软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(6):1670−1680 [doi: 10.13328/j.cnki.jos.005472]

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

一种嵌套中断系统的建模和分析方法*

崔 进[1,2], 段振华[1,2], 田 聪[1,2], 张 南[1,2]

[1](ISN 国家重点实验室(西安电子科技大学),陕西 西安 710071)

[2](西安电子科技大学 计算理论与技术研究所,陕西 西安 710071)

通讯作者: 段振华, 田聪, Email: {zhhduan,ctian}@mail.xidian.edu.cn

摘 要: 在嵌入式系统和各类操作系统中,中断机制是确保实时响应各类异步事件的重要方法.通常在处理一个中断事件的过程中,往往会有更紧迫的中断事件请求响应,因而发生中断嵌套.建模并验证嵌套中断系统是具有挑战性的工作.提出一种建模和验证嵌套中断系统的方法.首先,提出基于投影时序逻辑(projection temporal logic,简称 PTL)的定义,并将这种定义推广到包含任意多中断事件的中断系统上,从而得出嵌套中断系统基于投影时序逻辑的形式化模型;其次,使用投影时序逻辑定义的基本中断语句扩充建模仿真和验证语言(modeling, simulation and verification language,简称 MSVL),并扩展 MSVL 语言的解释器,使其可以对嵌套中断系统进行建模仿真和验证;最后,通过一个实例展现所提出方法的正确性和实用性.

关键词: 嵌套中断系统;投影时序逻辑;MSVL(modeling, simulation and verification language);形式化建模与验证

中图法分类号: TP311

中文引用格式: 崔进,段振华,田聪,张南.一种嵌套中断系统的建模和分析方法.软件学报,2018,29(6):1670−1680. http://www.jos.org.cn/1000-9825/5472.htm

英文引用格式: Cui J, Duan ZH, Tian C, Zhang N. Modeling and analysis of nested interrupt systems. Ruan Jian Xue Bao/Journal of Software, 2018,29(6):1670−1680 (in Chinese). http://www.jos.org.cn/1000-9825/5472.htm

Modeling and Analysis of Nested Interrupt Systems

CUI Jin[1,2], DUAN Zhen-Hua[1,2], TIAN Cong[1,2], ZHANG Nan[1,2]

[1](ISN State Key Laboratory (Xidian University), Xi'an 710071, China)

[2](Institute of Computing Theory and Technology, Xidian University, Xi'an 710071, China)

Abstract: The interrupt mechanism is a commonly used means to ensure real-time response to various asynchronous events in embedded systems and various real-time operating systems. Nested interrupts are likely to occur when the request of more urgent interrupt event arise while processing a less urgent one. Modeling and verifying such systems are challenging tasks. This paper proposes an approach to modeling and verifying systems with nested interrupts. First, MSVL (modeling, simulation and verification language) is extended to model such systems by defining an interrupt statement using projection temporal logic (PTL). Then, methods of modeling different nested interrupt systems using the interrupt statement are studied. Next, the MSV toolkit is extended to model, simulate, and verify MSVL programs with nested interrupts automatically. Finally, a case study is given to demonstrate how the proposed method works.

Key words: nested interrupt systems; projection temporal logic; MSVL (modeling, simulation and verification language); formal modeling and verification

* 基金项目: 国家自然科学基金(61420106004, 61732013, 61572386)
Foundation item: National Natural Science Foundation of China (61420106004, 61732013, 61572386)
本文由形式化方法的理论基础专题特约编辑傅育熙教授、李国强副教授、田聪教授推荐.
收稿时间: 2017-07-06; 修改时间: 2017-09-01; 采用时间: 2017-11-06; jos 在线出版时间: 2017-12-28
CNKI 网络优先出版: 2017-12-29 13:19:35, http://kns.cnki.net/kcms/detail/11.2560.TP.20171229.1318.013.html

在嵌入式系统和各类操作系统中,中断机制是确保实时响应各类异步事件的常用方法.在处理一个中断的过程中,通常会出现更紧迫的中断事件请求响应,因而导致中断嵌套.中断事件的发生具有极大的不确定性,而且中断的响应与处理往往与硬件密切相关,因而,确保中断系统的可靠性和安全性是一项极具挑战性的工作.

形式化验证是一种确保软硬件系统可靠性和安全性的有效方法.定理证明和模型检测是形式化验证方法中的两种关键技术,主要用来判定一个系统是否具备某种性质.与定理证明相比,模型检测可以自动化进行.近年来,很多研究围绕形式化建模和验证中断系统展开.本文主要考虑嵌套中断系统,这是对文献[1]所做工作的扩展和推广.文献[2]利用多线程系统验证的方法来验证中断系统,由于中断缺乏形式化语义支持,所以使用该方法时,将中断的语义用等价的线程语义来形式化描述.然而,中断和线程之间的精细的差别使得用线程语义描述中断这个过程变得棘手.文献[3,4]验证μC/OS-II 操作系统中的嵌套中断系统,该方法借助霍尔逻辑[5]和分离逻辑[6]建模中断系统,使用定理证明工具 Coq[7]完成定理证明过程.文献[8]用时间 Petri 网建模嵌套中断系统,并将 Petri 网所建立的系统模型转换为时间自动机[9],从而借助时间自动机理论进行模型检测.文献[10]使用 iDola 语言建模嵌套中断系统,与文献[8]的方法类似,验证是通过把 iDola 语言程序转换为时间自动机完成的.文献[11]直接使用时间自动机建模嵌套中断系统.然而,在基于时间自动机的模型检测中,待验证的性质采用 LTL[12]表示,使得对于周期性性质和区间相关的性质不容易表达[13].本文是对已有嵌套中断系统验证工作中存在不足的补充,采用基于 MSVL(modeling, simulation and verification language)[14,15]统一模型检测的方法[13]来验证嵌套中断系统.该方法为包含任意多中断事件的中断系统提供了有效的形式化语义支持,从而可以直接建模并验证嵌套中断系统.

MSVL 是一种建模仿真和验证各类软硬件系统的语言,该语言使用投影时序逻辑(projection temporal logic,简称 PTL)[14,15]定义得到,是 PTL 的一个可执行子集.本文使用 PTL 定义了基本中断模型,并将该模型推广到建模包含任意多个中断事件的中断系统.接着,使用定义的中断模型扩充 MSVL 语言,同时扩展 MSVL 语言的模型检测工具[13,16],使其支持对中断语句的处理.这样,借助 MSVL 语言以及工具集 MSV[16],可以实现对包含任意多个中断事件的中断系统的建模仿真与验证.相比于常见的模型检测方法,基于 MSVL 的模型检测方法具有以下优势.

(1) 系统建模语言和性质描述语言基于相同的时序逻辑;

(2) 该方法中的性质描述语言有较强的表达能力,可以描述周期性性质和区间相关的性质[13].

本文第 1 节介绍 PTL 和可执行语言.第 2 节研究如何使用 PTL 定义包含任意数量中断事件的中断系统.第 3 节介绍如何扩展工具集 MSV,使其支持对中断系统的模型检测.第 4 节使用本文提出的方法建模和验证一个实例,给出建模和验证的过程以及结果分析.最后一节总结全文,并对未来研究提出进一步展望.

1 基础知识

本节主要介绍投影时序逻辑[14,15]和可执行语言[14,15].

1.1 投影时序逻辑

语法. 令 V 代表变量集合,Pr 代表可数的原子命题集合,D 为数据域,B={true,false}为布尔域.PTL 的项 e 和公式 ϕ 归纳定义如下:

$$e ::= v \mid \bigcirc e \mid \Theta e \mid f(e_1,...,e_m)$$

$$\phi ::= p \mid e_1 = e_2 \mid P(e_1,...,e_m) \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \exists v : \phi \mid \bigcirc\phi \mid (\phi_1,...,\phi_m)\ prj\ \phi$$

其中,$v \in V$,$\bigcirc$和Θ为时序操作符,f 为函数,$p \in Pr$,P 代表谓词,$(\phi_1,\ldots,\phi_m)\ prj\ \phi$为投影公式.当一个公式不包含时序操作符$\bigcirc$,$\Theta$,$prj$ 时,称其为状态公式,否则为时序公式.

语义. 状态 s 是一组映射:(I_v,I_p),其中,I_v:$V \to D'$,$D'=D \cup \{nil\}$.这里,nil 为无定义,I_p:$Pr \to B$.我们用 $s[v]$和 $s[p]$分别表示变量 v 和命题 p 在状态 s 的取值.区间$\sigma=\langle s_0,s_1,\ldots,s_{|\sigma|}\rangle$是一个非空的状态序列,其中,$s_0$ 为起始状态,若σ不为无穷大,$s_{|\sigma|}$为终止状态.$|\sigma|$代表σ的长度,它等于σ的状态个数减 1.我们用$\sigma_{(i,j)}(0 \leqslant i \leqslant j \leqslant |\sigma|)$表示区间$\sigma$上以 s_i 为

起始状态、s_j为终止状态的子区间.令 $r_1,\ldots,r_h(h\geqslant 1)$为整数且 $r_1\leqslant\ldots\leqslant r_h\leqslant|\sigma|$,$\sigma$在 $r_1,\ldots,r_h$ 上的投影构成了一个新区间,称为投影区间: $\sigma\downarrow(r_1,...,r_h)=\langle s_{t_1},s_{t_2},...,s_{t_l}\rangle$,其中,$t_1,\ldots,t_l$ 是 $r_1,\ldots,r_h$ 的最长严格递增序列,例如:

$$\langle s_0,s_1,s_2,s_3,s_4\rangle\downarrow\langle 0,0,2,2,4\rangle=\langle s_0,s_2,s_4\rangle.$$

PTL 的项和公式的解释定义为四元组$\mathcal{I}=(\sigma,i,k,j)$,其中,$i,k,j$ 是非负整数且 $0\leqslant i\leqslant k\leqslant j\leqslant|\sigma|$.$(\sigma,i,k,j)$表示$\sigma$上以 s_k 为当前状态的子区间$\sigma_{(i,j)}$.项 e 的解释$\mathcal{I}[e]$定义如下:

$$\mathcal{I}[v]=s_k[v]\in D',$$

$$\mathcal{I}[\bigcirc e]=\begin{cases}(\sigma,i,k+1,j)[e], & \text{如果}k<j\\ nil, & \text{其他情况}\end{cases},$$

$$\mathcal{I}[\Theta e]=\begin{cases}(\sigma,i,k-1,j)[e], & \text{如果}k>i\\ nil, & \text{其他情况}\end{cases},$$

$$\mathcal{I}[f(e_1,...,e_m)]=\begin{cases}\mathcal{I}[f](\mathcal{I}[e_1],...,\mathcal{I}[e_m]), & \text{如果对任意}h,1\leqslant h\leqslant m,\mathcal{I}[e_h]\neq nil\\ nil, & \text{其他情况}\end{cases}.$$

PTL 公式的可满足关系⊨定义如下.

- $\mathcal{I}\models p$ 当且仅当 $s_k[p]$=true;
- $\mathcal{I}\models e_1=e_2$ 当且仅当$\mathcal{I}(e_1)=\mathcal{I}(e_2)$;
- $\mathcal{I}\models P(e_1,\ldots,e_m)$当且仅当$\mathcal{I}[P](\mathcal{I}[e_1],\ldots,\mathcal{I}[e_m])$=true 且$\mathcal{I}[e_1]\neq nil,\ldots,\mathcal{I}[e_m]\neq nil$;
- $\mathcal{I}\models\phi_1\wedge\phi_2$ 当且仅当$\mathcal{I}\models\phi_1$ 且$\mathcal{I}\models\phi_2$;
- $\mathcal{I}\models\neg\phi$当且仅当$\mathcal{I}\not\models\phi$;
- $\mathcal{I}\models\exists v:\phi$当且仅当$\exists\sigma'$,$(\sigma',i,k,j)\models\phi$,其中,$\sigma$与$\sigma'$的唯一区别是 v 的取值可以不同;
- $\mathcal{I}\models\bigcirc\phi$当且仅当 $k<j$ 且$(\sigma,i,k+1,j)\models\phi$;
- $\mathcal{I}\models(\phi_1,\ldots,\phi_m)\ prj\ \phi$当且仅当存在非负整数 $k=r_0\leqslant r_1\leqslant\ldots\leqslant r_m\leqslant j$,使得$(\sigma,i,r_{l-1},r_l)\models\phi_l(1\leqslant l\leqslant m)$,并且对于以下的$\sigma'$之一,$(\sigma',0,0,|\sigma'|)\models\phi$:
  - (a) $r_m<j$ 并且 $\sigma'=\sigma\downarrow(r_0,...,r_m).\sigma_{(r_{m+1},j)}$;
  - (b) $r_m=j$ 并且$\sigma'=\sigma\downarrow(r_0,\ldots,r_m)$.

如果$(\sigma,0,0,|\sigma|)\models\phi$,则称公式$\phi$在区间$\sigma$上可满足,记做$\sigma\models\phi$.如果存在区间$\sigma$使得$\sigma\models\phi$,则称公式$\phi$是可满足的.如果对于任意的区间$\sigma$,$\sigma\models\phi$,则称公式$\phi$是有效的,记做$\models\phi$.

投影加 $prj^{\oplus}$(projection plus)和投影星 $prj^{\circledast}$ (projection star)公式是 PTL 的两个派生公式,投影加的定义见定义式 A1,投影星 $prj^{\circledast}$ 的定义见定义式 A2.在 A1 中,n 为任意的正整数或者无穷大;在 A2 中,ε表示到达终止状态,定义为$\varepsilon\triangleq\neg\bigcirc$true.由于篇幅限制,PTL 的其他派生公式及相应逻辑规则未在本文给出,可见文献[14,15].

A1. $(\phi_1,...,(\phi_i,...,\phi_l)^{\oplus},...,\phi_m)\ prj\ \phi\triangleq(\phi_1,...,(\phi_i,...,\phi_l)^{n},...,\phi_m)\ prj\ \phi$.

A2. $(\phi_1,...,(\phi_i,...,\phi_l)^{\circledast},...,\phi_m)\ prj\ \phi\triangleq(\phi_1,...,\varepsilon,...,\phi_m)\ prj\ \phi\vee(\phi_1,...,(\phi_i,...,\phi_l)^{\oplus},...,\phi_m)\ prj\ \phi$.

1.2 可执行语言MSVL

语法和语义. MSVL[14,15]是 PTL 的一个可执行子集,它支持整型、浮点型、字符、字符串、指针、数组、结构体等数据类型[17]以及函数调用[18].它由表达式和基本语句构成.算术表达式 e 和布尔表达式 b 定义如下.

$$e::=c\mid v\mid\bigcirc e\mid\Theta e\mid f(e_1,...,e_m)$$

$$b::=\text{true}\mid e_1=e_2\mid e_1<e_2\mid\neg b\mid b_1\wedge b_2$$

其中,c 为常量,v 为变量,f 表示函数.MSVL 语言的语句及语法定义见表 1.

Table 1 Syntax of MSVL statements

表 1 MSVL 语句及语法定义

语句名称	语法定义	语句名称	语法定义
终止语句	*empty*	长度语句	*len*(*n*)
赋值语句	$v \Leftarrow e$	保持语句	*keep*(ϕ)
赋值语句	$v:=e$	并行语句	$\phi_1 \| \phi_2$
下一状态语句	$\bigcirc\phi$	框架语句	*frame*(*v*)
合取语句	ϕ_1 and ϕ_2	条件语句	if (*b*) then {ϕ_1} else {ϕ_2}
析取语句	ϕ_1 or ϕ_2	循环语句	while (*b*) {ϕ}
顺序语句	$\phi_1;\phi_2$	同步语句	*await*(*b*)
单位长度语句	*skip*	–	–

这些语句的定义及语义见文献[14,15],本文仅介绍其直观含义.终止语句 *empty* 表示程序到达终止状态.赋值语句 $v \Leftarrow e$ 表示在当前状态下将 v 的值置为 e,且将记录 v 是否被赋值的命题变元 p_v 置为 true.赋值语句 $v:=e$ 表示在长度为 1 的区间上,在下一状态将 v 的值置为 e,且将记录 v 是否被赋值的命题变元 p_v 置为 true.$\bigcirc\phi$表示ϕ从下一状态开始执行.$\phi_1 \wedge \phi_2$ 表示同时执行ϕ_1和ϕ_2.$\phi_1 \vee \phi_2$ 表示执行ϕ_1或者ϕ_2.$\phi_1;\phi_2$ 表示先执行ϕ_1,ϕ_1 结束后继续执行ϕ_2.$\phi_1 \| \phi_2$ 表示ϕ_1和ϕ_2并行执行.*len*(*n*)表示长度为 n 的区间,*skip* 表示长度为 1 的区间.*keep*(ϕ)表示从当前状态开始,ϕ在除终止状态外的每个状态上执行.*frame*(ϕ)表示如果变量 v 未被赋值,那么它的值保持不变.条件语句和循环语句的含义与传统的程序设计语言相同.*await*(*b*)表示等待条件 b 成立,且在 b 成立时结束等待.

基于 MSVL 的统一模型检测. 该过程的原理如下:将一个系统建模成 MSVL 程序 M,待验证的性质描述成 MSVL 程序 M',为了判断系统是否满足这一性质,我们需要证明公式 $M \rightarrow M'$的有效性.由于$\neg(M \rightarrow M') \equiv M \wedge \neg M'$,等价地,我们判断公式 $M \wedge \neg M'$是否不可满足.对每一个$\sigma \models M \wedge \neg M'$,$\sigma$确定了一条系统违背给定性质的反例.因而,模型检测问题被转化为判断形如 $M \wedge \neg M'$的 PTL 公式的可满足性问题.由于模型 M 和性质 M'都是 PTL 公式,所以这种模型检测方法被称作统一模型检测.判定 PTL 公式 $M \wedge \neg M'$的可满足性问题涉及构造 MSVL 程序的范式、范式图以及寻找反例路径等过程,可见文献[13].以下给出 MSVL 程序的范式的定义.在文献[14,15]中,已证明所有的 MSVL 语句都可以化为范式.

范式(normal form). MSVL 程序ϕ的范式为 $\phi \equiv \vee_{i=1}^{n_1} \phi_{ei} \wedge empty \vee \vee_{j=1}^{n_2} \phi_{cj} \wedge \bigcirc\phi_{fj}$, 其中,$n_1$ 和 n_2 为非负整数,$n_1+n_2 \leqslant 1$;ϕ_{ei} 和 ϕ_{cj} 为 true 或者状态公式;ϕ_{fj} 为 MSVL 程序.

2 中断及嵌套中断系统的建模

在嵌入式系统及操作系统中,中断的优先级往往高于普通任务.这样,普通任务会因中断请求而暂停执行,直到中断事件处理完毕后恢复执行.在包含多个中断事件的系统中,系统按照中断事件的重要程度为其分配优先级.系统在响应一个中断请求的时候,可能被更高优先级的中断事件中断,从而引起嵌套中断.图 1 展示了一个嵌套中断场景,其中,X 为系统主程序,执行过程中随时可能被中断事件 *irq*1,*irq*2,*irq*3 和 *irq*4(优先级依次升高)中断.系统执行对应的中断服务程序 *Handler*1~*Handler*4 来响应中断请求.针对中断系统模型,本节先使用投影时序逻辑 PTL 定义只含一个中断事件的中断系统,建立了基本中断模型;然后,针对包含任意个数中断事件的(嵌套)中断系统,给出了基于 PTL 的建模方法.

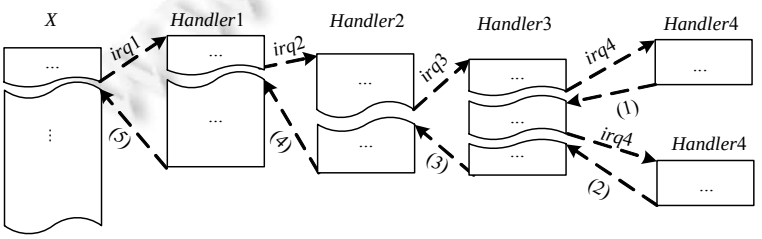


Fig.1 A snapshot of nested interrupts

图 1 一个中断嵌套场景

2.1 基本中断模型

我们使用语句ϕ when (p,b) do ϕ'表示只包含一个中断事件的中断系统的模型,简称基本中断模型.其中:公式ϕ代表中断系统中的主程序;p 是一个辅助定义的命题变元;b 为布尔表达式,表示是否可响应中断请求;公式ϕ'为中断服务程序.在定义中,我们需要确保中断语句ϕ when (p,b) do ϕ'随着主程序ϕ的执行结束而结束.基本中断模型的定义如下:

$$\phi \text{ when } (p,b) \text{ do } \phi' \triangleq ((\text{if } b \text{ then } \phi' \text{ else } skip)^{\circledast}, p \wedge \varepsilon)\ prj\ (\phi; p \wedge \varepsilon)) \wedge halt(p).$$

该定义式包含一个投影星($prj^{\circledast}$)公式,它是 PTL 中的投影结构的一个变种.在定义式中,p 是辅助定义的命题变元,便于投影结构左边和右边语句的执行能够同时结束.$halt(p)$是 PTL 的派生公式,它表示在任意时刻, p 为真等价于区间结束.基于 p,投影语句与 $halt(p)$的合取实现了投影符号(prj)两边的公式的同时结束,从而确保了中断语句随着主程序ϕ的执行结束而结束.显然,ϕ when (p,b) do ϕ'表示主程序ϕ在执行的过程中,一旦需要响应中断请求,ϕ将会暂停执行,让出执行权使得中断服务程序ϕ'执行.当ϕ'执行完后,返回主程序ϕ的中断点继续执行.图 2 给出了中断语句ϕ when (p,b) do ϕ'的执行示意图,其中:图 2(a)为主程序执行过程中无需响应中断请求的情况,图中垂直虚线连接的状态为投影状态,$skip$ 表示主程序ϕ的执行未被中断;图 2(b)表示在ϕ执行的过程中,有两次响应中断请求的过程,在这期间ϕ'的执行使得主程序ϕ暂停执行.

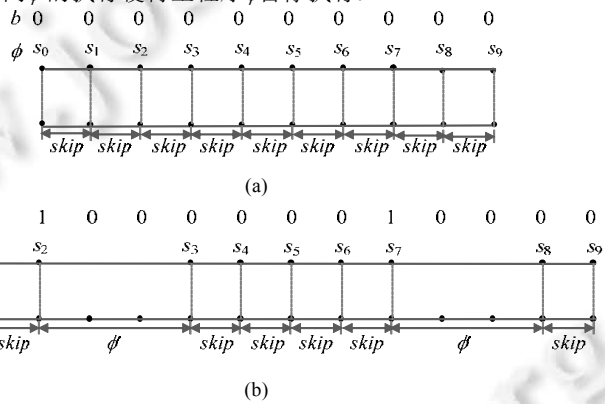


Fig.2 Schematic diagrams of executing interpretation statement ϕ when (p,b) do ϕ'

图 2 中断语句ϕ when (p,b) do ϕ'的执行示意图

2.2 嵌套中断系统模型

嵌套中断系统的模型基于基本中断模型定义而成.我们用如下析取式表示包含 n 个中断事件(优先级各不相同)的中断系统模型: $\vee_{i=1}^{n} interrupt(Q,r_i,H_i)$, 其中,

- Q 代表中断系统的主程序;
- 布尔类型变量 r_i 代表中断事件 i(i=1,…,n)是否请求响应:若请求响应,r_i=1;否则,r_i=0.且 r_i 的下标越大,所代表的中断事件的优先级越高;
- H_i 代表中断事件 i 的中断服务程序;
- $interrupt(Q,r_i,H_i)$表示主程序 Q 在响应中断事件 i 时的嵌套中断模型,它基于基本中断模型定义而成.

以 n=4 为例,$interrupt(Q,r_i,H_i)$(i=1,2,3,4)的定义如下:

$$interrupt(Q,r_1,H_1) \triangleq Q \text{ when } (p,r_1) \text{ do } (\vee_{i=2}^{4} interrupt(H_1,r_i,H_i)),$$
$$interrupt(Q,r_2,H_2) \triangleq Q \text{ when } (p,r_2) \text{ do } (\vee_{i=3}^{4} interrupt(H_2,r_i,H_i)),$$
$$interrupt(Q,r_3,H_3) \triangleq Q \text{ when } (p,r_3) \text{ do } (interrupt(H_3,r_4,H_4)),$$
$$interrupt(Q,r_4,H_4) \triangleq Q \text{ when } (p,r_4) \text{ do } H_4,$$

其中,

$$interrupt(H_1,r_2,H_2)\triangleq H_1 \text{ when } (p,r_2) \text{ do } (\vee_{i=3}^{4} interrupt(H_2,r_i,H_i)),$$
$$interrupt(H_1,r_3,H_3)\triangleq Q \text{ when } (p,r_3) \text{ do } (interrupt(H_3,r_4,H_4)),$$
$$interrupt(H_1,r_4,H_4)\triangleq H_1 \text{ when } (p,r_4) \text{ do } H_4,$$
$$interrupt(H_2,r_3,H_3)\triangleq H_2 \text{ when } (p,r_3) \text{ do } (interrupt(H_3,r_4,H_4)),$$
$$interrupt(H_2,r_4,H_4)\triangleq H_2 \text{ when } (p,r_4) \text{ do } H_4,$$
$$interrupt(H_2,r_1,H_1)\triangleq H_2 interrupt(H_3,r_4,H_4)\triangleq H_3 \text{ when } (p,r_4) \text{ do } H_4,$$
$$interrupt(H_3,r_i,H_i)\triangleq H_3 \ (i<3),$$
$$interrupt(H_4,r_i,H_i)\triangleq H_4 \ (i<4).$$

需要注意的是:在有中断请求发生的情况下,即使是多个中断嵌套,同一时刻也只有一个中断服务程序在执行,不存在一个主程序既被中断事件 r_i 中断,又被事件 $r_j(i\neq j)$中断.中断事件的模型析取式中的多个 *interrupt*()有且仅有一个为真.以 $interrupt(Q,r_1,H_1)$定义式中的 $\vee_{i=2}^{4} interrupt(H_1,r_i,H_i)$ 为例,当 $interrupt(H_1,r_i,H_i)(i=2,3,4)$为真时表示为 p_i,否则表示为$\neg p_i$.那么 $\vee_{i=2}^{4} p_i$ 应满足表 2 所示的真值表.即,$\vee_{i=2}^{4} p_i$ 应该满足 $p_2\wedge\neg p_3\wedge\neg p_4\vee\neg p_2\wedge p_3\wedge\neg p_4\vee\neg p_2\wedge\neg p_3\wedge p_4$.对于其他情况,与此类似.

Table 2 Truth table

表 2 真值表

p_2	p_3	p_4	
1	0	0	$p_2\wedge\neg p_3\wedge\neg p_4$
0	1	0	$\neg p_2\wedge p_3\wedge\neg p_4$
0	0	1	$\neg p_2\wedge\neg p_3\wedge p_4$

图 3 为含有 4 个中断事件的中断系统未发生中断请求时的模型图,主程序 Q 的执行生成了状态序列 $s_1,\ldots,s_{16}$.图 4 为发生中断请求时的模型图.如图所示:在多个中断事件同时请求响应的情况下,它们将按优先级高低的顺序依次被响应.例如:主程序执行到状态 s_2 时,中断请求 r_1,r_2,r_3 同时到达,而中断请求 r_3 优先被响应.且图 4 包含了低优先级的中断服务程序被高优先级的中断事件抢占的情况.例如:在状态 s_5 和 s_6 之间,中断响应程序 H_1 中断了主程序的执行;随后,H_1 被中断响应程序 H_3 中断.

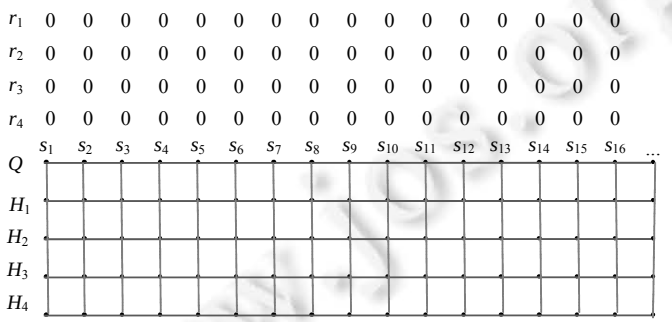


Fig.3 Nested interrupt model without interrupt request

图 3 无中断请求时的嵌套中断模型

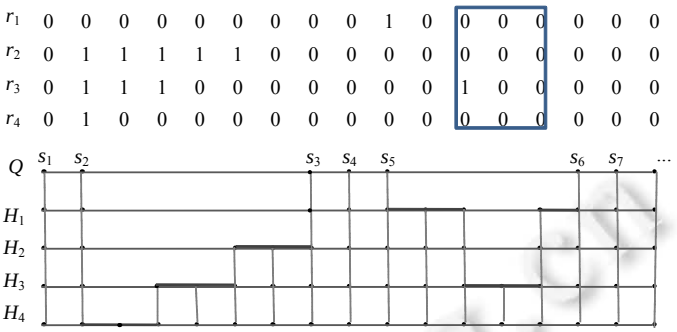


Fig.4 Nested interrupt model with interrupt requests

图 4 发生中断请求时的嵌套中断模型

3 MSVL 解释器的扩展

MSVL 解释器的架构如图 5 所示.建模、仿真和验证是 MSVL 解释器的三大功能模块.建模功能可以构造出输入程序的范式图[13],它显示了程序的状态空间;仿真功能可以得到程序的一条执行路径;验证功能验证程序是否满足指定的性质,若不可满足,给出一条反例路径.这 3 种功能的实现主要借助于对程序的状态化简和区间化简.其中,状态化简对 MSVL 程序的语法树进行表达式求值和化范式,状态化简完成后对程序做区间化简,从而确定下一状态的要执行的程序.

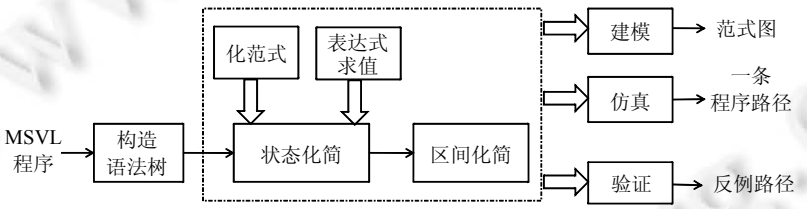


Fig.5 Architecture of MSVL interpreter

图 5 MSVL 解释器架构

为使得 MSVL 解释器支持对含中断结构的程序的建模仿真和验证,我们在 MSVL 解释器中增加对中断语句的处理.该工作主要涉及对中断语句构造语法树和化范式,其他的处理可以直接调用解释器的内部功能.

3.1 语法树的构造

基本中断语句ϕ when (p,b) do ϕ'的语法树结构如图 6 所示,根节点代表语句类型,即中断语句,3 个子节点分别对应主程序的语法树、中断处理的条件以及中断服务程序的语法树.由于 p 是辅助中断语句定义的命题变元,没有实际含义,可不做处理,故在语法树中省略了对 p 的表示.

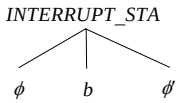


Fig.6 Syntax tree of the interrupt statement

图 6 中断语句的语法树

3.2 状态化简

状态化简涉及到对程序化范式以及对表达式(算术表达式和布尔表达式)求值,该过程见文献[15].

假设中断语句ϕ when (p,b) do ϕ'中主程序ϕ的范式为$\phi \Leftarrow \phi_e \wedge empty \vee \phi_c \wedge \bigcirc \phi_f$,$\phi'$的范式为

$$\phi' \equiv \phi'_e \wedge empty \vee \phi'_c \wedge \bigcirc\phi'_f \ .$$

图 7 给出了在化简中断语句ϕ when (p,b) do ϕ'时语法树的变化情况,其中,

- 图 7(i)代表ϕ when (p,b) do ϕ'的语法树;
- 图 7(ii)为$\phi_e \wedge empty$ 的语法树;
- 图 7(iii)为$\phi_c \wedge \bigcirc(\phi_f$ when (p,b) do $\phi')$的语法树;
- 图 7(iv)为 $\phi_c \wedge \phi'_c \wedge \bigcirc(\phi'_f;\phi_f$ when (p,b) do $\phi')$ 的语法树.

状态化简中断语句时,首先执行程序ϕ:

- 若ϕ结束,则中断语句执行结束,法树的变换如图 7(ii)所示;
- 若ϕ未结束,则判断中断响应条件 b 是否成立:若不成立,执行ϕ,使得语法树变为图 7(iii)所示的形式;否则,执行ϕ'且语法树变为图 7(iv)所示的形式.

在语法树(iii)和(iv)中,以 *NEXT_STA* 为根节点的子树为下一状态语法树.

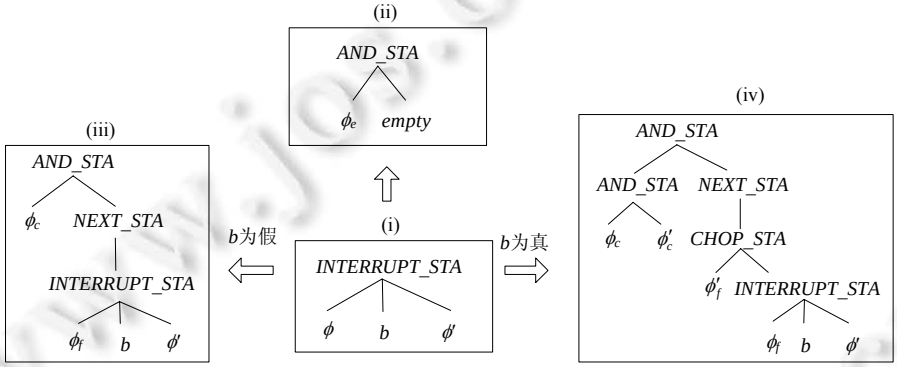


Fig.7 State reduction of the interrupt statement

图 7 中断语句的状态化简过程

4 实验与分析

本节建模并验证图 8 所示的中断系统实例.主程序包含一个局部变量 *sum*,主程序不断对 *sum* 的值加 1,直到 *sum* 的值等于 100 或者全局变量 *timerCount* 的值变为 0 时,结束对 *sum* 的自增操作,并返回 *sum* 的值.主程序在执行过程中,随时可能被两个中断事件中断:当发生中断事件 1 时,触发中断服务程序 *ISR*1 的执行,全局变量 *timerCount* 的值被减 1;当发生中断事件 2 时,触发中断服务程序 *ISR*2 的执行,使得全局变量 *timerCount* 的值被重设为 20.注意,这里,中断事件 2 的优先级高于中断事件 1.

```
int timerCount=20;
int main()
{   int sum=0;
    while(!(timerCount==0||sum==0))
    {sum=sum+1;}
    return sum;
}
```

```
void ISR1()
{
  //… disable interrupt
  if (timerCount!=0)
  {timerCount--;}
  display();
  //…enable interrupt
}
```

```
void ISR2()
{
  //… disable interrupt
  timerCount=20;
  display();
  //…enable interrupt
}
```

Fig.8 An instance of interrupt systems

图 8 一个中断系统实例

我们的目标在于验证主程序的数据安全性,即:中断服务程序的执行不改变主程序里变量的值,即 *sum* 的值.为验证这个性质,接下来我们用 MSVL 建模中断系统实例并描述待验证性质.

4.1 建模

我们将图 8 所示的中断系统实例的嵌套中断模型记做 M,其中,变量的声明与初始化模块用 *DcInt* 表示,主程序的形式化模型用 Q 表示,中断服务程序 *ISR*1 和 *ISR*2 的形式化模型分别用 H_0,H_1 表示.借助于第 2 节提出的建模方法,M 可被定义为如下形式.

- $M \triangleq DeInt;(interrupt(Q,r[0],H_0) \vee interrupt(Q,r[1],H_1))$;
- $interrupt(Q,r[0],H_0) \triangleq Q$ when $(p,r[0])$ do $(interrupt(H_0,r[1],H_1))$;
- $interrupt(H_0,r[1],H_1) \triangleq H_0$ when $(p,r[1])$ do H_1;
- $interrupt(Q,r[1],H_1) \triangleq Q$ when $(p,r[1])$ do H_1.

其中,$r[j]$代表事件 j+1(j=0,1)请求响应且系统允许中断.M 中的各个模块的 MSVL 代码如图 9 所示.在变量的声明与初始化模块 *DcInt* 中,各变量的含义如下.

- *sum* 和 *timerCount* 为中断系统实例中的变量;
- 数组 *iEv* 代表发生的事件,*iEv*[0],*iEv*[1]分别表示中断事件 1 和中断事件 2,若中断事件 j+1 请求处理,则 *iEv*[j]=1;否则 *iEv*[j]=0(j=0,1);
- *ie* 表示是否允许中断,若允许中断,*ie*=1;否则,*ie*=0.

在允许中断且没有更高优先级中断时,中断事件 *iEv*[j]的请求将得到响应,并且 *isr*[j]被设为 1.在该中断事件处理完毕后,*isr*[j]被设为 0.变量 *inv*1 和 *inv*2 是为辅助性质验证引入的变量,分别记录每次执行中断服务程序前变量 *sum* 和 *timerCount* 的值.

DcInt	*Q*	H_0	H_1

frame(*sum*,*timerCount*,*ie*,
isr,*iEv*,*inv*1,*inv*2) and
int sum⇐0 and
int timerCount⇐20 and
int iEv[2]⇐{0,0} and
int isr[2]⇐{0,0} and
int ie⇐1 and
*int inv*1, *inv*2 and
skip
	while (*timerCount*!=0)
	{
	if (*ie*=0 or (*iEv*[0]=0 and *iEv*[1]=0))
	then {*sum*:=*sum*+1}
	else {*skip*}
	}
		*inv*1:=*sum* and
		*inv*2:=*timerCount* and
		isr[0]:=1;
		ie:=0;
		timerCount:=*timerCount*−1;
		ie:=1;
		iEv[0]:=0 and
		isr[0]:=0
			*inv*1:=*sum* and
			*inv*2:=*timerCount* and
			isr[1]:=1;
			ie:=0;
			timerCount:=20;
			ie:=1;
			iEv[1]:=0 and
			isr[1]:=0

Fig.9 MSVL programs of the modules in the instance of interrupt systems

图 9 中断系统实例中各模块的 MSVL 代码

4.2 验 证

本节借助 MSVL 解释器仿真和验证模型 M.表 3 列出了 3 个输入实例的模型,其中:输入 1 对应先发生了一次中断事件 1,接着发生了一次中断事件 2 的情况,其时间间隔为 1 个时间单位;输入 2 描述在第 4 个时间单位,中断事件 1 和中断事件 2 同时请求响应的情况;输入 3 描述在第 4 个时间单位,中断事件 1 和中断事件 2 同时请求响应,随后,过了 25 个时间单位,它们分别又请求一次响应的情况.

Table 3 Input instances

表 3 输入样例

输入 1	*iEv*[0]:=1; *iEv*[1]:=1; true
输入 2	*len*(4); *iEv*[0]⇐1 and *iEv*[1]⇐1
输入 3	*len*(4);*iEv*[0]⇐1 and *iEv*[1]⇐1 and *empty*; *len*(25); *iEv*[0]⇐1 and *skip*; *iEv*[1]⇐1

本文验证中断系统中主程序的数据安全性性质.在主程序执行过程中,主程序中的一些变量不可以被中断服务程序改变,即:任何时候主程序被中断,在中断返回前后,这些变量的值都是一致的.在 M 中,变量 *sum* 为局部变量,因而在主程序被中断转而执行中断服务程序的过程中,该变量的值不允许被中断服务程序修改.为描述这个性质,我们用变量 *inv*1 记录在进入中断服务程序前的 *sum* 值,并定义命题 p_0,p_1,q_1,q_2 为如下形式:

define p_0: *isr*[0]=1; define p_1: *isr*[1]=1; define q_1: *sum*=*inv*1; define q_2: *timerCount*=*inv*2.

借助以上命题,变量 *sum* 和 *timerCount* 的数据安全性性质分别记为 Pr_1 和 Pr_2,则它们可用 PTL 公式定义为如下形式.

- $Pr_1 \triangleq \Box((p_0 \vee p_1) \rightarrow q_1)$;
- $Pr_2 \triangleq \Box((p_0 \vee p_1) \rightarrow q_2)$;

使用扩展后的 MSVL 解释器对表 3 的输入进行验证,验证结果见表 4.

Table 4 Verification results

表 4 验证结果

输入	验证结果(√:有效;×:违背)	
	性质 Pr_1	性质 Pr_2
输入 1	√	×
输入 2	√	×
输入 3	√	×

对于所给输入,性质 Pr_1 有效,而 Pr_2 被违背.由于仅主程序可以访问变量 *sum*,所以在响应中断服务程序的过程中,该变量的值是不允许被改变的,验证结果与实际相符.对于变量 *timerCount*,由于中断服务程序 *ISR*1 和 *ISR*2 都要访问且修改该变量,所以在响应中断的过程,该变量的值被改变,验证结果也与实际相符.

5 结束语

本文提出一种建模和验证嵌套中断系统的方法.

- 首先,为中断系统提出了基于投影时序逻辑的形式化定义,并将这种定义应用到建模包含任意个中断事件的中断系统上,从而得出嵌套中断系统基于投影时序逻辑的形式化模型;
- 其次,使用定义的中断语句扩充 MSVL 语言,使得 MSVL 语言支持对嵌套中断系统的建模;
- 再次,扩展了 MSVL 解释器,为建模仿真和验证包含任意多中断事件的中断系统提供了工具支持,从而实现了嵌套中断系统基于 MSVL 的统一模型检测;
- 最后,使用一个实例展示如何使用本文提出的方法建模和验证嵌套中断系统.使用本文提出的方法,方便建出简洁的系统模型,且与 PLTL[12]相比,MSVL 对性质的描述更完整.

在今后的工作中,我们将研究如何对实际应用中的中断系统建立 MSVL 模型.由于中断程序往往是由 C 语言或者汇编语言编写的,我们将结合文中所提出的模型,并借助 C 语言以及汇编语言到 MSVL 语言的转换技术辅助建模具体应用中的中断系统.

References:

[1] Cui J, Duan ZH, Tian C, *et al*. Modeling and verification of an interrupt system in μC/OS-III with TMSVL. In: Proc. of the Int'l Workshop on Structured Object-Oriented Formal Language and Method. Cham: Springer-Verlag, 2015. 15−28. [doi: 10.1007/978-3-319-31220-0_2]

[2] Regehr J, Cooprider N. Interrupt verification via thread verification. Electronic Notes in Theoretical Computer Science, 2007, 174(9):139−150. [doi: 10.1016/j.entcs.2007.04.002]

[3] Feng X, Shao Z, Dong Y, *et al*. Certifying low-level programs with hardware interrupts and preemptive threads. In: Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI 2008). ACM Press, 2008. 170−182. [doi: 10.1145/1375581.1375603]

[4] Xu F, Fu M, Feng X, *et al*. A practical verification framework for preemptive OS kernels. In: Proc. of the 28th Int'l Conf. on Computer Aided Verification (CAV 2016). Springer Int'l Publishing, 2016. 59−79. [doi: 10.1007/978-3-319-41540-6_4]

[5] Hoare CAR. An axiomatic basis for computer programming. Communications of the ACM, 1969,26(1):53−56.

[6] Brookes S. A semantics for concurrent separation logic. In: Proc. of the 15th Int'l Conf. on Concurrency Theory (CONCUR 2004). Springer-Verlag, 2004. 16−34. [doi: 10.1007/978-3-540-28644-8_2]

[7] Coq Development Team. The Coq Proof Assistant Reference Manual. The Coq Release v8.1. 2006.

[8] Alur R, Dill DL. A theory of timed automata. Theoretical Computer Science, 1994,126(2):183−235.

[9] Hou G, Zhou K, Chang J, *et al*. Interrupt modeling and verification for embedded systems based on time Petri nets. In: Proc. of the Int'l Workshop on Advanced Parallel Processing Technologies. Berlin: Springer-Verlag, 2013. 62−76. [doi: 10.1007/978-3-642-45293-2_5]

[10] Liu H, Zhang H, Jiang Y, *et al*. iDola: Bridge modeling to verification and implementation of interrupt-driven systems. In: Proc. of the Theoretical Aspects of Software Engineering Conf. (TASE). IEEE, 2014. 193−200. [doi: 10.1109/TASE.2014.33]

[11] Li G, Yuen S, Adachi M. Environmental simulation of real-time systems with nested interrupts. In: Proc. of the 2009 3rd IEEE Int'l Symp. on Theoretical Aspects of Software Engineering (TASE 2009). IEEE, 2009. 21−28. [doi: 10.1109/TASE.2009.12]

[12] Baier C, Katoen JP. Principles of Model Checking. The MIT Press, 2008.

[13] Duan ZH, Tian C. A unified model checking approach with projection temporal logic. In: Proc. of the 10th Int'l Conf. on Formal Engineering Methods (ICFEM 2008). Heidelberg: Springer-Verlag, 2008. 167−186. [doi: 10.1007/978-3-540-88194-0-12]

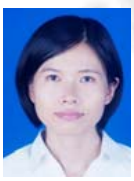
[14] Duan ZH, Yang X, Koutny M. Framed temporal logic programming. Science of Computer Programming, 2008,70(1):31−61. [doi: 10.1016/j.scico.2007.09.001]

[15] Duan ZH. Temporal Logic and Temporal Logic Programming. Beijing: Science Press, 2005.

[16] Zhang N, Duan ZH, Tian C. Model checking concurrent systems with MSVL. Science China Information Sciences, 2016,59(11): 118101. [doi: 10.1007/s11432-015-0882-6]

[17] Wang X, Duan ZH, Zhao L. Formalizing and implementing types in MSVL. In: Proc. of the Int'l Workshop on Structured Object-Oriented Formal Language and Method. New York: Springer-Verlag, 2013. 62−75. [doi: 10.1007/978-3-319-04915-1_5]

[18] Zhang N, Duan ZH, Tian C. A mechanism of function calls in MSVL. Theoretical Computer Science, 2016,654:11−25. [doi: 10.1016/j.tcs.2016.02.037]



崔进(1989－),女,陕西西安人,博士生,主要研究领域为时序逻辑,模型检测,实时系统.

段振华(1948－),男,博士,教授,博士生导师,主要研究领域为网络计算,高可信软件理论和技术.

田聪(1981－),女,博士,教授,主要研究领域为形式化方法,时序逻辑,模型检测.

张南(1984－),女,博士,副教授,CCF 专业会员,主要研究领域为形式化方法,软件验证,逻辑与自动机.