

文章编号:1007-130X(2014)05-0900-06

# 一种嵌入式操作系统运行时验证方法\*

张可迪,舒绍娴,董 威

(国防科学技术大学计算机学院,湖南长沙 410073)

**摘要:**作为测试、模型检验等开发阶段所用技术的有效补充,运行时验证技术越来越受到广泛的关注。然而,当前的运行时验证技术主要用于应用软件,很少专门针对操作系统进行研究。对面向嵌入式操作系统的运行时验证框架和关键技术进行了研究,并结合一个开源嵌入式操作系统 FreeRTOS 进行了设计与实现。首先提出了一种面向嵌入式操作系统的运行时验证和反馈调整框架,然后针对框架中的关键技术部分,完成了规约语言的设计、三值语义监控器的生成、FreeRTOS 嵌入式操作系统相关接口的实现等主要工作。

**关键词:**嵌入式操作系统;FreeRTOS;运行时验证;规约语言;三值语义监控器

**中图分类号:**TP311.5

**文献标志码:**A

**doi:**10.3969/j.issn.1007-130X.2014.05.020

## A runtime verification method for embedded operating system

ZHANG Ke-di, SHU Shao-xian, DONG Wei

(College of Computer, National University of Defense Technology, Changsha 410073, China)

**Abstract:** As an effective supplement of testing and model checking, runtime verification technique attracts more and more attentions. However, the current runtime verification technology is mainly used for application software. Very few are specialized for monitoring the running state of an operating system. The paper studies the runtime verification framework and key techniques for embedded operating system and realizes a demo combined with an open source system FreeRTOS. Firstly, an embedded operating system oriented framework for runtime verification and feedback adjustment is proposed. Secondly, based on the critical part of our frame, the specification language, three-valued semantic monitor generation and FreeRTOS related interfaces are designed and implemented.

**Key words:** embedded operating system; FreeRTOS; runtime verification; specification language; three-valued semantic monitor

### 1 引言

目前计算机技术的应用领域日益广泛,从传统到高新,从军事到民生,都与计算机技术密不可分。然而随着计算机应用中软件所占部分规模的日益扩大,如何提高软件的可靠性和安全性受到了学术界和工业界的广泛关注和深入研究。人们为确保

软件的正确性、可靠性、安全性、可用性和可维护性,在设计、编码、测试等不同阶段不遗余力地采取各种各样的方法和手段。但是,由于软件自身特性,软件失效仍然难以避免,尤其是关键软件的可靠性、安全性问题仍然面临严峻挑战。

运行时验证<sup>[1]</sup>是一种新兴的轻量级程序验证技术。在运行时验证中,通常从系统需求中产生监控器,监控器通过观测程序的执行来检查程序运行

\* 收稿日期:2012-11-09;修回日期:2013-04-17

基金项目:国家自然科学基金资助项目(60970035);国家 863 计划资助项目(2011AA010106)

通信地址:410073 湖南省长沙市国防科学技术大学计算机学院

Address: College of Computer, National University of Defense Technology, Changsha 410073, Hunan, P. R. China

过程是否满足系统需求,是传统的软件验证和确认技术如测试<sup>[2]</sup>和模型验证<sup>[3]</sup>的有效补充。它不但可以有效地检测系统运行中的异常行为,也使得在检测到正确性背离问题时有效地修复系统成为可能。

传统的运行时验证技术主要针对实际的应用程序,关注应用程序本身的运行过程是否满足系统需求,忽视了对可能在操作系统一级出现的如任务冲突或任务调度、资源使用等过程不满足事先设计等软件失效的监控。这是由于操作系统本身的控制调度十分复杂,监控软件要对其内核运行过程进行访问和操作来获取监控信息或执行反馈动作十分不易。而嵌入式操作系统结构相对简单,其在航空航天、国防等领域又非常关键,例如许多航天系统对操作系统中的任务执行过程都有严格限制,不仅关注具体状态,还对执行的时序、资源分配等有具体要求,因此嵌入式操作系统的运行过程采用运行时验证技术进行监控非常有必要。同时,只要在嵌入式操作系统内核中加入提供可供监控器使用的的数据获取接口,可使其运行时验证成为可能。

本文主要针对嵌入式操作系统的运行时验证技术展开研究,并结合 FreeRTOS 嵌入式操作系统提出了一种面向嵌入式操作系统的运行时验证框架,实现了对嵌入式操作系统调度过程信息的获取,并通过从事先编写好的规约自动生成监控器以对系统的运行轨迹进行监控,当系统的运行轨迹发生异常时,能够执行相应的反馈操作尽量避免违反规约的情况发生。

本文的组织如下:第 2 节介绍该方法的整体框架,第 3 节介绍相关的规约语言,第 4 节进行 FreeRTOS 监控接口和反馈接口的设计,第 5 节阐述了监控器的构造与实现,最后对文章进行了总结并对下一步工作进行概述。

## 2 整体架构

本文研究的问题根据运行时验证的主要工作流程,可以分解为以下几个子问题:

(1)如何定义嵌入式操作系统应该满足的性质规约和相关的键属性,如何建立键属性与性质规约之间的联系;

(2)如何在系统运行过程中提取系统键属性的相关信息;

(3)如何在目标系统和监控器之间传递信息;

(4)如何从性质规约生成监控器模型,并实现

能在违反规约时执行反馈动作的监控程序。

针对上面四个子问题,解决方案分别如下:

(1)将使用线性时序逻辑 LTL (Line Temporal Logic) 公式描述软件系统应该满足的性质规约,构造为基于 LTL 三值语义的运行时监控器;通过借鉴经典的运行时验证框架,引入一套事件和条件机制,给出扩展的事件定义语言 EDL (Event Definition Language)。按照事件定义语言的规定,由目标系统的键属性抽象出一组事件和条件,同时将这些事件和条件作为待验证的性质即 LTL 公式的谓词。这样,就在目标系统的键属性和待验证的性质规约之间建立起了联系。

(2)本文选用的嵌入式操作系统是 FreeRTOS<sup>[4]</sup>,该操作系统为开源代码,主要由 C 语言以及少量的汇编语言编写而成,这符合很多现实应用的需要(许多键领域如我国航天领域的代码都是由 C 语言编写而成)。对于 C 语言,不能像 Java 语言那样方便地进行相关监控代码的自动插装,这主要是由于它不存在用于给插装工具定位用的字节码和“类”结构。与应用程序的运行时验证相比,操作系统的运行时验证还存在以下特点:

①操作系统的功能和结构相对确定,而不同的应用程序其功能和设计结构区别很大。

②操作系统的内核在运行时访问受限,而应用程序则没有这种限制。

③操作系统要监控的内容和性质相对比较确定,而应用程序则因不同的需求而异。

结合嵌入式操作系统的特性,可以确定在运行时通过自动插装的方法到操作系统内核中去获取信息难以实现,因为运行时在内核中进行插装操作是不安全和不稳定的。而由于操作系统相对固定的结构和相对确定的监控属性类型,本文将不采用程序自动插装的方式,而是在 FreeRTOS 嵌入式操作系统中提前定义并实现相关监控接口,以提取系统键属性的信息。

(3)根据航天等领域的实际应用现状,操作系统监控接口本身将获取的数据以日志的形式进行存储和传输,监控程序通过对日志的访问分析获取监控信息。

(4)使用开源软件 LTL3 Tools 可以在输入 LTL 公式后自动生成监控器模型,根据监控器模型通过 JavaMOP 形成监控程序,同时要在 FreeRTOS 嵌入式操作系统中加入能够接受反馈控制指令的接口。

根据问题和解决方案,本文提出一种面向嵌入

式操作系统的运行时验证框架,如图 1 所示。

根据图 1 中嵌入式操作系统运行时验证的框架,本文将从规约语言的设计、操作系统的扩展、监控器的构造与实现三个方面进行具体工作的阐述。

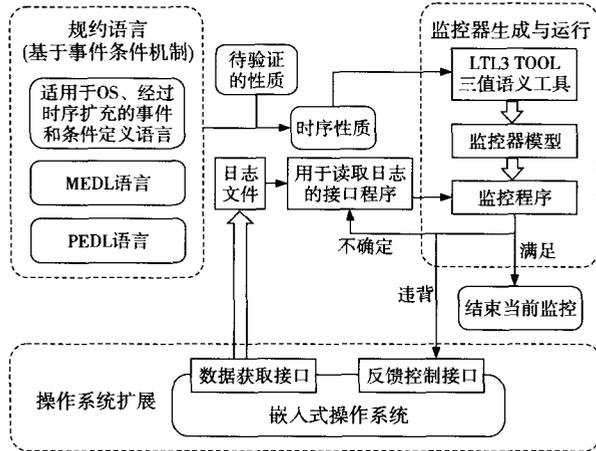


Figure 1 Runtime verification framework based on embedded operating system

图 1 基于嵌入式操作系统的运行时验证框架

### 3 规约语言

#### 3.1 事件与条件定义语言

线性时序逻辑 LTL<sup>[5]</sup>是在命题逻辑的基础上加上时序操作而得来的,基于线性时序逻辑的规约与验证是描述和验证软件系统的一类重要形式化方法,最终监控器要监控的性质就是由 LTL 公式描述。为了将系统的“高层规约”与程序相关的“底层信息”联系起来,作者借鉴了经典的运行时验证框架 MaC<sup>[6]</sup>中的事件和条件机制。将事件定义为某个时刻发生的一个动作,条件则定义为某一段时间成立的一个命题。比如进入或者退出某个方法是一个事件,而像  $x=1$  之类表示程序在某一段时间内成立的一个命题则表示一个条件。下面介绍针对操作系统和时序逻辑的需要进行扩展后的事件与条件机制。

(1)语法。

条件由原子条件  $C$  通过逻辑符号相连递归而成。其中的原子条件是指程序中某些属性抽象而成的一个条件,比如由监控的变量通过简单的符号连接形成真假表达式,原子条件的定义在面向嵌入式操作系统的运行时验证框架中和在 MaC 中的定义类似,并不需要扩充。同样,事件由原子事件  $E$  通过逻辑符号连接递归而成, MaC 定义的一些原子事件的关注对象是方法和变量,因为它监控的对

象是应用程序。而当主要的关注对象是操作系统运行过程与任务、中断、资源等相关的内容时,则定义一部分适用于操作系统的原子事件,这些原子事件与条件能用于方便地描述操作系统的具体信息。另外, MaC 中定义的事件与条件机制只满足了部分逻辑表达的能力,但要完全表达 LTL 公式中的时序逻辑关系还有很大不足,因此要对事件与条件机制进行相应的扩展。

条件  $\langle C \rangle$  和事件  $\langle E \rangle$  扩展后的语法定义如图 2 所示,其中  $\langle TC \rangle$  是对条件关于线性时序逻辑部分的扩展,  $\langle atom E \rangle$  是针对于操作系统中的原子事件的定义。

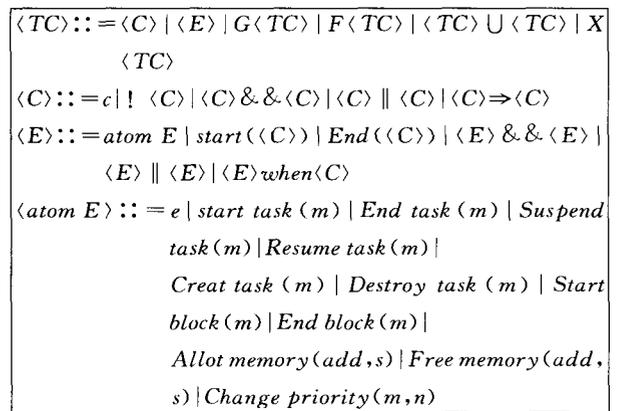


Figure 2 Formal definition syntax of events and condition

图 2 事件与条件的形式化定义语法

(2)语义。

首先定义模型  $M$  为二元组  $\{S, \tau\}$ , 其中  $S = \{s_1, s_2, \dots\}$ ,  $\tau$  是  $S$  到时间域的一个映射,即  $\tau(S_i)$  表示事件  $S$  发生的时间。若模型  $M$  中条件  $c$  在时间  $t$  下值为 true,则表示为  $(M, t \models c)$ ;同理,若事件在时间  $t$  发生,则表示为  $(M, t \models e)$ 。事件与条件的形式化语义如图 3a 所示,针对操作系统的原子事件  $atom E$  的形式化定义语义如图 3b 所示。

#### 3.2 事件定义语言

在面向嵌入式操作系统的运行时验证方法框架中,事件和条件分为由目标系统中的变量和方法抽象而成的原子事件和条件,以及由原子事件和条件构成的复合事件和条件两种。

为了准确描述原子事件和条件以及复合事件和条件,本文在事件定义语言的基础之上引进了 MaC-Java<sup>[7]</sup>中的原子事件定义语言(PEDL)和复合事件定义语言(MEDL)<sup>[8]</sup>,它们分别用 PEDL 规约和 MEDL 规约描述。其中 PEDL 规约中描述了原子事件和条件是如何由目标系统中的变量和方

$M, t \models c$	iff	在模型 $M$ 中, 条件 $c$ 在时刻 $t$ 成立;
$M, t \models e$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 时, 事件 $e$ 发生;
$M, t \models \text{start}(c)$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 并且 $M, \tau(s) \models c$ , 并且如果 $\delta > 0$ , $M, \tau(s + \delta) \models c$ ;
$M, t \models \text{end}(c)$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 并且 $M, \tau(s) \models !c$ , 并且如果 $\delta > 0$ , $M, \tau(s + \delta) \models !c$ ;
$M, t \models a \parallel e$	iff	$M, t \models a$ 或者 $M, t \models e$ ;
$M, t \models a \& \& e$	iff	$M, t \models a$ 并且 $M, t \models e$ ;
$M, t \models e \text{ when } c$	iff	$M, t \models e$ 并且 $M, t \models c$ ;
$M, t \models !c$	iff	$M, t \not\models c$ ;
$M, t \models a \parallel c$	iff	$M, t \models c$ 或者 $M, t \models a$ ;
$M, t \models a \& \& c$	iff	$M, t \models c$ 并且 $M, t \models a$ ;
$M, t \models c_1 \& c_2$	iff	$M, t \models c_1$ 或者 $M, t \models c_2$ ;
$M, t \models G(c)$	iff	对于 $\forall t' \geq t$ , $M, t' \models c$ ;
$M, t \models F(c)$	iff	$\exists t' > t$ , 使得 $M, t' \models c$ ;
$M, t \models c_1 \cup c_2$	iff	存在 $t' \leq t$ 使得成立 $M, t' \models c_1$ , 并且对于 $t_0 \leq t' < t$ , $M, t' \models c_2$ ;
$M, t \models X(c)$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 并且如果 $\delta > 0$ , $M, \tau(s + \delta) \models c$ .

a 事件与条件的形式化定义语义

$M, t \models \text{Start task}(m)$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 时, 任务 $m$ 进入运行态;
$M, t \models \text{End task}(m)$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 时, 任务 $m$ 退出运行态;
$M, t \models \text{Suspend task}(m)$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 时, 任务 $m$ 进入挂起状态;
$M, t \models \text{Resume task}(m)$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 时, 任务 $m$ 退出挂起状态;
$M, t \models \text{Start block}(l)$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 时, 任务 $m$ 开始中断;
$M, t \models \text{End block}(l)$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 时, 任务 $m$ 结束中断;
$M, t \models \text{Alloc memory}(add, s)$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 时, 从 $add$ 位置开始分配 $s$ 大小内存区域;
$M, t \models \text{Free memory}(add, s)$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 时, 从 $add$ 位置开始释放 $s$ 大小内存区域;
$M, t \models \text{Change priority}(m, n)$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 时, 变更任务 $m$ 的优先级到 $n$ ;
$M, t \models \text{Creat task}(m)$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 时, 创建任务 $m$ ;
$M, t \models \text{Delet task}(m)$	iff	存在状态 $s$ : 使得 $\tau(s) = t$ 时, 删除任务 $m$ .

b atom E 形式化定义语义

Figure 3 Semantics for event & condition and atom E formal definition

图3 事件与条件和针对于操作系统的原子事件 atom E 的形式化定义语义

法抽象而成;而 MEDL 规约中描述了原子事件和条件如何组成复合事件和条件,即构成监控器要使用的时序性质谓词,同时声明要使用的反馈动作。而与 MaC-Java 中 PEDL、MEDL 语言不同,因为当前不能实现自动插装来获取相应的信息,针对的目标也是 C 语言,所以具体的语法要重新设计。其功能上主要是为了能规范地描述事件与条件,方便生成相关的代码,并为构造监控器提供用于连接底层信息与高层规约的谓词。

PEDL 规约脚本包含三个主要部分:导出事件和条件部分、监控对象声明部分、事件和条件定义部分并以 Mobscr 开头和 END 结尾。MEDL 规约脚本包含三个主要部分:导入事件和条件、复合事件和条件定义和反馈动作的定义,并以 Eventspec

开头和 END 结尾。两种规约的具体格式将在第 5 节最后的案例中体现。

## 4 嵌入式操作系统接口设计

为了获取相应事件的信息,例如进程的创建与调度、中断的产生等,需要对操作系统的实现进行一些扩充,以便把规约相关的事件的发生以及相关的信息通过监控接口输送出来;为了对操作系统的运行进行反馈调整,还需要定义相应的反馈接口。

当前相关的工作主要包括两个方面,即 FreeRTOS 的运行信息获取和反馈控制接口。

### (1) FreeRTOS 的运行信息获取。

目前本文使用的是 FreeRTOS 的 Win 32 模拟器,在 Windows 7 环境下运行,即由 Windows 7 操作系统在 PC 环境下模拟 FreeRTOS 嵌入式操作系统的独立运行。在此种运行模式下,内存的管理部分 FreeRTOS 交由 Windows 进行管理,因此本文目前考虑监控的时序性质规约暂不包括内存管理方面的性质。同时,监控软件是运行在 Windows 平台上的,因为在实际使用情况中,由于内置嵌入式操作系统的终端往往运算能力有限,进行验证计算的监控程序也往往独立运行在另外的终端上。如在航天领域中,常见的方式就是监控软件运行在地面,而航天器上的系统把运行信息保存在存储区中,再根据需要发送给地面系统。

在 FreeRTOS 中获取的相关信息先输出到一个日志文件中,再由监控程序读取使用。该日志是一种事件日志,用于记录相关原子事件的发生。为了在这些事件发生时把相应信息记录到日志文件中,需要对 FreeRTOS 内核进行修改,当前完成的信息获取的内容如下:获取系统任务的状态、获取时间信息、任务状态转换、优先级的变化、创建与删除任务。

获取方法是在已有的内核函数执行关键动作时增加用于记录信息到日志的代码。以创建和删除任务为例,创建任务由 API 函数 xTaskCreate() 负责,任务的删除由 API 函数 xTaskDelet() 负责,当创建和删除任务时会触发 Create task( $m$ ) 和 Destroy task( $m$ ) 事件。我们在 API 函数中进行修改,当创建和删除任务的时候输出任务名、优先级和事件发生的时间到事件日志。具体 FreeRTOS 的相关内核代码请参考文献[9]。

### (2) 反馈控制接口。

反馈接口用于在发现问题时,监控程序可以对

操作系统的运行进行干涉,例如停止某些任务、改变任务优先级、禁止某些中断等。在 FreeRTOS 中有一些 API 函数是用于改变相关对象状态,由于在运行时不能直接访问操作系统内核,所以需要设计一个可由监控程序调用的反馈接口,间接对操作系统的运行进行调整。当前反馈控制接口使用到 FreeRTOS 中 API 函数提供的调控能力能完成以下反馈控制:将调度器挂起、改变任务优先级、挂起任务、中断任务、删除任务。

具体使用到的 API 函数的相应信息需求请参考文献[9]。为了对操作系统的反馈控制使用一个统一的接口,本文在 FreeRTOS 中定义一个接口 Steer()。当运行的系统出现了违反规约的情况,监控程序根据预先设计好的反馈动作向 Steer()接口传输指令,每种操作指令都对应好要调用的 API 函数,当接到指令时 Steer()就能对相应任务调用相应的 API 函数。这样的设计使得监控程序只用完成与 Steer()接口的通信工作而不用自己直接调用 API 函数,而 Steer()又可以执行不同的监控程序的反馈请求,加强了通用性。

## 5 监控器的构造与实现

运行时验证的 LTL 公式的预测语义,也称三值语义<sup>[10]</sup>(简称 LTL3)。与已经被用于运行时验证工具的基于有穷轨迹上的两值语义(true/false)监控器相比,基于三值语义的监控器非常适合于嵌入式系统运行时验证。一方面,三值语义的公平性使得监控器的裁决始终是正确的,另一方面,三值语义的预测性使得监控器有发现一条无穷运行轨迹的最小好(坏)前缀<sup>[11]</sup>的能力,即监控器能尽可能早地作出裁决,因此在一定意义上具有预测性。所以,本文的监控器构造也将使用三值语义。三值语义监控器的理论构造过程请参照文献[12]。

基于三值语义的监控程序的构造过程如图 4 所示。

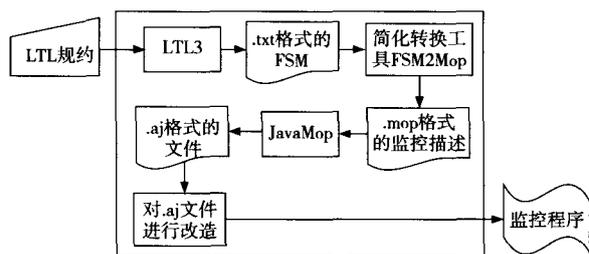


Figure 4 Generation process for monitoring program

图 4 监控程序生成流程图

在基于 LTL 三值语义的监控器构造的实现中,使用了一个开源工具 LTL3 Tools 用于生成监控器的有限状态机 FSM(Finite-State Machine)。LTL3 Tools 输出相应的 FSM 的 .txt 格式文本文件,如图 5 所示,可以看作是一个监控器模型,输入有穷字  $u$ ,它会判断出是否公式成立。通过将 .txt 文件中的 FSM 进行简化,去除冗余的边,并生成下一步 JavaMOP 所需的 .mop 文件,运行 JavaMOP 得到相应的 .aj 文件,当前正在编写相关的简化转换工具 FSM2Mop,用于自动将 FSM 生成下一步 JavaMOP 所需的 .mop 文件,避免人工带来的误操作。得到的代码已经是一个较为完整的监控器实现框架,但目前还不能完全自动执行,仍需要一些人工改造以加入相应的信息获取代码和调整代码,才能得到相应的监控程序,人工参与和调整代码可能会带来误操作,而造成监控程序的不准确,本文下一步将考虑如何把该过程完全自动化。同时,本文还实现了对日志进行读取的相应接口函数,用于读取日志内的数据。

```

digraph G {
  "(-1, 1)" -> "(-1, 1)" [label = "<empty>"];
  "(-1, 1)" -> "(-1, 1)" [label = "(init&&spawn)"];
  "(-1, 1)" -> "(-1, 1)" [label = "(init)"];
  "(-1, 1)" -> "(-1, 1)" [label = "(spawn)"];
  "(0, 0)" -> "(0, 0)" [label = "<empty>"];
  "(0, 0)" -> "(1, -1)" [label = "(init&&spawn)"];
  "(0, 0)" -> "(1, -1)" [label = "(init)"];
  "(0, 0)" -> "(-1, 1)" [label = "(spawn)"];
  "(1, -1)" -> "(1, -1)" [label = "<empty>"];
  "(1, -1)" -> "(1, -1)" [label = "(init&&spawn)"];
  "(1, -1)" -> "(1, -1)" [label = "(init)"];
  "(1, -1)" -> "(1, -1)" [label = "(spawn)"];
  "(1, -1)" [label="(1, -1)", style=filled, color=green];
  "(0, 0)" [label="(0, 0)", style=filled, color=yellow];
  "(-1, 1)" [label="(-1, 1)", style=filled, color=red];
}

```

Figure 5 Instance for monitor automata

图 5 生成的监控器自动机模型的图形和文字表示示例

这里例举一个具体的案例:系统中一个任务  $a$  开始运行时,当前系统中还有另外的任务  $b$  处于等待运行的状态,任务  $a$  开始运行到完成期间,任务  $a$  不能被中断,且任务  $b$  的优先级不能高于任务  $a$ ,发现违反规约的情况后就反馈控制挂起任务  $b$ ,直到任务  $a$  完成运行。

该案例首先可以用 LTL 公式  $G(SM \rightarrow KO)$  表示,其中复合条件  $SM$ (Start Mission)表示任务开始运行,复合条件  $KO$ (Keep Operating)表示保持运行状态,公式中  $G$  表示 always。针对 LTL 公式  $G(SM \rightarrow KO)$ ,首先定义复合条件  $SM = Start\_a$ ,

其中  $Start\_a = Start\ task(a)$  是原子事件,表示输出字符的任务  $a$  开始运行,再定义复合条件  $KO = (! (Block\_a) \&\& Priority) \cup End\_a$ , 其中  $Block\_a = Start\ block(a)$  为原子事件,表示任务  $a$  中断,  $Priority$  是一个原子条件,表示任务  $b$  的优先级不能高于任务  $a$ ,  $End\_a$  为一个原子事件,表示任务  $a$  结束运行。案例生成的 PEDL 与 MEDL 规约如图 6 所示。根据 LTL 公式,由上文所述过程即可得到相应的监控程序,结合添加了相应接口的 FreeRTOS 嵌入式操作系统以及接口程序,在该案例中监控程序能够及时、准确地发现系统运行时性质规约违背的情况,并给出警报,进行反馈控制。

```

Monscr Output
export event    Start_a,End_a,Block_a;
export condition Priority ;

Monobj task.a task.b;
task.a.priority task.b.priority;

event Start_a=Start task(a);
event End_a=End task(a);
event Block_a=Start block(a);
condition Priority=a.priority>b.priority;

END

```

```

Repscr Output
import event    Strat_a,End_a,Block_a;
import condition Priority;

condition SM=Start_a;
condition KO=((!Block_a)&&Priority) U End_a;
tempeoral condition= G(SM→KO);

steering suspend=vTaskSuspend(h);

END

```

Figure 6 PEDL and MEDL statute of usecase

图 6 案例的 PEDL 和 MEDL 规约

## 6 结束语

本文在传统的运行时验证技术的基础上,结合 FreeRTOS 嵌入式操作系统提出了一种针对嵌入式操作系统而不是应用程序的运行时验证框架,并对相关的具体工作进行了初步实现。

当前对嵌入式操作系统的运行时验证框架的实现还处于初步阶段,今后工作的重点主要在:构造监控的对象更加全面,反馈的动作更多,并能够像 Java-MaC 工具一样自动生成相应的监控程序的工具集;结合其他工作让 FreeRTOS 构成一个功能更全面、能够满足实际应用的带运行时验证技术的嵌入式系统,使其有更广泛的用途。另外,当前运行时验证技术最大的瓶颈就是验证计算本身对系统的损耗,对于嵌入式操作系统这种实时性要

求很高的系统更是一个关键问题。下一步,作者还将研究使用运行时验证对系统的影响有多大,影响的因素是什么,通过什么样的技术手段使得这种影响降低到最小的程度。

### 参考文献:

- [1] Colin S, Mariani L. Run-time verification, chapter 18 [J]. Proc of LNCS, 2005, 3472:525-555.
- [2] Peleska J. Test automation for safety-critical systems: Industrial application and future developments[C] // Proc of the 3rd International Symposium of Formal Methods, 1996, 1051:39-59.
- [3] Clarke E M, Grumberg O, Peled D A. Model checking[M]. London: The MIT Press, 1999.
- [4] Liu Bin, Wang Qi, Liu Li-li. Principle and implementation of embedded operating system FreeRTOS[J]. Microcontroller and Embedded System, 2005(7): 1-2. (in Chinese)
- [5] Kroger F. The temporal logic of programs[M]. New York, Springer-Verlag, 1987.
- [6] Sammapun U, Lee I, Sokolsky O. RT-MaC: Runtime monitoring and checking of quantitative and probabilistic properties[C] // Proc of the 11th IEEE International Conference of Embedded and Real-Time Computing Systems and Applications, 2005: 147-153.
- [7] Kim M, Kannan S, Lee I, et al. Java-MaC: A runtime assurance approach for Java programs[J]. Formal Methods in Systems Design, 2004, 24(2): 129-155.
- [8] MaC Research team of Univeristy of Pennsylvania. Languages in the MaC prototype implementation[EB/OL]. [2008-01-01]. <http://rtg.cis.upenn.edu/mac/index.php3>.
- [9] Barry R. FreeRTOS user manual[EB/OL]. [2004-12-09]. <http://www.FreeRTOS.net>.
- [10] Geilen M C W. On the construction of monitors for temporal logic properties [J]. Electronic Notes in Theoretical Computer Science, 2001, 55(2): 181-199.
- [11] Bauer A, Leucher M, Schallhart C. Runtime verification for LTL and PTLTL[J]. Journal of ACM Transactions on Software Engineering and Methodology, 2011, 20(4): Article No. 14.
- [12] Sui Ping. Software runtime verification method based on three-valued semantics[D]. Changsha: National University of Defense Technology, 2010. (in Chinese)

### 附中文参考文献:

- [4] 刘滨, 王琦, 刘丽丽. 嵌入式操作系统 FreeRTOS 的原理与实现[J]. 单片机与嵌入式系统应用, 2005(7): 1-2.
- [12] 隋平. 基于三值语义的软件运行时验证方法. [D]. 长沙: 国防科学技术大学, 2010.

### 作者简介:



张可迪(1988-),男,黑龙江哈尔滨人,硕士生,研究方向为软件工程。E-mail: Zkd008@21cn.com

ZHANG Ke-di, born in 1988, MS candidate, his research interest includes software engineering.