时刻.因此 A/D 转换完成的实际时刻 $t=t_i'+J_s$, D/A 转换完成的实际时刻 $t_o=t_o'+J_{io}$. A/D 转换的延迟可能会对控制应用程序的执行产生影响,即在不同的采样值可能会导致程序的执行路径不一样,而 D/A 转换的延迟可能会对受控对象的执行产生影响,尤其当前后两个控制周期的控制命令不一样时,这决定了从何时刻开始受控对象开始在新的控制指令下运行.因此,在本文的研究中将考虑不同情况下延迟对整个系统的影响.

2 验证性质描述

模型检验通常关注系统两个方面的性质:安全性(safety)和活性(livness).

(1) 安全性(safety),即 $G\varphi$,坏的事情永远不会发生.安全性可以转化为可达性分析,即通过可达性分析看是否存在一条路径从初始状态到达这个不安全的状态.

(2) 活性(livness),即 $G(p\rightarrow F\varphi)$,是另一类功能性要求,即好的事情总会发生.

本文使用 LTL 描述系统属性,然后将要验证的断言添加到控制应用程序中.通过定义在物理连续变量上的布尔表达式描述物理过程,使用 LTL 描述控制应用程序的离散事件和受控对象的连续时间的时序关系,例如一个活性属性规约,$\boldsymbol{G}(\{Rotary.Velocity>T\}\rightarrow\boldsymbol{F}\{Actuator.FanVoltage\}=fullNeg)$,当转盘的角速度超过限制 T 时,控制器最终会将风扇的电压设为 *fullNeg*.这条规约涉及连续变量 *Rotary.Velocity* 的值,以及离散控制变量 *Actuator.FanVoltage* 的值.

LTL 标准语义是定义在无穷路径上,针对有穷轨迹.文献[22]提出了 LTL 四值语义的思想.

定义 1. 假定 $\varphi\in$LTL,$i\in\mathbb{N}$表示序列中的一个位置,$u=u_0u_1...u_n\in\Sigma^*$ 为有穷状态序列,$\omega\in\Sigma^\omega$ 为无穷状态序列,u^i 从序列中的第 i 个位置开始的有限后缀($u^i=u_iu_{i+1}u_{i+2}...u_{n-1}$,其中,$i<n$,如果 $i\geqslant n$,则为ε),a^ω表示只由 $a\in\Sigma$ 所组成的无限路径,$[u\vDash\varphi]_\omega$ 表示$[u\vDash\varphi]$是定义在无穷路径ω上的,那么 LTL 公式的四值语义被定义如下:

$$B_4=\{\mathrm{T},\mathrm{T}^P,\bot,\bot^P\},\bot\subseteq\bot^P\subseteq\mathrm{T}^P\subseteq\mathrm{T},\overline{\bot}=\mathrm{T}\text{和}\overline{\bot^P}=\mathrm{T}^P,$$

$$[u\vDash\varphi]_B=\begin{cases}\mathrm{T} & \text{iff } \forall\omega\in\Sigma^{\omega},\ [u\vDash\varphi]_{\omega}=\mathrm{T}\\ \mathrm{T}^P & \text{iff } \left[uu_{n-1}^{\omega}\vDash\varphi\right]_{\omega}=\mathrm{T}\wedge\exists\omega\in\Sigma^{\omega},\ [u\omega\vDash\varphi]_{\omega}=\bot\\ \bot & \text{iff } \forall\omega\in\Sigma^{\omega},\ [u\omega\vDash\varphi]_{\omega}=\bot\\ \bot^P & \text{iff } \left[uu_{n-1}^{\omega}\vDash\varphi\right]_{\omega}=\bot\wedge\exists\omega\in\Sigma^{\omega},\ [u\omega\vDash\varphi]_{\omega}=\mathrm{T}\end{cases}.$$

3 系统验证模型

软件源程序直接进行模型检验的基础是模型抽象,即从程序中抽象得到有限状态空间模型.下推系统(pushdown system,简称 PDS)[24]为带递归的程序提供了一个理想的执行模型,但是下推系统并不接受输入.在一个控制闭环中,受控对象的行为可能会影响软件的执行,例如采样.因此,下推系统需要扩展,即要接受受控对象的输入.基于已有的研究基础[19],带标记的下推系统(labeled pushdown system,简称 LPDS)的定义如下.

定义 2. 带标记的下推自动机是具有如下形式的五元组 $\left(I,G,\Gamma,\Delta,\left\langle g_0,\omega_0\right\rangle\right)$, 其中各元素的含义如下.

I:输入字母表的有限集合.

G:状态的有限集合.

Γ:堆栈字母表的有限集合.

Δ: $\Delta\subseteq(G\times\Gamma)\times I\times(G'\times\Gamma^{*})$ 是状态迁移的集合,表示为 $\langle(G\times\Gamma),I,(G'\times\Gamma^{*})\rangle$, 迁移发生是指输入标记 τ 时,发生从源状态($G\times\Gamma$)到目标状态($G'\times\Gamma$*)的转移,记作 $\left\langle g,\gamma\right\rangle\xrightarrow{\tau}\left\langle g',\omega\right\rangle\in\Delta$. τ可以缺省,此时τ为空,表示无条件迁移.

$\left\langle g_0,\omega_0\right\rangle$: 初始状态.

带标记的下推系统 LPDS 的状态为一个二元组 $s=\langle g,\omega\rangle$,其中,$g\in G,\omega\in\Gamma$,系统轨迹则是由初始状态经变迁所构成的状态序列.

混成系统通常采用混成自动机(hybrid automata,简称 HA)[25]来建模,混成自动机是在状态机基础上进行实时连续变量扩展所构成的一种建模语言.受控对象被抽象成混成自动机 HA,其定义如下所述.

定义 3. 混成自动机为多元组 $HA=(\tilde{x},\tilde{\mathrm{x}}_0,V,v_0,inv,dif,E,act,lab,syn)$, 其中各元素的含义如下.

$\tilde{x}=(x_1,x_2,x_3,x_n)\in\mathbb{R}^n$ 是实数值连续状态变量的有限集合.

$\tilde{x}_0$ 是连续变量初始条件.

V 是离散状态(位置节点)的有限集合.

$v_0\in V$ 是初始位置的集合.

inv 是一个标注函数,它为每个位置 $v\in V$ 赋予一个关于 $\tilde{x}$ 的不变集约束 $inv(v)$.

dif 是一个标注函数,它为每个位置赋予一个关于 $\tilde{x}$ 的微分方程.

$E\in V\times V$ 是所有离散变迁(或边)的有限集合.

$act=\{act(e)\,|\,e\in E\}$ 是一个将 E 中的转换 e 标注为一组约束的标注函数,它为每条边 $e\in E$ 赋予一个能触发离散变迁的连续变量取值条件.

lab 是一组同步标记的集合.

syn 是一个将 E 中的转换 e 标注为一组同步约束的标注函数.

图 4 为 TableSat 受控对象的混成自动机模型(TableSat 介绍详见第 6.2 节),其中连续变量ω表示转盘角速度,v表示风扇转速,t 表示时间,离散位置 On_AD,On_DA,On_Rest 表示在风扇电压处于“12V”状态下,转盘分别处于采样、计算控制和控制后状态.离散位置 Off_AD,Off_DA,Off_Rest 表示在风扇电压处于“0V”状态下,转盘分别处于采样、计算控制和控制后状态.假设起始转盘角速度,风扇转速,开始时刻都为 0,风扇电压处于“12V”状态.当风扇电压处于“0V”时,转盘角速度ω随微分方程变化而下降;当控制指令 On 到达时,风扇电压将由“0V”状态转变为“12V”状态,系统则变迁至 On 模式开始加速;此时,若风扇电压处于“12V”,则转盘角速度ω随微分方程变化而上升;若控制指令 Off 到达,则风扇电压将由“12V”状态转变为“0V”状态,系统变迁至 Off 模式又开始

减速.

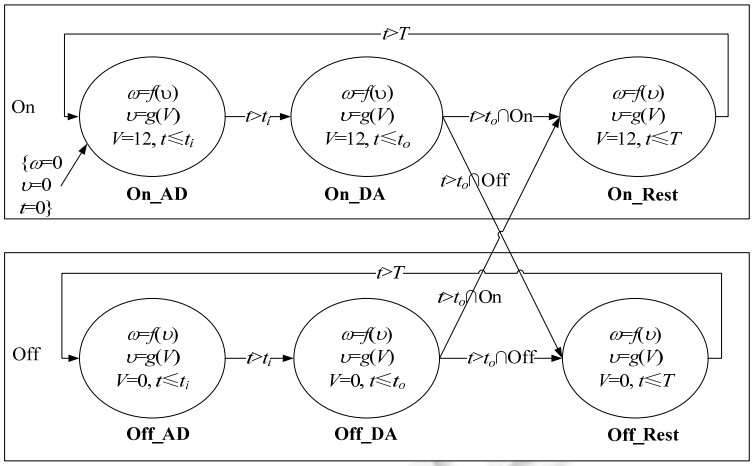


Fig.4 Hybrid automat model of TableSat plant

图 4 TableSat 受控对象的混成自动机模型

由于混成自动机中连续行为非常复杂,现有的相关工具大都使用多面体计算来判定线性混成自动机状态空间的可达集,复杂度高、效率低,无法解决实际应用规模的问题.因此,本文的研究主要关注于其中一个比较特别的子类——线性混成自动机.线性混成自动机的流条件一般分两类.一类形如 $\dot{x}\in[a,b]$ 的自动机形式,相应的流条件积分展开后可以得到变量 x 的取值是与时间 t 相关的线性函数.另一类形如 $\dot{x}=Ax(t)+Bu(t)$, 其积分展开后会成为包含 e^t 的非线性方程.第 1 类可以精确求解,第 2 类形式主要依靠数值解法来解决,把连续性的问题加以离散化,采用“步进式”求出离散节点的数值解,即

$$\dot{x}=Ax(t)+Bu(t)\Rightarrow x[k+1]=A_d x[k]+B_d u[k].$$

数值解法会引入误差,以经典四阶龙格库塔法(Runge-Kutta)为例,RK4 法是四阶方法,也就是说,每步的误差是 h^5阶,而总积累误差为 h^4 阶.要提高精度需要根据问题选择合适的数值算法.

下面给出协同验证模型的统一形式描述,并建立模型描述的层间关系.

定义 4(标记迁移系统). 一个标记迁移系统 TS 是一个四元组 $TS=(States,Labels,\rightarrow,S_0)$, 其中,$States$ 是一个状态的集合,$Lables$ 是一个标记的集合, $\rightarrow: States\times Labels\times States$ 状态转换的集合, $S_0\in States$ 是初始状态的集合.

定义 5. 一个系统统一验证模型的形式描述表示为 $HAPS=LPDS||HA$,表示系统组合同步后的乘积自动机.

HAPS 将各成员依据共享事件标签进行笛卡尔乘积,构造出系统组合同步后的乘积自动机.因此,针对 LPDS 每条变迁,都要搜索 HA 的所有变迁去构建 HAPS 的变迁规则,即需要在 LPDS 模型每条变迁的后面插装 HA 模型.HAPS 的语义可以定义为一个状态迁移系统.HAPS 的状态迁移也分为连续迁移和离散迁移两种类型,其中离散迁移又分为两类:一类是 HAPS 中的子自动机独立地执行其内部迁移而引发的 HAPS 的状态迁移;另一类是 HAPS 中的两个子自动机之间通过同步标记进行同步,然后这两个子自动机执行其内部迁移而引发的 HAPS 的状态迁移.

首先定义同步标记和同步标注函数.

(1) 带标记的下推系统 LPDS 的输入字母表 I 是定义在混成自动机 HA 离散状态上的原子命题的集合的幂集;

(2) 混成自动机 HA 的同步标记 lab 是定义在带标记的下推系统 LPDS 状态上的原子命题的集合的幂集;

(3) 两个同步标注函数:

$L_{P2H}:(G\times\Gamma)\to lab$,在 LPDS 的每个配置赋予一组命题集合,以 TableSat 为例,图 2 中 L_{P2H} 的赋值情况为 $L_{P2H}(\Diamond_3,dscDAConvert_0)\to da$,程序其余状态都为 $L_{P2H}(\Diamond_0,main_2)\to no\_evt$.

$L_{H2P}:V\to I$,在 HA 的每个离散状态赋予一组命题集合,以 TableSat 为例,图 4 中的 L_{H2P} 的赋值情况为 $L_{H2P}(\text{On\_AD})\to \text{ad}$, $L_{H2P}(\text{Off\_AD})\to \text{ad}$,其余离散状态都为 $L_{H2P}(xx)\to no\_event$.

定义 6. 系统统一验证模型(HAPS)的状态是 $s=(g,\omega,v,\tilde{x})$,其中,g 是 LPDS 的状态,ω是 LPDS 的栈字母表,$\tilde{x}_0$ 是连续变量初始条件,V 是 HA 的离散状态.

假设有一个标记迁移系统 S,其初始状态为 S_0.一个以 S_0 为根的系统行为轨迹是一个有限的或者无限的序列,其路径是

$$t=init(s_0)\wedge\Delta_{p'}(s_0,s_1)\wedge\Delta_{p'}(s_1,s_2)\wedge\ldots\wedge\Delta_{p'}(s_{k-1},s_k)\wedge\ldots,$$

k 步的路径为

$$[HAPS]^k=init(s_0)\wedge\Delta_{p'}(s_0,s_1)\wedge\Delta_{p'}(s_1,s_2)\wedge\ldots\wedge\Delta_{p'}(s_{k-1},s_k).$$

不确定因素.系统在运行过程中通常会有不确定因素,如应用程序向作动器写入控制指令,在第 1.2 节的分析中,由于操作系统优先级抢占的调度策略和控制器计算执行时间的不定使得输入/输出的延迟J_{io}可能会不同,因此控制指令的输出时刻并不确定.通常情况下,我们希望系统能够描述在一定范围内的所有可能输出时刻.因此,在可达性的分析过程中,我们将这些不确定因素通过不确定函数 non-determinism 来描述,并将不确定量定义在一个固定集合中,如 $J_{io}\in[m,n]=\{r\in\mathbb{R}\mid m\leqslant r\leqslant n\}(m,n\in\mathbb{R})$ 用于对控制应用程序或物理模型的时钟变量做约束,在程序中表示为{float non_determinism();J_{io}=non_determinism();__CPROVER_assume((n<=J_{io})&&(J_{io}<=m));}.

4 系统验证算法

根据混成自动机网络的操作语义,网络中的各自动机既可以并发执行各自的动作,又可以在共享事件标签上实现同步通信.然而,依据此方法所生成的组合自动机的规模随着成员数目的增长而急剧增加,从而影响了可验证系统的规模,而这也正是著名的组合状态空间爆炸问题.为了应对该问题,一是利用偏序规约化简组合自动机的迁移关系;二是将形式化验证技术和协同仿真技术结合起来,通过仿真观察记录系统的运行,并利用仿真的运行记录对某些重要节点状态进行形式化验证.

4.1 可达性

有界模型检验方法的基本思想是有限步的展开程序控制流程图,检查在该步数内能否找到错误状态,若找到,则返回系统不安全的信息,若找不到,则增加步数,直到步数到达上界,此时返回找不到错误状态的信息.

(1) 可达性,即 $F\varphi$.我们用下面的式子表达系统模型是否在 k 步内可达.

$$[M\vDash\mathcal{F}\varphi]^k=init(s_0)\wedge\Delta(s_0,s_1)\wedge\ldots\wedge\Delta(s_{k-1},s_k)\wedge\bigcup_{i=0}^{k}\varphi(s_i).$$

通过将系统验证模型转换为下推系统 PDS,那么验证模型的可达性问题可被转换为 PDS 的可达性分析,然后就可以使用 PDS 的模型检验工具进行验证.通过转换得到的 $PD=(G,\Gamma,\Delta,c_0)$, 其中, $G=V\times G'\times\tilde{x}$, $c_0=\langle(v_0,g_0),\tilde{x}_0,\omega_0\rangle$, V 是定义为整数型的离散状态,G'是程序状态的有限集合,$\tilde{x}$ 是定义为浮点类型的实数集,Γ是程序堆栈字母表的有限集合,Δ将各成员自动机依据共享事件标签进行笛卡尔乘积所产生的变迁条件.

一个程序的状态可以表达为 $s=\langle g,\omega\rangle$, 其中,$g\in G$ 是为程序状态的有限集合,$\omega\in\Gamma$是程序栈字母表的有限集合.可达集合是在程序的指定位置,并且变量满足一定的约束.这里的约束是一个谓词表达式,例如 x=12∧y>13.4,也就是所有的变量值满足 x=12∧y>13.4.在程序的指定位置插入断言 assert(x=12∧y>13.4),就表达了一个状态集合.基于断言的属性规约描述了从程序的初始状态到某个状态集合的可达性.通常情况下,这个断言是一个否定形式的表示,那么当一个满足公式的赋值被 SAT/SMT 找到之后,这个赋值对应的路径就是一个违反需要验证的性质的反例.

4.2 LTL模型检验算法及优化

1. 模型检验算法

1) 安全性转化为可达性分析

可达性问题与安全性问题的关系在于,将系统不安全的行为构建成一个独立的离散状态,然后检验此状态是否可达:如可达,则系统中可能发生不安全行为;反之,则系统行为安全.因此,检验系统安全性的问题就变成了计算系统可达空间的问题.

2) 活性转化为基于四值语义的 LTL 检测

系统模型是 M,系统的属性φ是用 LTL 描述,我们的目的是通过有界模型检验 M 是否满足 LTL 描述的规约φ.如果把φ看成某类自动机,模型检验问题就是判断 M 的语言是否包含在φ的语言,即 $L(M)\subseteq L(\varphi)$.时序逻辑 LTL 的活性验证难点在于:

(1) 考虑研究如何把 LTL 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.

(2) 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.

Büchi 自动机(Büchi automata,简称 BA)的定义如下.

定义 7. Büchi 自动机是具有如下形式的五元组 (Σ,Q,δ,q_0,F), 其中各元素的含义如下.

Σ是输入字母表的有限集合.

Q 是状态的有限集合.

δ: $\delta\subseteq(Q\times\Sigma\times Q)$ 是状态转换的集合.

q_0: $q_0\in Q$ 是初始状态.

F: $F\subseteq Q$ 是终止状态集合.

LTL 检测的基本思想是做 Büchi 自动机的判空运算.因此,LTL 检测算法包括 3 步.

1) 给定系统性质描述的 LTL 公式φ,并利用 LTL2BA 工具[26]构造相应的广义 Büchi 自动机 $A_{\neg\varphi}$.

目前,C 程序广泛应用在嵌入式系统开发中,其语义被工程师广泛理解,因此我们把 BA 转换成等价的用 C 程序描述的一组断言.如图 5 所示.对于公式$\varphi=\boldsymbol{G}(p\rightarrow\boldsymbol{F}q)$,首先对公式$\varphi$取反,即$\neg\varphi$.图 5 左边是$\neg\varphi$所对应的 $BA_{\neg\varphi}$,右边是等价的用 C 程序描述的一组断言.自动机是独立于程序本身的,因此与程序分离不会影响程序自身的结构.在嵌入式多事件系统中,该方法将有利于对系统属性规约.如图 3~图 4 所示,每次执行 monitor 函数时,通过__CPROVER_assume()语句使得每当 $BA_{\neg\varphi}$ 迁移条件满足时才能进入其后的语句,即记录当前状态.当程序要终止时,再根据 statevar 状态值和程序的状态,依据判定准则进行判断.

2) 利用同步积算法求出积自动机 $B=HAPS\times A_{\neg\varphi}$. 系统模型每发生一次转移,由 LTL 转化为的 Büchi 自动机也要发生一次转移.这就需要将 LTL 时序逻辑公式插装到在系统验证模型等价的 C 程序里的每条语句后.

3) 程序运行时,验证器检测判定 $L(B)$是否为空,如为空,则说明构建的 HAPS 模型满足性质;否则,不为空的接收集即是违背性质的反例.

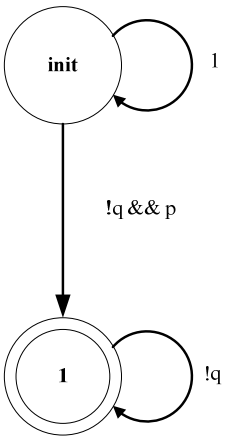
生成接收有穷序列的自动机(验证器),该自动机接收当前执行的有穷前缀,输出相应的验证结果,使得在存在软件缺陷的情况下尽可能早地检测到该缺陷.基于以往执行轨迹(有穷前缀)的记录,能够根据基于四值语义检测系统缺陷和异常,及早终止对程序的进一步检测.基于四值语义的 LTL 的判定准则.

(1) $[S\vDash\varphi]_B=\mathrm{T}$ iff $\nexists s\in S,\omega\in\Sigma^{\omega},[s\omega\vDash\neg\varphi]_{\omega}=\mathrm{T}$;

(2) $[S\vDash\varphi]_B\supseteq\mathrm{T}^P$ iff $\nexists s\in S,\left[ss_{n-1}^{\omega}\vDash\neg\varphi\right]_{\omega}=\mathrm{T}$;

(3) $[S\vDash\varphi]_B=\perp$ iff $\exists s\in S,\forall\omega\in\Sigma^{\omega},[s\omega\vDash\neg\varphi]_{\omega}=\mathrm{T}$;

(4) $[S\vDash\varphi]_B\supseteq\perp^P$ iff $\exists s\in S,\left[ss_{n-1}^{\omega}\vDash\neg\varphi\right]_{\omega}=\mathrm{T}$.




```
void monitor(bool state_stats, int num_iters)
{
...
   choice = nondet_int();
   switch(statevar) {
      case state_0:
         if (choice == 0) {
            __CPROVER_assume(1);
            statevar = state_0;
         } else if (choice == 1) {
            __CPROVER_assume(!q && p);
            statevar = state_1;
         } else {
            __CPROVER_assume(0);
         }
         break;
      case state_1:
         if (choice == 0) {
            __CPROVER_assume(!q);
            statevar = _state_1;
         } else {
            __CPROVER_assume(0);
         }
         break;
  If (state_stats)
   visited_states[statevar]++;
  }
...
return;
}
```

Fig.5 An LTL property expressed as Büchi automata and C

图 5 利用 Büchi 自动机与相应的 C 程序所描述的一条 LTL 属性

LTL 根据当前的有穷轨迹区分下面 3 种情况.

(1) 当前观测到的有穷轨迹 u 能够证明待验证的性质 φ 成立,而不用考虑将来的行为.

(2) 当前观测到的有穷轨迹 u 能够证明无论将来发生什么行为,待验证的性质 φ 始终不成立.

(3) 当前观测到的有穷轨迹 u 不能证明待验证的性质 φ 成立与否,需要根据观测到的有穷轨迹 u 预测 φ,这种情况下又分两种情形.

(1) 当前观测到的有穷轨迹 u 不违反 φ,则 φ 有可能成立.

(2) 当前观测到的有穷轨迹 u 违反 φ,则 φ 有可能不成立.

2. 算法的优化

系统验证模型是将各成员依据共享事件标签进行笛卡尔乘积,因此需要在控制应用程序每条语句的后面插装受控对象模型.因此,因为 HPDS 的变迁关系并不是严格意义上的笛卡尔积,LPDS 和 HA 的各自变迁必须满足使能关系,仅考虑离散变迁,算法的时间复杂度 $O(|\delta|\times|\Delta|)$.算法的优化包括对可达性分析和 LTL 的模型检查算优化.下面分别加以介绍.

(1) 对于可达性分析,使用偏序规约简化

首先定义 3 个概念描述不同变迁条件之间的关系.

使能性(enabledness).如果当且仅当 $\tau \subseteq L_{H2P}(v)$, 则称一个 LPDS 的变迁关系 $r=\langle g,\gamma\rangle \xrightarrow{\tau} \langle g',\omega\rangle \in \Delta$ 在一个 HA 离散状态 V 使能,否则,γ 被 V 否定发生变迁.同样地,对于一个 HA 的离散变迁 $d=\langle v,\tilde{x}\rangle \xrightarrow{\alpha} \langle v',\tilde{x}'\rangle$ 能够被一个 LPDS 的配置 $\langle g,\gamma\rangle$ 使能发生变迁当且仅当 $\alpha \subseteq L_{P2H}(g,\gamma)$, 否则,$d$ 被 $\langle g,\gamma\rangle$ 否定发生变迁.

例如一个 LPDS 的变迁关系 $\langle \Diamond_1, reanSensors_0\rangle \xrightarrow{ad} \langle \Diamond_1, dscADScan_0 reanSensors_1\rangle$ (图 2 中第 19 条语句)和一个 HA 离散状态 On_AD(如图 4 所示),L_{H2P}(On-AD)→ad,则离散状态 On_AD 使变迁 $\langle \Diamond_1, reanSensors_0\rangle \xrightarrow{ad} \langle \Diamond_1, dscADScan_0 reanSensors_1\rangle$ 能够发生.但是对于 HA 离散状态 On_DA,因为 L_{H2P}(On-DA)→no_vert,所以离散状态 On_DA 使变迁 $\langle \Diamond_1, reanSensors_0\rangle \xrightarrow{ad} \langle \Diamond_1, dscADScan_0 reanSensors_1\rangle$ 不能发生.

不可分辨性(indistinguishability).给一个 HA 的离散变迁 $d=\langle v,x\rangle\xrightarrow{\alpha}\langle v',\tilde{x}'\rangle(e=(v,v'),\tilde{x}\mapsto act(e))$, 一个 LPDS 的变迁关系 $r=\langle g,\gamma\rangle\xrightarrow{\tau}\langle g',\omega\rangle\in\Delta$ 对 d 是不可分辨的当且仅当 $\alpha\subseteq L_{P2H}(g,\gamma)\cap L_{P2H}(g',\omega)$. 同样地,$d$ 对于γ是不可分辨的当且仅当 $\tau\subseteq L_{H2P}(v)\cap L_{H2P}(v')$.

例如对于一个 HA 的离散变迁 $t_2=\text{On\_DA}\xrightarrow{\{t>t_o\cap \text{On}\}}\text{On\_Rest}$ (如图 4 所示),一个 LPDS 的变迁关系 $\langle\Diamond_0,main_2\rangle\xrightarrow{no\_evt}\langle\Diamond_0,wait\_clock_0main_3\rangle$ (如图 2 中第 4 条语句所示),由于 $\text{On}\subseteq L_{P2H}(\Diamond_0,main_2)\cap L_{P2H}(\Diamond_0,wait\_clock_0main_3)$, 所以该 LPDS 的变迁关系对 t_2 是不可分辨的.但是对于另一个 LPDS 的变迁关系 $\langle\Diamond_3,commandMotor\rangle\xrightarrow{da}\langle\Diamond_3,dscDAConvert_0commandMotor_1\rangle$ (见图 2 中第 24 条语句),由于 $L_{P2H}(\Diamond_3,dscDAConvert_0)\cap L_{P2H}(\Diamond_3,dscDAConvert_0commandMotor_1)=\varnothing$, 所以该 LPDS 的变迁关系对 t_2 是可分辨的.

独立性(Independence).给一个 HA 的离散变迁 d 和一个 LPDS 的变迁关系γ,如果 d 和γ相互不可分辨,我们就说 d 和γ相互独立;否则,d 或者γ不是不可分辨并且 d 和γ满足使能关系,我们就说 d 和γ是相互依赖的.独立性是对称的.

例如对于一个 HA 的离散变迁 $t_1=\text{On\_AD}\xrightarrow{\{t>t_1\}}\text{On\_DA}$ (如图 4 所示)和一个 LPDS 的变迁关系 $\langle\Diamond_0,main_2\rangle\xrightarrow{no\_evt}\langle\Diamond_0,wait\_clock_0main_3\rangle$ (图 2 中第 4 条语句),因为 t_1 和 LPDS 的变迁关系相互不可分辨,所以它们相互独立.

对于一个 HA 的离散变迁 $t_2=\text{On\_DA}\xrightarrow{\{t>t_o\cap On\}}\text{On\_Rest}$ (如图 4 所示)和一个 LPDS 的变迁关系 $\langle\Diamond_3,commandMotor_0\rangle\xrightarrow{da}\langle\Diamond_3,dscDAConvert_0commandMotor_1\rangle$ (图 2 中第 24 条语句)不是不可分辨并且满足使能关系,所以它们相互依赖.

可达性的优化是针对 LPDS 和 HA 的异步变迁,如图 6 所示异步变迁主要有 3 类变迁关系:图中的水平线代表当 HA 变迁,LPDS 等待,即系统验证模型的这样一类变迁关系: $\Delta_{hori}=\langle(g,q),\gamma\rangle\to\langle(g,q'),\gamma\rangle$; 图中的垂直线代表当 LPDS 变迁,HA 等待,即系统验证模型的这样一类变迁关系: $\Delta_{vert}=\langle(g,q),\gamma\rangle\to\langle(g',q),\omega\rangle$; 图中的对角线代表 HA 和 LPDS 同时变迁,即系统验证模型的这样一类变迁关系: $\Delta_{diag}=\langle(g,q),\gamma\rangle\to\langle(g',q'),\omega\rangle$.

经过优化,如图 7 所示化简的异步变迁主要有两类变迁关系.化简部分变迁 Δ_{hori} : 图中的水平线代表当 HA 变迁,LPDS 等待,即 $\Delta_{hori}=\langle(g,q),\gamma\rangle\to\langle(g,q'),\gamma\rangle$. 化简所有变迁 Δ_{diag} : 图中的对角线代表 HA 和 LPDS 同时变迁,即 $\Delta_{diag}=\langle(g,q),\gamma\rangle\to\langle(g',q'),\omega\rangle$.

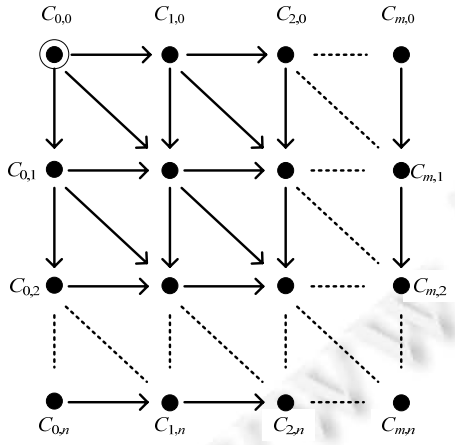


Fig.6 Complete transition graph

图 6 完整的变迁关系图

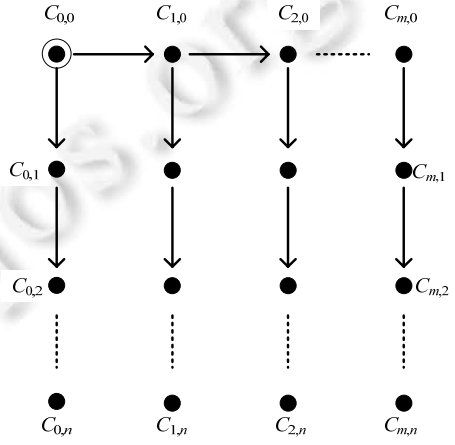


Fig.7 Reduced transition graph

图 7 化简后的变迁关系图

对比图 6 和图 7 可以看出,基于偏序规约思想化简全部的 Δ_{diag} 和部分的 Δ_{hori} 并不影响系统的可达性.通过定义关键集(sensitiveSet)来定义哪些变迁 Δ_{hori} 不能被化简掉.

定义 8(关键集 SensitiveSet). $SensitiveSet = \{head(\langle g_0, \omega_0 \rangle)\} \cup \{head(c') \mid \gamma$和$d$相互依赖$\}$.

根据偏序规约,优化后系统模型的变迁条件生产如算法 1 所示.算法 1 省去了全部 Δ_{diag} 和部分 Δ_{hori}. 因此,仅考虑离散变迁.如图 8 所示,统一验证模型的组装只需在控制应用程序特定语句后插装受控对象模型(函数 physical_run()).

算法 1. 优化后的 HAPS 变迁条件生成算法.

输入: $\Delta_{sync} \leftarrow \varnothing, \Delta_{hori} \leftarrow \varnothing, \Delta_{vert} \leftarrow \varnothing$.

输出:Δ'.

Forall the $r = \langle g, \gamma \rangle \xrightarrow{\tau} \langle g', \omega \rangle \in \Delta$ **do**

Forall the $d = \langle v, X \rangle \xrightarrow{\alpha} \langle v', X' \rangle (e = (v, v'), X \mapsto act(e)),\ \alpha \subseteq L_{P2H}(g, \gamma)$ and $\tau \subseteq L_{H2P}(v)$ **do**

if γ和 d 相互依赖 **then**

$\Delta_{sync} \leftarrow \Delta_{sync} \cup \{\langle (g,q), \gamma \rangle \rightarrow \langle (g',q'), \omega \rangle\}$ {LPDS 和 HA 必须同步变迁}

else

$\Delta_{vert} \leftarrow \Delta_{vert} \cup \{\langle (g,q), \gamma \rangle \rightarrow \langle (g',q), \omega \rangle\}$ {LPDS 变迁,HA 等待}

if $\langle g, \gamma \rangle \in SensitiveSet$ **then**

$\Delta_{hori} \leftarrow \Delta_{hori} \cup \{\langle (g,q), \gamma \rangle \rightarrow \langle (g,q'), \gamma \rangle\}$ {HA 变迁,LPDS 等待}

$\Delta' \leftarrow \Delta_{sync} \cup \Delta_{hori} \cup \Delta_{vert}$

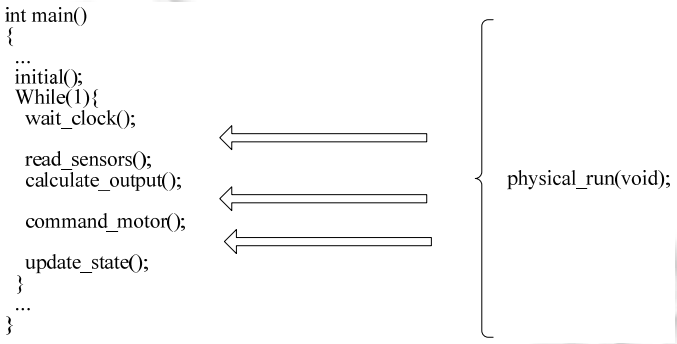


Fig.8 An example of BA and HPDS both specified in C programs

图 8 由 C 程序描述的 BA 和 HPDS 的组装示例

以 TableSat 为例,将图 4 表示的混成自动机转化等价的 C 程序.转换后的 C 程序如图 9 所示.第 1 行定义了 3 个浮点类型变量 *physical*,*nu* 和 *t*,分别表示 3 个连续变量 (ω,*v*,*t*); 第 2 行,函数 physcial_dynamic1 描述了 HA 模型的连续行为,即物理运动方程;第 5 行~第 30 行实现了 TableSat 的行为变迁,每次执行函数 physical_run,都会通过 system_mode(第 5 行和第 22 行)判断 HA 模型所处的状态:是处于 12V 的还是处于 0V 的状态;第 6 行、第 13 行和第 17 行,判断变量 *t* 是否满足 *act*(*e*);第 10 行、第 16 行和第 20 行通过调用函数 physcial_dynamic1 在一段时间内运行 HA 模型;第 7 行和第 8 行通过将时间 *t* 设为符号量,并假设限定在区间[*t_s*,*t_i*),第 14 行和第 15 行、第 18 行和第 19 行类似.

另外,由于使用了偏序规约优化,化简掉了部分系统的下一状态,所以 LTL 属性规约中不包括 *X* 算子.

(2) 对于活性

线性时序逻辑(LTL)对模型属性进行规约,模型每发生一次转移,由 LTL 转化为的 Büchi 自动机也要发生一

次迁移.根据第 3.5.3 节的算法,在将 LTL 时序逻辑公式自动转换为与其等价的一组断言后,我们需要将一组断言插装到在 HAPS 等价的 C 程序里的每条语句后,这样会造成验证规模的极大增加,因此,需要针对活性验证算法进行化简.如果当一条 HAPS 的变迁并不改变φ中命题变元的值时,此时我们不需要在这条变迁后插入这组断言去检测断言的情况,因为该组断言变迁前后没有变化,我们定义这种情况为该变迁关系对于 LTL 公式是不可见的.

```
01 float physical, nu, t;
02 float physcial_dynamic1(float Time);
03 void physical_run(void)
04 {
05       if (system_mode1==1):{
06            if((t_s<=t_clock) && (t_clock<t_i)){
07                 t=make_symbolic();
08                 __CPROVER_assume((t_s<=t) && (t<t_i));
09  //physical dynamic in system mode 1
10                 y=physcial_dynamic1(t);
11                //get senor value
12                 ad_convertion();
13          ...}else if((t_i<=t_clock) && (t_clock<t_o)){
14                        t=make_symbolic();
15                        __CPROVER_assume((t_i<=t) && (t<t_o));
16              y=physcial_dynamic1(t-t_i);
17                        ...}else if((t_o<=t_clock) && (t_clock<T)){
18                      t=make_symbolic();
19                              __CPROVER_assume((t_o<=t) && (t<T));
20               y=physcial_dynamic1(t-t_o);
21                              ...}
22     }else if(system_mode==0):{
23                 ...}
24 }
```

Fig.9 C program for TableSat plant

图 9 C 程序描述的 TableSat 受控对象

定义 9(可见命题). 给定一个 LTL 公式φ和系统模型 HAPS,当且仅当 $L_\varphi(c)=L_\varphi(c')$, 那么一个 HAPS 的变迁关系 $t=c\xrightarrow{}_{\text{HAPS}}c'$ 对于φ是不可见的;否则,t 对于φ就是可见的.

$L_\varphi(c)$ 表示 c 中的有关φ中命题变元的值,如果 $L_\varphi(c)=L_\varphi(c')$, 表示 HAPS 的变迁并没有改变$\varphi$中命题变元的值,此时该变迁关系对于 LTL 公式是不可见的.

定义 10(可见命题变元集合). *VisProp*(t)标记了被变迁关系 t 所影响的命题变元集合.如果 *VisProp*(t)=∅,t 就是不可见的.

定义 11(可见集 *VisibleSet*). $VisibleSet=\{head(\langle g',\omega\rangle)\mid \exists r=\langle(v,g),\tilde{x},\gamma\rangle\xrightarrow{}_{\text{HAPS}}\langle(v',g'),\tilde{x}',\omega\rangle\in\Delta'$对于$\varphi$是可见的}.

通过定义可见集 *VisibleSet*,我们只需要在程序的适当位置将一组断言插装到系统模型,根据断言的违反情况即可判断出性质的满足情况.因此,并不需要在将一组断言插装到系统模型每条语句后,这极大地化简了验证规模.

4.3 综合仿真和形式化验证的协同验证

安全攸关场景的正确性在 CPS 系统运行安全上起着非常重要的作用.一方面,由于 CPS 系统规模的不断增大,系统的复杂性有了很大的提高,直接对系统进行完全形式化验证已经变得越来越困难.另一方面,虽然仿真在工程实践中已经得到了广泛的应用,但是由于仿真测试的不完备性,仅能在一定输入的情况下运行系统来观测系统行为,无法穷尽地检测系统所有可能的输入与场景.因此,结合目前工程实践中广泛应用的仿真技术,对仿真辅助形式化验证在安全攸关系统设计中是非常必要的,尤其是针对具有 CPS 特征的复杂控制应用程序.

仿真测试方法使用具体数值来表现测试场景和预期结果,而模型检验技术使用系统模型来替代具体数值.系统模型描述了预期和非预期的系统行为.模型检验依赖严格的数学过程,搜索系统模型的可能执行路径以获

得反例.通过模型执行的形式化分析与仿真方法互补,使用的形式化验证方法可以发现特定动态运行情况是否会发生以及在什么条件下发生.这些信息可用于改进设计以及设计需求,用于辅助仿真调试和验证.

因此,本文综合仿真和形式化验证方法,提出了协同验证方法:通过仿真,驱动系统进入某一状态,然后结合形式化验证方法从当前仿真状态开始做 k 步有界模型检验.因此,基于系统仿真,我们可以选取系统运行过程中一个场景进行形式验证.

定义 12. CPS 系统行为迁移的条件记为 $r=\varphi\cup t$, φ表示事件集合,t 表示时钟集合.r 既可以表示为事件触发也可以表示时间触发.

因此,一个 CPS 系统的路径表示 $s_0,s_1,s_2,...,s_{k-1},s_k,...,s_n$ (其中, $0<k\leqslant n\in\mathbb{N}$),我们将这条路径记为

$$\pi=s_0\xrightarrow{r_0}s_1\xrightarrow{r_1}s_2\ldots\xrightarrow{r_{k-2}}s_{k-1}\xrightarrow{r_{k-1}}s_k\ldots\xrightarrow{r_{n-1}}s_n.$$

定义 13. CPS 系统行为迁移的条件序列表示为 $seq=r_0,r_1,r_2,...,r_n,n\in\mathbb{N},seq_i=r_i$ 表示从第 i 个迁移的条件,其中 $0\leqslant i\leqslant n$.

本文提出了一种将协同仿真和形式化验证相结合的办法,通过仿真执行和 k 步有界模型检验对关键场景进行验证.首先在 CPS 虚拟原型中通过仿真记录一条系统执行路径 π;然后在 CPS 系统的状态集合 $\{s_0,s_1,s_2,...,s_{k-1},s_k,...,s_n\}$ 中根据场景需求选择某些状态做有界模型检验.如图 10 所示,在状态 s_{k-1} 上,结合迁移的条件 seq_{k-2} 共执行了 4 步模型检验,其中, $S'_{k,1}$ 表示执行第 1 步模型检验时系统所处的状态.将 s_{k-1} 作为关键状态,则在状态 s_{k-1} 上,结合迁移的条件 seq_{k-1} 共执行了 4 步模型检验,其中, $s'_{k,1},s'_{k,2},s'_{k,3}$ 和 $s'_{k,4}$ 表示执行第 1 步、第 2 步、第 3 步和第 4 步模型检验时系统所处的状态.

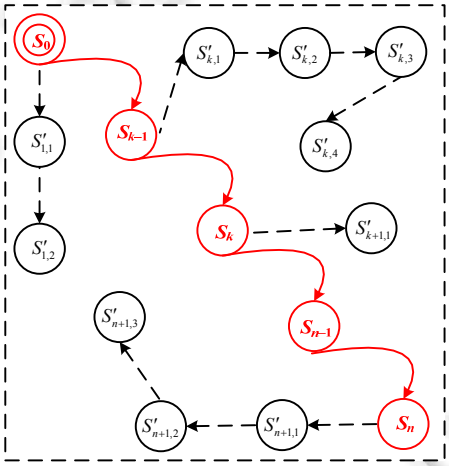


Fig.10 Semi-Formal execution using simulation and bounded model checking

图 10 综合仿真和有界模型检验的半形式化执行

5 协同验证工具原型

如图 11 所示,我们实现了 LTL 模型检验算法.协同验证工具原型 Co-Ver 基于 CBMC.CBMC 是一个针对 C 程序的模型检查器,它擅长检查程序的诸如检查数组边界、缓存溢出、指针安全等程序性质.

协同验证工具接受 3 类输入:(1) LTL 规约;(2) 控制应用 C 程序;(3) 受控对象模型.验证主要有 4 个步骤:

(1) 通过 LTL2BA 工具将 LTL 规约转换成 BA,然后再将 BA 转换为相应的 C 程序所描述的一组断言[26].

(2) 将受控对象(HA 模型)转换为相应的 C 程序,然后与控制应用程序(LPDS 模型)组装成统一验证模型 HAPS(C 程序描述).

(3) 将规约插装到模型 HAPS 的适当位置.

(4) 执行有界模型检验.

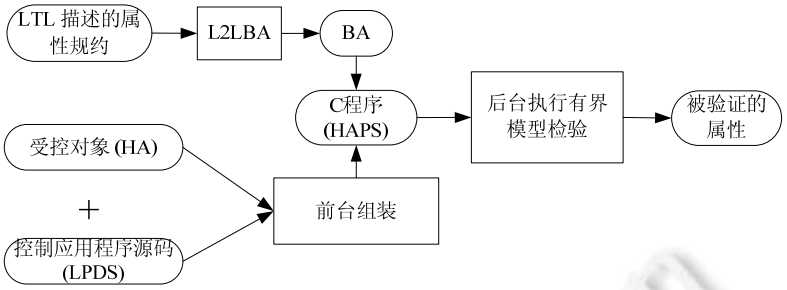


Fig.11 Implementation for co-verification

图 11 协同验证工具实现

通过不确定函数 non-determinism()描述系统中的不确定行为,如采样的延迟.通过假设 assume 来限制系统可能造成的非确定性,并用断言 assert 将要验证的属性的声明添加到设计的内部状态之中.因此,结合假设 assume、断言 assert 和不确定函数 non-determinism()可以构造丰富的场景.

使用模型检验可以是对仿真的一个补充,它能够将仿真结果用于形式化方法进行模型分析.传统仿真测试方法使用具体数值来表现测试场景和预期结果,而模型检验技术使用系统行为模型来替代具体数值.系统行为模型可以包括模型测试方案和验证目标,描述预期和非预期的系统行为.通过行为模型执行的形式化分析与仿真方法互补,使用的形式化方法可以发现特定动态运行情况是否会发生以及在什么条件下发生.这些信息可用于改进设计以及设计需求,用于辅助仿真调试和验证.协同仿真和模型检验综合验证框架如图 12 所示.

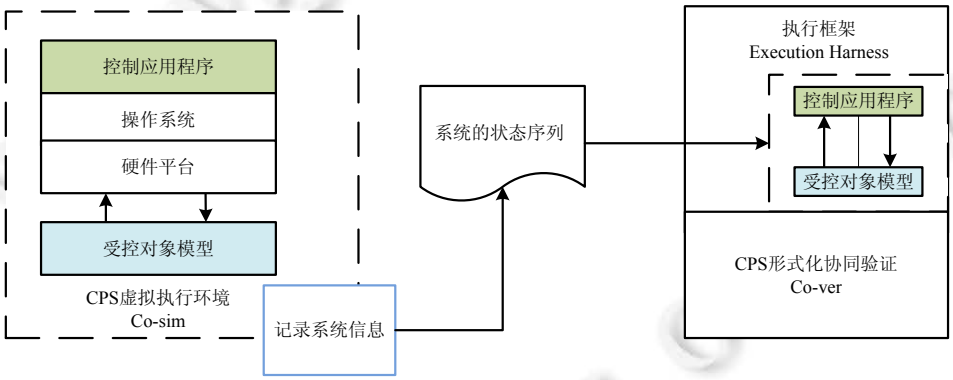


Fig.12 Combine simulation with bounded model checking

图 12 综合仿真和有界模型检验

本文提出了针对实现层的 CPS 协同验证技术方法.基于所设计的虚拟执行平台 Co-Sim[27]对嵌入式系统进行协同仿真:使用 QEMU 和 Matlab 模拟仿真物理实体连续动态变化、计算实体离散动态变化及其运行时交互过程.首先通过协同仿真,在虚拟执行环境中执行测试用例并记录系统信息,然后将仿真数据导出成系统的状态序列,最后在形式化验证工具中执行模型验证并输出验证结果.其中,执行框架 Execution harness 是一个包含了系统验证模型和仿真数据(这些数据已经过配置)的集合,用以验证一个系统,使其在不同的条件下运行,并监控其行为和输出.执行框架有 3 个主要的组成部分:执行引擎、模型和仿真数据库(repository).验证如算法 2 所示.

算法 2. 混合执行路径验证算法.

输入:仿真路径集 $s_n \in ST$, 待验证状态集 $ws \in WS$, 控制周期 T,A/D 转换完成时刻 t_i^k, D/A 转换完成时刻 t_o^k,

截止时间 *Time_Bound*.

输出:验证结果 Result.

Forall the $i<num$ **do**

$i\leftarrow 0$; //loop iteration

$s_n \leftarrow Load\_Simulation\_Trace(ST)$;

$num \leftarrow number\_of\_state\_in\_ws$;

$s_i \leftarrow Get\_State(ws,i)$;

$s_{i,0} \leftarrow Reset\_State(s_i)$;

while $t < Time\_Bound$ **do**

$s_{i,k+1} \leftarrow Compute\_Next\_state(s_{i,k})$;

$\mathrm{Re}sult \leftarrow Check\_State(s_{i,k+1})$

$k \leftarrow k+1$;

end while

end for

6 实验结果与分析

为了验证本文所提出的方法,我们做了两组实验:一是混成系统的经典例子——温控自动控制系统;二是一个实际的嵌入式自动控制系统 TableSat.实验平台:3.40GHz Intel(R) Core(TM),16G memory,Ubuntu 12.04 64 位操作系统.

6.1 温度自动控制系统

温度自动控制系统:利用控制器保证某一特定空间温度达到期望值的系统,是一个经典的混成系统案例,我们通过改造成为一个嵌入式控制系统应用场景.在温控系统中,环境温度是连续变化的物理过程,温控器是离散的计算过程:当控制器检测到系统温度降低到 19°C 以下时,控制应用就发出控制指令 on,从而打开加热器,对系统进行加热;而当系统温度高于 21°C 时,则正好相反,控制应用发出控制指令 off,从而关闭加热器.温度自动控制系统要确保环境温度在 18°C~22°C 之间.

1) 系统属性

首先使用 LTL 描述系统属性,然后通过工具形成相应的 C 程序段.

规则 1. 控制器永远不会将温度加热超过一个上限,LTL 形式化表示为 $\boldsymbol{G}(Temper \leqslant TemperUpLimit)$.

规则 2. 温度不会低于一个下限,LTL 形式化表示为 $\boldsymbol{G}(Temper \geqslant TemperDownLimit)$.

规则 3. 当温度超过上界值时,控制器总会发出 off 指令,LTL 形式化表示为 $\boldsymbol{G}((Temper \geqslant up\_th) \rightarrow \boldsymbol{F}(\mathrm{Off})$.

规则 4. 当温度低于下界值时,控制器总会发出 On 指令,LTL 形式化表示为 $\boldsymbol{G}((Temper < down\_th) \rightarrow \boldsymbol{F}(\mathrm{On})$.

然后将描述 LTL 的 C 程序段插装到程序的适当位置,执行有界模型检验.

2) 形式化验证

(1) 目前的验证方法一般不对控制周期进行细分,尤其是控制领域,通常假设计算不消耗时间,即在采样、计算和控制是在同一时刻完成,因此第 1 个实验场景也基于这个思想设置.实验参数:温度初始值不确定,是范围在[19,24]之间的实数值;温控器处于关闭状态;系统的控制时间间隔 t=0.3s,A/D 转换完成的时刻 t_i=0.0s,D/A 转换完成的时刻 t_o=0.0s,即从 A/D 转换到 D/A 转换的时间间隔为(d=0s).

图 13 是受控对象温度在这个场景下的模型图.在该模型中,连续变量 t 表示时间,连续变量 x 描述的是系统中实时变化的温度数值.离散位置 On_AD,On_DA 表示加热器被打开(on)时,环境温度分别处于采样和计算控制状态.离散位置 Off_AD,Off_DA 表示在加热器被关闭(off)时,环境温度分别处于采样和计算控制状态.当控制指令 Off 到达时,加热器被关闭,环境中的温度按照物理规律 $\dot{x} = -2$ 下降;当控制指令 On 到达时时,加热器被打开,环境中的温度按照物理规律 $\dot{x} = 4$ 上升.系统的初始条件被设定为温度 20°C.

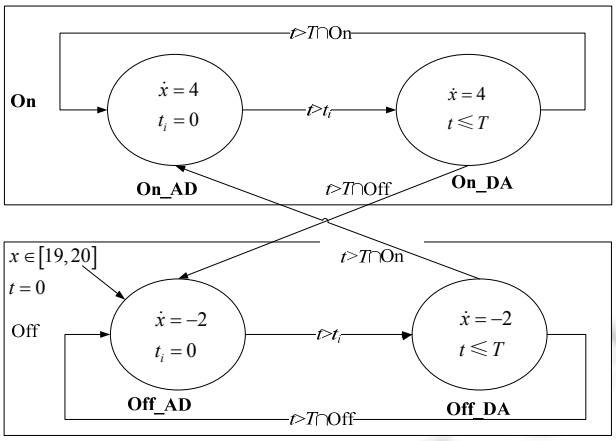


Fig.13 Plant model 1 for temperature control system

图 13 温控系统的受控对象模型 1

然后针对这个场景对系统属性 1~属性 4 执行有界模型检验,实验结果见表 1.

Table 1 Experimental result 1

表 1 实验结果 1

规则序号	控制循环次数 K	时间(s)	结果
1	1	0.27	T^p
	10	3.078	T^p
	15	5.806	$\perp$
2,3,4	1	0.27	T^p
	20	7.764	T^p
	40	30.732	T^p
	60	139.231	T^p
	80	380.052	T^p
	100	696.585	T^p
	120	826.031	T^p
	150	2 015.53	T^p

实验结果显示,只有规则 1 不满足,其余规则都满足.规则 1 不满足的原因在于,当时环境温度接近上限(22°C)时,温控器仍然打开加热器,使得环境温度达到 22.031 76°C,超过了上界.另外,有界模型检测的效率与循环和递归的展开有密切的关系,因为(1) 循环和递归展开会使代码量激增;(2) 代码的增加会导致变量复制的增加.通过实验结果分析,验证的时间开销主要来自于求解器,当控制循环次数达到 150 时,验证需要 2 015.53s.

(2) 为了与第 1 个实验作对比,我们做了第 2 个实验.在第 2 个实验中考虑了采样控制时间的影响:系统的控制时间间隔 T=0.3s,A/D 转换完成的时刻 t_i=0.0s,D/A 转换完成的时刻 t_o=0.2s,即从 A/D 转换到 D/A 转换的时间间隔为(d=0.2s),由于采样延迟抖动($j_s\in[0,0.1]$)和输入/输出抖动($j_{io}\in[0,0.05]$)不确定,利用不确定函数 non-determinism()设置为符号量.其他实验参数:温度初始值为 20,温控器处于关闭状态.图 14 是受控对象温度在这个场景下的模型图.图 14 与图 13 所描述的模型类似,增加了两个离散位置 On_Rest 和 Off_Rest,分别表示加热器被打开(on)时和关闭(off)时的控制后状态.系统的初始条件被设定为温度 20°C.

然后针对这个场景对系统属性 1~属性 4 执行有界模型检验,实验结果见表 2.实验结果显示,所有规则都满足.规则 1 在第 2 个实验中满足的原因在于,当环境温度接近上限时,在控制时刻上温控器会更早关闭加热器.通过细致分析可以发现,第 2 个实验场景更符合实际情况,计算不消耗时间的假设过于理想.另外,在本实验中对比了算法优化的效果,本例中的 Timeout 时间设为 3 600s.通过实验结果可以清楚地看出,算法的优化效果很明显,时间开销平均降低了 82%.

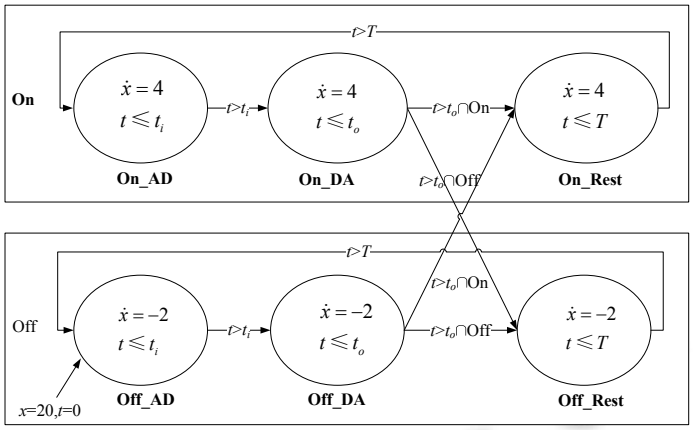


Fig.14 Plant model 2 for temperature control system

图 14 温控系统的受控对象模型 2

Table 2 Experimental result 2

表 2 实验结果 2

规则序号	K	无优化		优化后	
		时间(s)	结果	时间(s)	结果
1,2,3,4	1	0.947	T^p	0.222	T^p
	10	27.805	T^p	7.7	T^p
	20	615.436	T^p	16.813	T^p
	30	Timeout	N/A	39.532	T^p
	40	Timeout	N/A	57.01	T^p
	50	Timeout	N/A	73.4	T^p
	80	Timeout	N/A	275.921	T^p

6.2 TableSat

人造卫星姿态控制模拟环境 TableSat,如图 15 所示,是一个模拟卫星姿态控制的自由度实验平台.NASA 使用这套实验平台模拟人造卫星姿态的变化、采样和控制过程.我们通过与 University of Michigan 的合作在实验室搭建了这套实验平台.本文将主要通过 TableSat 作为人造卫星姿态控制应用场景的实验平台.

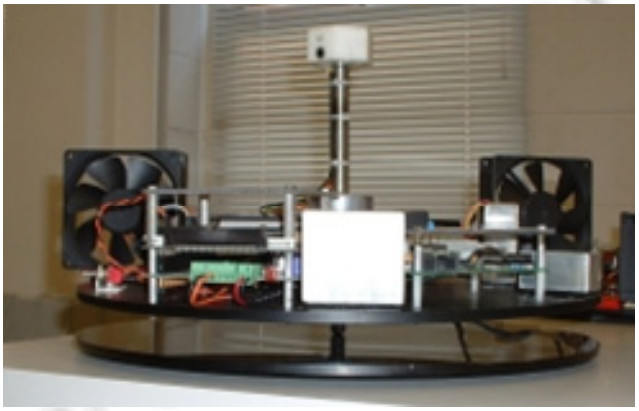


Fig.15 TableSat

图 15 TableSat

TableSat 模拟了卫星的姿态控制过程.PC/104 通过 16 位的模拟数字转换器获得角速度传感器所测的转盘角速度值,根据控制算法计算所应施加的给电脑风扇的电压值,然后通过 16 位的数字模拟转换器完成电脑风扇电压的控制,并且通过 802.11b 无线接口与实验机(模拟地面卫星接收站,ground station)通信.用户设计实现转盘角速度的控制算法用来控制转盘的转速,使得转盘的转速维持在预定目标值.TableSat 在运行期间,实时并发计算、环境感知以及自主控制依据严格的时间约束在持续地融合,并与物理环境不断交互,共同确定下一时刻的行为.

在人造卫星控制问题中,从简单的姿态控制到复杂的交会问题,大多数情况都采用燃料燃烧所产生的推力或力矩进行控制,为了节省昂贵的燃料,工程上提出燃耗最少的控制问题.单纯以节省燃料为目的的控制往往导致系统的响应过慢,因而又出现了兼顾响应时间及燃料的时间一燃料最优控制问题.为了更好地解决以上问题,人们提出了砰-砰(Bang-Bang)控制,它是卫星系统中不可缺少的控制方式.所以本实验以一个砰-砰控制应用为例.

(1) 物理过程建模

连续系统一般会用微分方程来表示.TableSat 方程线性处理后为

$$\begin{cases}\dot{\omega}=\dfrac{lK_{\omega f}}{I}v\\ \dot{v}=-\alpha v+K_{v\dot{\omega}}V\end{cases} \tag{1}$$

其中,I 是 TableSat 的转动惯量,ω是 TableSat 的角速度,l 是风扇的力臂,$K_{\omega f}$ 是风扇与动力的比例常数,υ是风扇的转速,α是风扇常数,$K_{v\dot{\omega}}$ 是风扇电压与转速的比例常数,V 是风扇所施加的电压值.

(2) 协同验证

首先使用 LTL 描述系统属性,然后通过工具生成相应的 C 程序段.

规则 1. 控制器永远不会将 TableSat 加速超过一个速度上限,LTL 形式化表示为

$$\boldsymbol{G}(\ Rotary.Velocity \leqslant VelocityUpBound\ ).$$

规则 2. 在一定时间内,控制器必须将 TableSat 加速超过一个速度下限,LTL 形式化表示为

$$\boldsymbol{G}(\ (Time \geqslant TimeBound) \rightarrow (Rotary.Velocity \geqslant VelocityDownBound)\ ).$$

规则 3. 当 TableSat 的转盘角速度超过期望值的 1.5 倍后,控制器将风扇的电压置为 0V,LTL 形式化表示为

$$\boldsymbol{G}(\ (Rotary.Velocity > 1.5\times TargetVelocity) \rightarrow (Actuator.FanVoltage \equiv 0)\ ).$$

规则 4. 当 TableSat 的转盘角速度低于期望值的 0.4 倍后,控制器将风扇的电压置为 12V,LTL 形式化表示为

$$\boldsymbol{G}(\ (Rotary.Velocity < 0.4\times TargetVelocity) \rightarrow (Actuator.FanVoltage \equiv 12)\ ).$$

规则 5. 在一定时间内,TableSat 的转盘角速度与期望值的差的绝对值必须在误差允许范围内,LTL 形式化表示为

$$\boldsymbol{G}(\ (Time \geqslant TimeBound) \rightarrow (\left|Rotary.Velocity - TargetVelocity\right| \leqslant SteadyStateError)\ ).$$

规则 6. 当 TableSat 的转盘角速度大于期望值时,控制器总会将风扇的电压置为 0V,LTL 形式化表示为

$$\boldsymbol{G}(\ (Rotary.Velocity > TargetVelocity) \rightarrow \boldsymbol{F}\ (Actuator.FanVoltage = fullNeg)\ ).$$

规则 7. 当 TableSat 的转盘角速度小于期望值时,控制器总会将风扇的电压置为 12V,LTL 形式化表示为

$$\boldsymbol{G}(\ (Rotary.Velocity < TargetVelocity) \rightarrow \boldsymbol{F}\ (Actuator.FanVoltage = full)\ ).$$

然后编写 Bang-Bang 控制器的 C 程序,接着将该程序和受控对象组装成统一验证模型,最终形成一个描述系统的完整 C 程序(程序行数为 644 行).最后将描述 LTL 的 C 程序段插装到程序的适当位置.已知受控对象的状态方程为 $\dot{x}=Ax(t)+Bu(t)\Rightarrow x[k+1]=A_d x[k]+B_d u[k]$. 这里主要做了两组实验,一是直接对系统进行模型检验;二是结合协同仿真,对关键场景辅助模型检验.下面详细加以介绍.

(1) 直接应用模型验证

从某种意义上说,系统控制的启动瞬间响应是系统危险的时段.针对系统的初始状态不确定的情况下设计验证场景:转盘角速度初始值不确定,是范围在[0,40]之间的实数值;风扇的初始电压为 12V,风扇转速为 0;系统

的控制时间间隔 T=0.4s,A/D 转换完成的时刻 t_i=0.0s,D/A 转换完成的时刻 t_o=0.0s.角速度期望值为 30deg/s,我们将 TableSat 的转盘角速度初始值利用不确定函数 non-determinism()设为一定范围内的符号量 $\omega\in[0,40]$,依据场景设定其他参数.

实验结果见表 3.在验证中,无优化算法发现规则 1 违背,路径是:当初速度为 40.0deg/s,在 0.323 9197s 时,转速达到 45.080 17deg/s,此时风扇转速为 13 043.52deg/s.优化后算法发现规则 1 违背,路径是:当初速度为 37.999 95deg/s,在 0.394s 时,转速达到 45.515 26deg/s,此时风扇转速为 13 463.87deg/s.

Table 3 Experimental result 3

表 3 实验结果 3

规则序号	K	无优化			优化后		
		变量数	时间(s)	结果	变量数	时间(s)	结果
1	1	953 323	384.676	⊥	91 981	14.615	⊥
	2	N/A	N/A	N/A	N/A	N/A	N/A
	3	N/A	N/A	N/A	N/A	N/A	N/A
	4	N/A	N/A	N/A	N/A	N/A	N/A
2,3,4,5,6,7	1	953 323	384.676	N/A	91 981	14.615	T^p
	2	N/A	Timeout	N/A	308 452	164.288	T^p
	3	N/A	Timeout	N/A	526 190	713.443	T^p
	4	N/A	Timeout	N/A	742 550	2 396.97	T^p

通过实验结果分析,验证的时间开销主要来自于求解器,相对于温控系统,一方面由于受控对象是一个二元二次方程组,另一方面由于本例的规模以及大量浮点数的计算,验证的代价比较大,通过算法优化,验证的耗时显著提高.

(2) 结合仿真执行模型验证

在第 1 组实验中,直接应用模型检验,虽然经过优化,可以执行有限步,但是由于系统的复杂性,执行的步骤有限,所以目前无法完全直接应用模型检验.在实际的开发中,仿真可以有效地分析系统的控制性能.通过执行典型的测试用例可以获得的仿真路径,然后基于经验在某些容易出现问题的场景,结合仿真的数据作为初态,用模型检验辅助验证场景.

仿真测试场景:在时间段[0s,40s)时,控制目标值为 30deg/s;在时间段[40s,80s)时,控制目标值为 50deg/s;在时间段[80s,120s)时,控制目标值为 70deg/s;在时间段[120s,160s)时,控制目标值为 20deg/s;在时间段[160s,200s)时,控制目标值为 50deg/s;在时间段[200s,240s)时,控制目标值为 30deg/s.测试采用固定周期(T=0.4s),A/D 转换完成时刻为($t_i^k\in[0,0.1]$),D/A 转换完成时刻为($t_o^k\in[0,0.1]$).我们每 0.01s 记录一次测量值,仿真结果如图 16 所示.

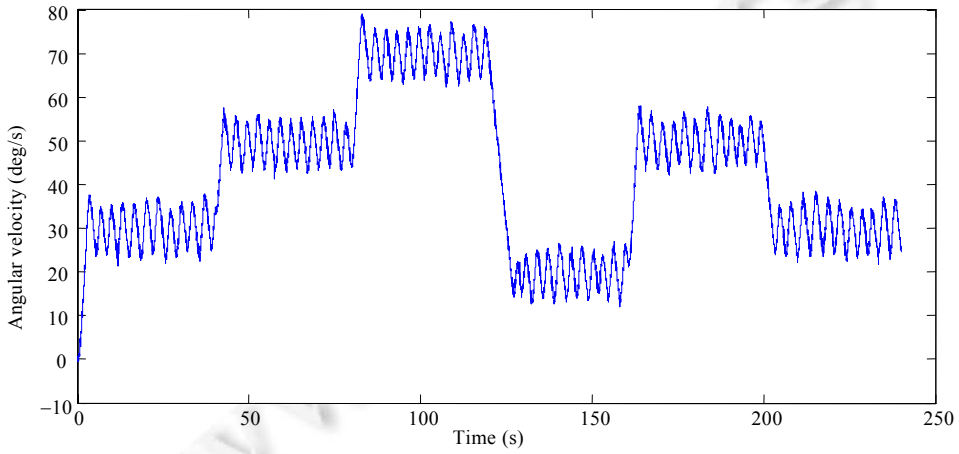


Fig.16 Simulation result

图 16 仿真结果

在仿真中,延迟是由某个区间上的随机函数产生,在每次仿真中是某一常值,因此测试是不完备的.在系统初始状态为0时,当系统采样抖动J_s和输入/输出j_{io}不确定的情况下,测试控制器能否正确地控制受控对象.结合仿真数据,我们将7条系统性质应用于模型验证中,在这条仿真路径上共选取了8个状态点,在每个状态点执行3步.下面主要介绍其中两个场景的实验.

场景 1. 当系统阶跃响应曲线第1次越过稳态值而达到峰点时记录到达峰点时刻系统的状态,然后基于这一状态执行有界模型检验.由于系统中部件具有惯性,故系统在外作用下由一个平衡状态(或稳态)过渡到另一个平衡状态(或稳态)需要有一个过程,这一过程称为瞬态过程.在瞬态过程中,系统阶跃响应曲线第 1 次越过稳态值而达到峰点时,系统的采样抖动J_s和输入/输出j_{io}不确定的情况下,测试控制器能否正确地控制受控对象.

根据仿真数据,在t=3.5s时刻,系统阶跃响应曲线第1次越过稳态值而达到峰点(ω=36.7298deg/s),记录此时系统的状态.将采样延迟抖动($j_s\in[0,0.1]$)和输入/输出抖动($j_{io}\in[0,0.1]$)利用不确定函数 non-determinism()设为符号量.实验结果见表4.

Table 4 Experimental result 4

表 4 实验结果 4

规则序号	K=1		K=2		K=3	
	时间(s)	结果	时间(s)	结果	时间(s)	结果
1	271.57	T^p	1 746.37	T^p	28 422	T^p
2	271.57	T^p	1 746.37	T^p	28 422	T^p
3	271.57	T^p	1 746.37	T^p	28 422	T^p
4	271.57	T^p	1 746.37	T^p	28 422	T^p
5	271.57	T^p	1 746.37	T^p	28 422	T^p
6	271.57	T^p	1 746.37	T^p	28 422	T^p
7	271.57	T^p	1 746.37	T^p	28 422	T^p

场景 2. 当控制目标值从20deg/s切换到50deg/s时,系统的采样抖动J_s和输入/输出j_{io}不确定的情况下,测试控制器能否正确地控制受控对象.根据仿真数据,在 t=160s 时刻,ω=25.0607deg/s,风扇转速:1788.54,记录此时系统的状态.将采样延迟抖动($j_s\in[0,0.1]$)和输入/输出抖动($j_{io}\in[0,0.1]$)利用不确定函数 non-determinism()设为符号量.实验结果见表5.

从上述的实验结果可以看出,随着系统规模的不断增大,直接对实现级 CPS 系统进行形式化验证已经变得越来越困难.在本例中,如果直接对 TableSat 进行模型检验,时间开销很大,基本只能检查系统从初始开始 3 步内的行为.但是结合仿真,我们可以在整条仿真路径上对某些关键场景辅助模型检验,在该实验中,每个场景能够检查 3 步内的行为,总计可以检查 8 个关键场景共 24 步状态空间的行为.可以利用 Co-Sim 对不同的场景以及场景转换的组合情况进行仿真,然后再将仿真结果用形式化方法进行验证.Co-ver 提供了 LTL 属性模型检验,在指定的时间内,Co-ver 会验证是否存在能触发断言的情况.例如,可以构造一些断言,每当 TableSat 加速超过一个速度上限时触发.当设定的断言被违反时,它会报错而且也会生成触发这个断言的反例.

7 结束语

与一般嵌入式系统相比,在CPS概念兴起之后,现在面向CPS的嵌入式系统更加强调协同,通过实时采集环境中或者其他协作成员的数据,从而更加智能地做出控制指令.在此类系统中,大规模复杂异构性、延迟不确定性、精确控制以及计算过程同步或异步等需求导致非功能属性的表示和验证更加复杂.

本文基于自动机理论提出了控制环路的验证模型,针对组合状态空间爆炸问题,一方面在求解的过程中通过偏序规约等手段尽量化简模型规模,另一方面结合虚拟仿真提出了综合虚拟仿真和形式化验证相结合的方法,灵活配置验证的场景,提高验证的可用性.针对目前属性规约描述表达能力不强,如无法表达时序、活性等,本文基于 LTL 描述属性的规约,通过定义在物理连续变量上的布尔表达式描述物理过程,而后将 LTL 性质规约公式转换为 Büchi 自动机,即将性质分解为与其等价的一组断言,然后再将这组断言插装到控制应用程序程序的适当位置,根据断言的违反情况即可判断出性质的满足情况.运行时验证考虑控制应用这样的非终止程序,在

任何时刻只能获得当前执行的一个有穷状态前缀,针对有穷路径基于四值语义进行判断.

Table 5 Experimental result 5

表 5 实验结果 5

规则序号	*K*=1		*K*=2		*K*=3	
	时间(s)	结果	时间(s)	结果	时间(s)	结果
1	315.027	T^p	1 841.2	T^p	27 056	T^p
2	315.027	T^p	1 841.2	T^p	27 056	T^p
3	315.027	T^p	1 841.2	T^p	27 056	T^p
4	315.027	T^p	1 841.2	T^p	27 056	T^p
5	315.027	T^p	1 841.2	T^p	27 056	T^p
6	315.027	T^p	1 841.2	T^p	27 056	T^p
7	315.027	T^p	1 841.2	T^p	27 056	T^p

实验结果表明,形式化协同验证可以有效地对系统进行验证,但是由于实际系统的规模很难直接应用,结合协同仿真,可以有效地对系统进行辅助验证.由实验结果分析可知,本文方法所消耗的时间较多.因此,研究求解器里的约简技术,将作为本文的下一步工作.

References:

[1] Lee EA. CPS foundations. In: Proc. of the 47th Design Automation Conf. (DAC). ACM, 2010. 737−742. [doi: 10.1145/1837274.1837462]

[2] Clarke E, Kroening D, Lerda F. A tool for checking ANSI-C programs. In: Jensen K, Podelski A, eds. Tools and Algorithms for the Construction and Analysis of Systems. LNCS 2988, Berlin, Heidelberg: Springer-Verlag, 2004. 168−176. [doi: 10.1007/978-3-540-24730-2_15]

[3] Beyer D, Keremoglu M, CPAchecker: A tool for configurable software verification. In: Gopalakrishnan G, Qadeer S, eds. Computer Aided Verification. LNCS 6806, Berlin, Heidelberg: Springer-Verlag, 2011. 184−190. [doi: 10.1007/978-3-642-22110-1_16]

[4] Ball T, Bounimova E, Kumar R, Levin V. SLAM2: Static driver verification with under 4% false alarms. In: Proc. of the 10th Int'l Conf. on Formal Methods in Computer-Aided Design, FMCAD 2010. Lugano, 2010. 35−42.

[5] Henzinger T, Jhala R, Majumdar R, Sutre G. Software verification with BLAST. In: Model Checking Software. LNCS 2648, Berlin, Heidelberg: Springer-Verlag, 2003. 235−239. [doi: 10.1007/3-540-44829-2_17]

[6] Havelund K, Pressburger T. Model checking Java programs using java pathfinder. Int'l Journal on Software Tools for Technology Transfer, 2000,2(4):366−381. [doi: 10.1007/s100090050043]

[7] Holzmann G. The model checker spin. IEEE Trans. on Software Engineering, 1997,23(5):279−295. [doi: 10.1109/32.588521]

[8] Herrmann P, Blech JO, Han FL, Schmidt H. A model-based toolchain to verify spatial behavior of cyber-physical systems. Int'l Journal of Web Services Research, 2016,13(1):40−52. [doi: 10.4018/IJWSR.2016010103]

[9] Shan LJ, Zhou XS, Wang YY, Zhao L, Wan LJ, Qiao L, Chen JX. Statistical model checking of cyber-physical systems control software. Ruan Jian Xue Bao/Journal of Software, 2015,26(2):380−389 (in Chinese with English abstract). http//www.jos.org.cn/9825-1000/4788.html [doi: 10.13328/j.cnki.jos.004788]

[10] Chan M, Ricketts D, Lerner S, Malecha G. Formal verification of stability properties of cyber-physical systems. 2016. http://www.chargueraud.org/events/coqpl2016/CoqPL_2016_paper_5.pdf

[11] Bae K, Krisiloff J, Meseguer J, Ölveczky PC. Designing and verifying distributed cyber-physical systems using multirate PALS: An airplane turning control system case study. Science of Computer Programming, 2015,103:13−50. [doi: 10.1016/j.scico.2014.09.011]

[12] Majumdar R, Saha I. Symbolic robustness analysis. In: Proc. of the 30th IEEE of Real-Time Systems Symp (RTSS 2009). IEEE, 2009. 355−363. [doi: 10.1109/RTSS.2009.17]

[13] Anta A, Majumdar R, Saha I, Tabuada P. Automatic verification of control system implementations. In: Proc. of the EMSOFT. 2010. [doi: 10.1145/1879021.1879024]

[14] Majumdar R, Saha, Yazarel. Compositional equivalence checking for models and code of control systems. In: Proc. of 2013 IEEE the 52nd Annual Conf. on Decision and Control (CDC). 2013. 1564−1571. [doi: 10.1109/CDC.2013.6760105]

[15] Lerda F, Kapinski J, Maka H, Clarke EM, Krogh BH. Model checking in-the-loop: Finding counterexamples by systematic simulation. In: Proc. of the American Control Conf. IEEE, 2008. 2734−2740. [doi: 10.1109/ACC.2008.4586906]

[16] Pasareanu C, Schumann J, Mehlitz P, Lowry M, Karsai G, Nine H, Neema S. Model based analysis and test generation for flight software. In: Proc. of the 3rd IEEE Int'l Conf. on Space Mission Challenges for Information Technology (SMC-IT 2009). 2009. 83−90. [doi: 10.1109/SMC-IT.2009.18]

[17] Bouissou O, Goubault E, Putot S, Tekkal K, Vedrine F. Hybridfluctuat: A static analyzer of numerical programs within a continuous environment. In: Bouajjani A, Maler O, eds Computer Aided Verification. LNCS 5643, Berlin, Heidelberg: Springer-Verlag, 2009. 620−626. [doi: 10.1007/978-3-642-02658-4_46]

[18] Majumdar R, Saha I, Shashidhar K, Wang Z. CLSE: Closed-Loop symbolic execution. In: Goodloe A, Person S, eds. NASA Formal Methods. LNCS 7226, Berlin, Heidelberg: Springer-Verlag, 2012. 356−370. [doi: 10.1007/978-3-642-28891-3_33]

[19] Zhang Y, Xie F, Dong YW, Zhou XS, Ma CY. Cyber/Physical co-verification for developing reliable cyber-physical systems. In: Proc. of 2013 IEEE the 37th Annual Computer Software and Applications Conf. IEEE Computer Society, 2013. 539−548. [doi: 10.1109/COMPSAC.2013.88]

[20] Biere A, Cimatti A, Clarke EM, Strichman O, Zhu YS. Bounded model checking. Advances in Computers, 2003,58(3):117−148. [doi: 10.1016/ S0065-2458(03)58003-2]

[21] Pnueli A. The temporal logic of programs. In: Proc. of the 18th Annual Symp. on Foundations of Computer Science. 1977. 46−57. [doi: 10.1109/SFCS.1977.32]

[22] Bauer A, Leucker M, Schallhart C. Comparing LTL semantics for runtime verification. Journal of Logic and Computation, 2010,20(3): 651−674. [doi: 10.1093/logcom/exn075]

[23] Årźen K, Cervin A, Henriksson D. Implementation aware embedded control systems. In: Handbook of Networked and Embedded Control Systems. 2005. 377−394. [doi: 10.1007/0-8176-4404-0_16]

[24] Schwoon S. Model-Checking pushdown systems [Ph.D. Thesis]. Technische Universität München, Institutfür Infomatik, 2002.

[25] Henzinger TA. The theory of hybrid automata. In: Proc. of the Symp. on Logic in Computer Science. 1996. 278−292. [doi: 10.1109/ LICS.1996.561342]

[26] Morse J, Cordeiro L, Nicole D, Fischer B. Model checking LTL properties over ANSI-C programs with bounded traces. Software & Systems Modeling, 2015,14:65−81. [doi: 10.1007/s10270-013-0366-0]

[27] Zhang Y, Xie F, Dong YW, Yang G, Zhou XS. High fidelity virtualization of cyber-physical systems. Int'l Journal of Modeling, Simulation, and Scientific Computing, 2013,4(2):1340005. [doi: 10.1142/S1793962313400059]

附中文参考文献:

[9] 单黎君,周兴社,王宇英,赵雷,万丽景,乔磊,陈建新.信息物理融合系统控制软件的统计模型检验.软件学报,2015,26(2):380−389. http//www.jos.org.cn/9825-1000/4788.html [doi: 10.13328/j.cnki.jos.004788]

张雨(1983－),男,河南济源人,博士,讲师,CCF 专业会员,主要研究领域为嵌入式系统的协同设计与验证.

冯文龙(1968－),男,博士,教授,CCF 专业会员,主要研究领域为大数据与智慧服务.

董云卫(1968－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为嵌入式系统,信息物理融合系统,可信软件设计与验证.

黄梦醒(1973－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为云计算,信息系统技术.