软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(2):400−418 [doi: 10.13328/j.cnki.jos.004540]

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

信息物理融合系统的时间需求一致性分析*

尹 玲, 陈小红, 刘 静

(上海市高可信计算重点实验室(华东师范大学),上海 200062)

通讯作者: 陈小红, E-mail: xhchen@sei.ecnu.edu.cn

摘 要: 信息物理融合系统(cyber-physical system,简称 CPS)蕴藏着巨大的潜在应用价值.时间在 CPS 中起到非常重要的作用,应该在需求早期阶段明确.提出了一个基于逻辑时钟的 CPS 时间需求一致性分析框架.首先,构建了 CPS 软件的时间需求概念模型,提供时间需求和功能需求的基本概念,并给出了概念模型的形式化语义;然后,在模型制导下,从CPS的交互环境特性和约束中提取出其软件时间需求规约.基于形式化语义,定义了时间需求规约的一致性特性.为了支持形式化验证,将时间需求规约转换成 NuSMV 模型,用 CTL 公式表述要检测的特性,并使用 NuSMV 工具实施了一致性检测.

关键词: 信息物理融合系统;需求工程;时间需求建模;一致性检测;形式化验证

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 尹玲,陈小红,刘静.信息物理融合系统的时间需求一致性分析.软件学报,2014,25(2):400−418. http://www.jos.org.cn/1000-9825/4540.htm

英文引用格式: Yin L, Chen XH, Liu J. Consistency analysis of timing requirements for cyber-physical system. Ruan Jian Xue Bao/Journal of Software, 2014,25(2):400−418 (in Chinese). http://www.jos.org.cn/1000-9825/4540.htm

Consistency Analysis of Timing Requirements for Cyber-Physical System

YIN Ling, CHEN Xiao-Hong, LIU Jing

(Shanghai Key Laboratory of Trustworthy Computing (East China Normal University), Shanghai 200062, China)

Corresponding author: CHEN Xiao-Hong, E-mail: xhchen@sei.ecnu.edu.cn

Abstract: Cyber-Physical Systems (CPSs) have great potentials in several application domains. Time plays an important role in CPS and should be specified in the very early phase of requirements engineering. This paper proposes a framework to model and verify timing requirements for the CPS. To begin with, a conceptual model is presented for providing basic concepts of timing and functional requirements. Guided by this model, the CPS software timing requirement specification can be obtained from CPS environment properties and constraints. To support formal verification, formal semantics for the conceptual model is provided. Based on the semantics, the consistency properties of the timing requirements specification are defined and expressed as CTL formulas. The timing requirements specification is transformed into a NuSMV model and checked by this well-known model checker.

Key words: cyber-physical system; requirement engineering; timing requirement modeling; consistency checking; formal verification

信息物理融合系统(CPS)[1]是计算过程和物理过程的融合系统.它强调信息世界与物理世界的融合,强调对交互环境的实时监测与控制.它通过与交互环境的实时交互来增加或扩展新的功能,以安全、可靠和实时的方式检测或者控制一个物理实体[2].实际上,CPS 核心是 3C(computation,communication,control)融合,自主适应交互环境的变化.从其交互环境的角度来看,其功能的关键是感知与控制.而感知与控制对时间非常敏感,因此,时间需求在 CPS 中非常重要.由此,我们提出对 CPS 的时间需求进行建模与验证.

* 基金项目: 国家自然科学基金(61202104, 91318301, 61332008, 61170084); 教育部博士点基金(20120076120016); 上海市知识服务平台项目(ZF1213)

收稿时间: 2013-05-08; 修改时间: 2013-09-29; 定稿时间: 2013-12-05

需求存在于环境之中[3],CPS 的时间需求也来自于其交互环境.所谓交互环境,指的是所有将要与 CPS 软件交互的实体.由于 CPS 软件应用的广泛性,其交互环境实体多样,可以是自然环境、建筑、机器、物理设备等等,当然也包括人类自身.这样的环境可以被实时感知,部分(如物理实体)还可以被控制.这些异构的环境实体导致交互出现多样性,使得时间度量本身可能存在多种形态(比如用距离、角度来衡量时间).而且由于交互是实时交互,正是通过交互才实现了 CPS 的功能,因此,CPS 的时间需求的建模有 3 个要求:1) 必须建立在功能性需求之上,与功能性需求无缝地联系在一起;2) 在需求模型中,时间必须被显式地建模,与功能性需求的基本概念类似,时间必须作为第一类概念存在;3) 时间应该存在多种形态,且能够被定性与定量地描述.另外,由于 CPS 功能表现为其对交互环境的感知与控制,因此在需求模型的选择中,其功能性需求必须与物理实体的变化有关.

目前,代表性的需求建模方法有基于目标的方法[4]、基于主体与意图的方法[5,6]和基于问题框架的方法[3,7].面向目标的方法以目标为需求来源和依据,在时间需求建模上,它使用带类型的一阶时序逻辑作为语言,包含传统的时序算子和描述涉及实时节点特性的附加实时算子.面向主体和意图的方法提供一种基于组织层次上下文的需求获取和建模的思路,其建模理念为刻画有目的的参与者.它们也采用带类型的一阶时序逻辑语言来描述时间需求.这些时间需求工作都基于时序逻辑,可以描述定性约束.为了描述定量约束,人们进行了各种扩充,如 Timed Computation Tree Logic[8]等.它们通常用于表示实时系统的性质和规范,在需求工程中用到较多的是一阶时序逻辑和谓词时序逻辑.由于目标和有目的的参与者等与物理实体、交互之间都不存在直接关系,因此,基于目标的方法和基于主体与意图的方法都不适合建模 CPS.

与上述需求建模方法不同,问题框架方法认为,软件系统对现实世界的作用是软件问题的来源,对软件系统将与现实世界(交互环境)发生的作用进行结构化分析是需求分析的切入点.问题框架方法强调需要对软件系统将要作用的现实世界进行刻画,并将需求的含义指称到对现实世界相关领域的描述上.其建模的基本概念是现实世界中的问题领域(环境实体)以及未来软件系统与领域的交互.

我们认为,基于问题框架的方法适合对 CPS 进行功能需求建模,因为问题框架的方法可以用交互与其引起的物理环境变化描述功能.基于问题框架的方法,我们曾结合 MARTE(modeling and analysis of real-time and embedded systems)[11]及 CCSL(clock constraint specification language)[12]做过一些时间需求建模工作[9,10].基于这些工作,本文将逻辑时钟引入问题框架方法,提出构建 CPS 时间需求一致性分析框架.

逻辑时钟特别适合建模 CPS 时间,它有如下好处:1) 逻辑时钟是一个抽象概念,可以表述和测量事件的有序多次发生.一个事件可以对应为一个逻辑时钟,事件的每次发生看作该时钟的一个时间点(或者说该时钟 tick 了 1 下).2) 逻辑时钟建模中重要的是时间点之间的序关系.一个时钟相邻时间点之间的物理时间间隔可以不同.我们也可以约束某一时钟,使其每次 tick 之间间隔相同,在此意义上,物理时钟可以看作逻辑时钟的一个特例.3) 它不限于常用的物理时间(比如手表时间),还可以由用户来定义,关联到任何实体上(例如温度、距离、角度等),支持多形态时间.

从上述描述中我们可以看出,将逻辑时钟引入问题框架方法来建模 CPS 的时间需求非常合适:功能性需求的基本概念问题领域和交互都可以与时钟关联起来,不同的问题领域可以建立不同的逻辑时钟,其时间点就是交互的每次发生.这样,时间需求就与功能性需求无缝地联系在一起.因此,本文提出引入逻辑时钟到问题框架的方法中,为 CPS 时间需求进行建模,利用时钟之间的关系定义 CPS 的交互环境约束,由此提取出软件时间需求规约.由于我们以前的工作[10]重点介绍了多形态时间,因此这不是本文的侧重点.本文重点建模时间需求的定性与定量关系,其中,定性的关系包括“时间点 a 发生在时间点 b 之前”之类的描述,定量的关系包括“时间点 a 发生时间点 b 之前,并且之间间隔等于(大于、小于)某一时钟的 t 个时间单位”这样的描述.

为了获得一致的时间需求规约,我们首先给出了其时间需求的概念模型,给出了其形式化语义以精确描述时间需求,定义了一系列定性和定量的时钟关系.在此基础上,定义了问题领域级别上的时钟构造算子,由此来构造系统级别的时间需求规约.以形式化语义为基础,我们定义了一致性属性,并将其表达为时态逻辑 CTL 公式.由于其语义为由标号迁移系统给出的状态变迁模型,本文选择使用 NuSMV——一个经典的模型检测工具,来验证时间需求规约的一致性.

本文第 1 节简单介绍背景知识,包括问题框架方法、问题图、MARTE/CCSL 以及模型检测器 NuSMV.第 2 节定义时间需求的概念模型.第 3 节定义其形式化模型.第 4 节给出时钟构造算子以构造更高级别的时钟.第 5 节验证时间需求规约的一致性.第 6 节给出建模和验证过程,并用一个简单案例来展示方法的可行性.第 7 节比较相关工作,总结全文并提出进一步的工作.

1 预备知识

1.1 问题框架方法/问题图

问题框架方法是一个经典的需求工程方法[3,7],用来建模、分析和结构化现实世界的需求问题.问题图是其功能性需求的描述模型,其基本元素如下:

- 机器,如图 1 所示的 Controller machine(缩写为 CM),形状为一个带双竖线的矩形框,表示要开发的软件.
- 问题领域,如图 1 所示的 Device,形状为一个矩形框,表示机器将要交互的实体.
- 需求,如图 1 所示的 Work regime,形状为虚线椭圆框,表示文字描述的需求.
- 行为交互,如图 1 所示的 CM!{on,off},形状为为实线,表示机器与问题领域之间的交互,其中,CM!表示为 CM 发起,on,off 表示现象.
- 期望交互,如图 1 所示的{is_on,is_off},形状为虚线箭头,表示在问题领域上期望看到的现象.

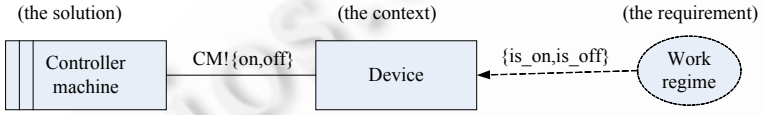

Fig.1 A simple problem diagram

图 1 一个简单的问题图示例

1.2 MARTE/CCSL

MARTE 是 OMG 组织认可的实时嵌入式系统建模语言[11,12],CCSL 是在其上开发的时钟约束语言.CCSL 提供了一些元素来表达时间和时间约束,其时间的基本元素为时钟.时钟为一个五元组,如定义 1 所示.

定义 1. 时钟(*clock*):

$$Clock\triangleq\langle I,\prec,D,\lambda,u\rangle.$$

其中,I 是时间点集;$\prec$是 I 上的偏序关系,命名为严格先于;D 是一组标签;$\lambda:I\to D$ 是一个标签函数;u 是一个符号,代表时间单位.时钟可以是物理时钟,也可使是逻辑时钟.物理时钟参考物理时间,逻辑时钟参考逻辑时间.一个逻辑时钟是由一组离散的有序时间点组成,这些时间点为事件出现的时刻.当时钟 *tick* 时,表示事件发生一次,可以用时序+1 表示.因为它离散,可以通过自然数对 I 中的元素进行排序.对于任意的离散时钟 $Clock\triangleq\langle I_C,\prec_C,D_C,\lambda_C,u_C\rangle$,$C[k]$表示 I_C 中的第 k 个时间点.下文中,在不会引起歧义的情况下,我们省略下标 C.

两个时间点之间可以存在如下的时序关系:

- 同步(coincidence,$\equiv\triangleq\preceq\cap\succeq$).它是一个很强的关系,强迫时间点发生的同步性.它具有对称性,也就是说,$\forall a,b\in Instant$,如果 $a\equiv b$,那么 $b\equiv a$.
- 严格先于(strict precedence,$\prec\triangleq\preceq\setminus\equiv$).它表示时间点发生的严格时序关系.它有传递性,也就是说,$\forall a,b,c\in Instant$,如果 $a\prec b,b\prec c$,那么 $a\prec c$.
- 先于(precedence,$\preceq$,等价于($\prec\equiv$).它表示时间点发生的时序关系,具有传递性和自反性.
- 排除(exclusion,$\#\triangleq\prec\cup\succ$).它表示这两个时间点永远不会在同一时刻发生.

1.3 NuSMV

NuSMV 是一个经典的得到广泛应用的模型检测工具[13],可以验证有限状态系统是否满足用时态逻辑公式

描述的性质.它支持多种建模方式,包括有限状态自动机(FSM).

FSM 方式建模系统的基本概念有:一个系统包括多个模块(module),每个模块由模块名和模块定义组成,模块定义由形参(parameter)和主体(body)部分构成.每个模块主体定义一个 FSM,变量声明在(VAR)部分,用来建模 FSM 中的状态,初始化约束(INIT constraint)决定 FSM 的初始状态,迁移约束(TRANS constraint)用命题公式描述 FSM 中状态迁移关系(状态变量的当前值及下一个状态的值).一个系统中要有一个 main 模块,且 main 模块不能有形参.通常,在 main 模块中实例化并调用需要的模块,这些模块实例描述的 FSM 的同步乘积(synchronous product)构成该系统的 FSM.

下面是 NuSMV FSM 模型的一个小例子,它包括两个 MODULE:

- 一个是 main 模块声明两个布尔变量,初始化为假,并创造了另一个模块 *subClock* 的一个实例;
- 模块 *subClock* 有两个参数,它的迁移约束限制每个状态下这两个参数的赋值只有 3 种情况:1) 都为真;2) 左边参数为假,右边为真;3) 都为假.

```
MODULE main
  VAR
    c1:  boolean;
    c2:  boolean;
    ctr1: subClock(c1,c2);
  INIT
    c1=FALSE & c2=FALSE;
MODULE subClock(left,right)
  TRANS
    (next(left)=TRUE & next(right)=TRUE)|(next(left)=FALSE & next(right)=TRUE)
      |(next(left)=FALSE & next(right)=FALSE);
```

2 时间需求概念模型

我们以前的工作[14]构建了一个功能性需求的概念模型.由于该功能性需求基于交互环境的变化,它适用于 CPS 系统.本文将这个模型进行扩展,加入与功能相关的时间概念,由此得到一个时间需求概念模型,如图 2 所示.

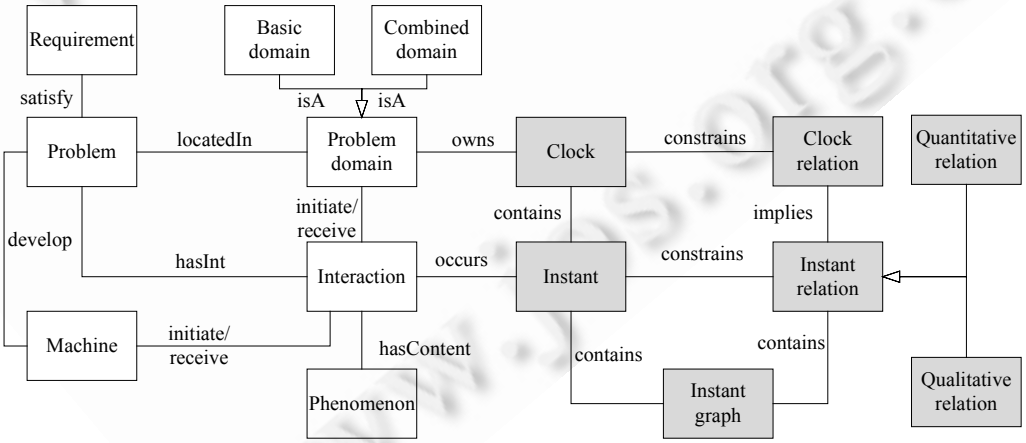

Fig.2 Timing requirement conceptual model

图 2 时间需求概念模型

在图 2 中,

- 白色的矩形框及其关联表示了功能需求的基本概念及其关联,它表示了基于交互环境变化功能性需

求的基本概念:一个问题(problem)位于一组现实世界的问题领域(problem domain)中,其目的就是要开发一个机器(machine)来满足需求(requirement).其中,问题就是要由软件开发完成的任务,需求是需要完成的功能,机器则是要开发的软件,问题领域就是将要与机器交互的现实世界实体.一个问题领域可以是基本领域(basic domain),也可以是组合领域(combined domain).基本领域是一个现实世界实体,而组合领域则是可以由若干基本领域组合而成的领域.在机器与问题领域之间共享的现象,称为交互(interaction).

- 灰色矩形表示与时间相关的概念,包括时间点(instant)、时间点关系(instant relation)、时钟(clock)以及时钟关系(clock relation).时间点就是交互发生的时刻,它们之间的关系称为时间点关系.时钟是用来显示、协调时间的一个可视工具,系统时间可以看作一组时钟的集合.需要指出的是,时钟使用的是逻辑时钟,每个时钟拥有一组有序关系的时间点.为了可视地表示时间点及时间点的关系,我们定义时间点图(instant graph)为一个有向图,其结点是时间点,边为时间点关系.根据是否带有数量,时间点关系可以分为定性(qualitative relation)与定量关系(quantitative relation).定性关系主要是 MARTE/CCSL 中的 4 种关系,分别为 coincidence,strict precedence, precedence 和 exclusion.这些关系描述了时间点发生的先后顺序关系.至于两个时间点之间的定量关系,该量化数值必须基于某一时钟的度量,这称为参考某个时钟.若两个或多个时钟的多个时间点之间存在关系,则时钟之间存在时钟关系.

上述两组概念通过如下的关联联系在一起:每个问题领域都拥有(或参考)某一个时钟,问题领域的每个交互发生的时刻都是该时钟对应的一个时间点.

3 时间需求形式化描述

本节拟给出时间需求概念模型中概念的语法与操作语义.由于概念时钟关系内容较多,所以分为基本概念与时钟关系两个小节进行描述.本文中的操作语义由标号迁移系统(labeled transition system,简称 LTS)[15]给出.LTS 是一种传统的广为接受的基于抽象状态机的状态变迁模型,一个 LTS 由一个四元组(S,s_0,A,T)组成,分别代表系统状态集合、初始状态、动作标号和迁移关系.LTS 可以直观地用图形表示,状态表示为圈,迁移表示为圈之间的有向箭头,标号标于对应箭头上.为了方便理解,时间点关系、时钟关系和下一节中的时钟构造算子的操作语义都用 LTS 图形式给出.

3.1 基本概念

根据概念模型,问题领域可以由它的身份标识和问题描述定义,因此我们给出如下定义:

定义 2. 问题领域(*pd*):

$$pd\triangleq\langle id,pdescription\rangle.$$

其中,*id* 是问题领域的标识,*pdescription* 是问题领域的描述.

交互是问题领域与机器之间共享的现象,因此交互可由发起方、接收方及共享内容定义:

定义 3. 交互(*interaction*):

$$Interaction\triangleq\langle id_{int},initiator,receiver,hasContent\rangle.$$

其中,id_{int} 是交互的标识;*initiator* 是现象的发起方,可以是问题领域(或机器);*receiver* 是现象的接受方,可以是机器(或问题领域);*hasContent* 是接受方和发起方共享的现象.

由于时钟与问题领域关联在一起,因此本文将时钟定义为 $d.C$ 的形式,其中,d 表示问题领域,C 表示与它相关的时钟.由于本文重点不在多形态时钟,因此在定义 1 的基础上,将时钟定义简化为一个二元组:

定义 4. 时钟($d.C$):

$$d.C\triangleq\langle I,\prec\rangle.$$

其中,I 是领域 d 的所有交互发生时刻的时间点集;$\prec$是 I 上的偏序关系,也称为严格先于.我们定义函数 $idx:I\rightarrow\mathbb{N}^*,\mathbb{N}^*=\mathbb{N}\backslash 0$ 用来取得时间点在时钟的时间点集合中的编号,对$\forall i\in I,idx(i)=k$ 当且仅当 i 在 I 中的编号为 k.

在此基础上,将整个时间系统定义为一组时钟的集合,标记为 $T=\{d_1.C_1,d_2.C_2,\ldots,d_n.C_n\}$,其所有的时间点集合标记为 TI.不同问题领域的时钟可能不具有可比性,它们有不同的 tick 速率和不同的形式.在一个系统当中,不一定有全局时钟.

不同交互发生的时间约束可以通过对应的时间点之间的关系来衡量,我们定义时间点关系.

定义 5. 时间点关系(*TR*)包含两种关系:定性关系(*IL*)与定量(*IT*)关系.

- $IL:TI\times TI$

  定性关系 *IL* 与 MARTE 中定义的 4 种关系相同,即,包括 coincidence,strict precedence,precedence 和 exclusion.

- 定量关系 *IT* 由定量算子 *op* 给出.由于定量关系需要考虑量度问题,所以 *op* 与时钟密切相关,其定义如下:

$$C_2[j]-C_1[i]\ op\ k, k\in\mathbb{N}^*, op\in\{=,>,\geqslant,<,\leqslant\}\text{当且仅当}$$
$$\exists m,n\in\mathbb{N}^*, C_1[i]\equiv C_3[m], C_2[j]\equiv C_3[n], idx_{C_3}(n)=(idx_{C_3}(m)+1)\ op\ k,$$

其中,$idx_{C_3}(n)$ 表示时钟 C_3 的时间点 n 在 I_{C_3} 中的编号.该定量关系约束时钟 C_1 的第 i 个时间点与时钟 C_2 的第 j 个时间点之间的区段中时钟 C_3 *tick* 等于(或大于、大于等于、小于、小于等于)k 下,其 LTS 语义如图 3 所示.在图 3 中,Φ为一特殊标号,表示没有时钟 tick,它包含 3 个状态:状态 0、状态 1 和状态 2.当约束处于状态 0 时,时间点 $C_1[i]$可以发生,它的发生使得算子进入状态 1,当算子处于状态 1 时,时间点 $C_2[j]$才可以发生.这保证了时间点 $C_1[i]$先于 $C_2[j]$的时序关系.由于该算子只描述这两个时间点之间的关系,当两个时间点都发生后,该算子不再约束,状态 2 只有到自身的Φ变迁.

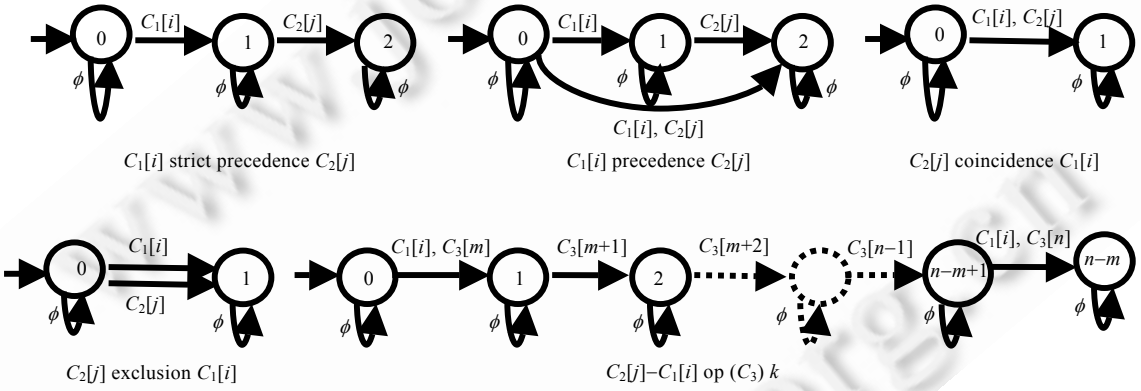

Fig.3 LTS semantics of instant relations

图 3 时间点关系的 LTS 语义

3.2 时钟关系

在 CPS 这样的复杂系统中,交互可能无数次发生,只定义其中一部分时间点关系远远不够.因此,我们尝试在问题领域级别的时钟上建立关系,即,在时钟层面定义关系来描述和反映时钟之间时间点的对应关系.

时钟之间的关系由时间点之间的关系来定义.时间点关系有定性与定量之分,因此,我们将时钟关系也分为定性关系与定量关系.定性关系可以包括很多,用户也可以自己定义,本文仅仅给出 3 种比较常用的关系:*subClock*,*fasterThan* 和 *alternate*.

- 定性关系 1.$d_1.C_1$ *subClock* $d_2.C_2$ 是子时钟关系,它表示在一个时钟 $d_1.C_1$ 的时间点发生的同时,其父时钟 $d_2.C_2$ 一定有对应的时间点发生.两个时钟对应的问题领域应该也具有父子关系.该关系成立,如果 $\exists h: I_{d_1.C_1}\rightarrow I_{d_2.C_2}$ 满足如下关系:

  1) h 是单射;

2) h 单调: $\forall i,j\in I_{d_1.C_1},(i\prec_{d_1.C_1} j)\Rightarrow(h(i)\prec_{d_2.C_2} h(j))$;

3) 时钟 $d_1.C_1$ 的每个时间点与其对应的 $d_2.C_2$ 的时间点同时发生: $\forall i\in I_{d_1.C_1}, i\equiv h(i)$.

图 4 给出了 $d_1.C_1$ *subClock* $d_2.C_2$ 的 LTS 的语义,其只有 1 个状态,表示在任一时刻(同一条迁移上),要么两个时钟都 tick,要么父时钟 tick 子时钟不 tick,要么两个时钟都不 tick.

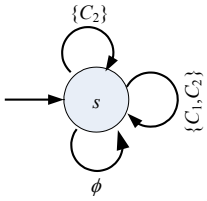

Fig.4 LTS semantics of $d_1.C_1$ *subClock* $d_2.C_2$

图 4 $d_1.C_1$ *subClock* $d_2.C_2$ 的 LTS 语义

- 定性关系 2.$d_1.C_1$ *fasterThan* $d_2.C_2$ 表示时钟 $d_1.C_1$ 的第 i 个时间点要先于时钟 $d_2.C_2$ 的第 i 个时间点发生. 它有两个版本:严格的 *strictPre* 和非严格的 *nstrictPre*,该关系成立,如果 $\exists h: I_{d_1.C_1}\to I_{d_2.C_2}$ 满足如下条件:

  1) h 是单射;

  2) h 单调: $\forall i,j\in I_{d_1.C_1},(i\prec_{d_1.C_1} j)\Rightarrow(h(i)\prec_{d_2.C_2} h(j))$;

  3) 时钟 $d_2.C_2$ 的每个时间点与其对应的 $d_1.C_1$ 的时间点有序:

     - $\forall i,j\in I_{d_1.C_1}, h(i)\prec i$ (严格的);
     - $\forall i,j\in I_{d_1.C_1}, h(i)\preceq i$ (非严格的).

$d_2.C_2$ 的每个时间点可以发生的条件是,该时间点对应的 $d_1.C_1$ 的时间点已经发生过了.为了确保 $d_2.C_2$ 不过早 tick,我们需要监控 $d_1.C_1$ 已经提前(相对于 $d_2.C_2$)tick 了多少下,所以在其 LTS 语义中(如图 5 所示),我们记录两个时钟已发生过的时间点的个数的差值δ=$idx(d_1.C_1)$–$idx(d_2.C_2)$,不同的状态对应于不同的δ数值.由于 $d_1.C_1$ 比 $d_2.C_2$ 快,$\delta\geqslant 0$.算子所处的状态(即当时δ的值)决定下一个时刻哪个时钟可以 tick.例如严格版本中,当δ=0 时,只有 $d_1.C_1$ 可以 tick,那么当 $d_1.C_1$ tick 而 $d_2.C_2$ 不 tick 时,差值δ加 1,算子状态迁移到状态 s_1.以此类推,算子在状态 s_1(δ=1)表示 $d_1.C_1$ 已经比 $d_2.C_2$ 多 tick 了一下,那么接下去的时刻,可以 $d_1.C_1$ 单独 tick 使得δ加 1;或者 $d_2.C_2$ 单独 tick 使得δ值减 1;或者 $d_1.C_1$ 和 $d_2.C_2$ 都 tick,那么δ值不变;或者两个时钟都不 tick,当然δ值也不变.

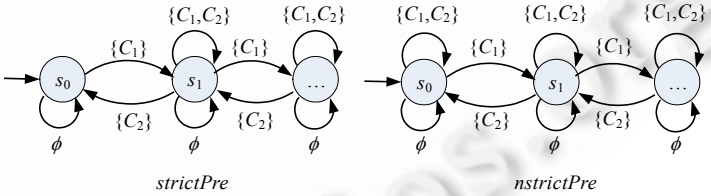

Fig.5 LTS semantics of $d_1.C_1$ *fasterThan* $d_2.C_2$

图 5 $d_1.C_1$ *fasterThan* $d_2.C_2$ 的 LTS 语义

- 定性关系 3.$d_1.C_1$ *alternate* $d_2.C_2$ 表示时钟 $d_1.C_1$ 和 $d_2.C_2$ 交替 tick,从左边的时钟开始,该关系成立,如果 $\exists h: I_{d_1.C_1}\to I_{d_2.C_2}$ 满足:

  1) h 是单射;

  2) h 单调: $\forall i,j\in I_{d_1.C_1},(i\prec_{d_1.C_1} j)\Rightarrow(h(i)\prec_{d_2.C_2} h(j))$;

  3) 两时钟时间点交替发生: $\forall i,i'\in I_{d_1.C_1}, idx_{d_1.C_1}(i')=idx_{d_1.C_1}(i)+1, i\prec h(i)\prec i'$.

其 LTS 语义如图 6 所示.初始时只有 $d_1.C_1$ 可以 tick.它 tick 后,算子进入状态 s_1,$d_1.C_1$ 将不可以连续 tick,只有 $d_2.C_2$ 可以 tick,使算子回到状态 s_0.如此往复.其运行序列将为 $d_1.C_1$;$d_2.C_2$;$d_1.C_1$;$d_2.C_2$;…

定量关系表示两个时钟的时间点之间存在某些量化关系.定量关系也很多,用户也可以自己定义.本文仅给出最常用的 *boundedDiff*(*i*,*j*).

- 定量关系 1.$d_1.C_1$ *boundedDiff*(*i*,*j*) $d_2.C_2$ 表示这两个时钟的时间点之间的时间差在整数闭区间[*i*,*j*]之内,*i* 为负整数,*j* 为正整数.可以把 *boundedDiff* 看作 *fasterThan* 的一个扩展,限定了差值δ的边界,但不规定哪个时钟更快,如图 7 所示:当下届到达时,右边的时钟不能在下一个时刻单独 tick;而当上界到达时,左边的时钟不能在下一时刻单独 tick.

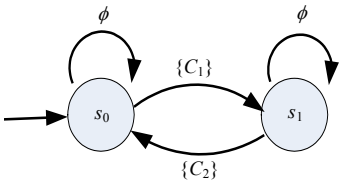

Fig.6 LTS semantics of $d_1.C_1$ *alternate* $d_2.C_2$

图 6 $d_1.C_1$ *alternate* $d_2.C_2$ 的 LTS 语义

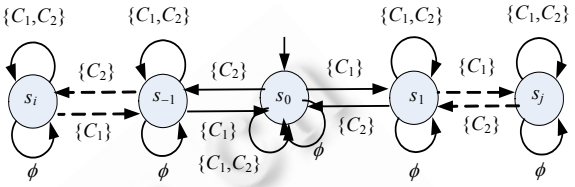

Fig.7 LTS semantics of $d_1.C_1$ *boundedDiff*(*i*,*j*) $d_2.C_2$

图 7 $d_1.C_1$ *boundedDiff*(*i*,*j*) $d_2.C_2$ 的 LTS 语义

4 时钟构造算子

在已定义的时钟基础上,为了进一步构造问题领域级的时钟及系统级时钟,本节拟定义一些时钟构造算子.由于我们的模型的时钟与问题领域相关,时钟构造算子可以分为两类:一类针对单问题领域,另一类针对组合问题领域.

4.1 单问题领域时钟算子

这种情况发生在一个问题领域的某些交互周期性发生的时候,这个领域(或者子领域)对应的新时钟就可以通过二进制字符(binary word)当前位(0 或 1)选择所有交互的一部分发生而构造(0 表示舍弃,1 表示保留),即,过滤出时钟的一部分时间点,这种算子称为 *filteredBy*.

定义 6. *filteredBy*:

$$d.C \triangleq d_1.C_1 \blacktriangle \omega.$$

根据已有时钟 $d_1.C_1$ 和二进制字符ω创建新时钟 $d.C$,该关系成立,如果 $\exists h: I_{d_1.C_1} \to I_{d_2.C_2}$ 满足如下条件:

1) h 是单射;
2) h 单调: $\forall i,j \in I_{d_1.C_1}, (i \prec_{d_1.C_1} j) \Rightarrow (h(i) \prec_{d_2.C_2} h(j))$;
3) 时钟 $d_1.C_1$ 的任一时间点与其对应的 $d.C$ 的时间点同时发生:

$$\forall i,i' \in I_{d_1.C_1}, h(i) \equiv i, idx_{d_1.C_1}(h(i)) = \omega \uparrow idx_{d_1.C_1}(i).$$

其中,$\omega\uparrow k$ 表示ω第 k 个“1”的位置,当时钟 $d_1.C_1$ tick 并且ω的当前位(current bit)为“1”时,时钟 $d.C$ tick.通常,ω写作 $u(v)^\omega$,u 为初始部分,v 为重复部分.这时,$d.C$ 被创建为 $d_1.C_1$ 的周期性时钟,周期为 v 的位数.

例如,$d.C \triangleq d_1.C_1 \blacktriangle 0(01)^\omega$的 LTS 语义如图 8 所示:在 $d_1.C_1$ 第 1 个时间点 tick 之后,算子进入状态 s_1,之后的运行序列将为 $d_1.C_1; d.C, d_1.C_1; d_1.C_1; d.C, d_1.C_1; \ldots$图 9 是其工作原理的直观展示.

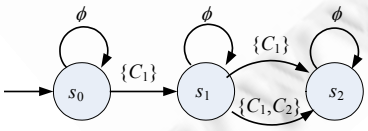

Fig.8 LTS semantics of $d.C \triangleq d_1.C_1 \blacktriangle 0(01)^\omega$

图 8 $d.C \triangleq d_1.C_1 \blacktriangle 0(01)^\omega$的 LTS 语义

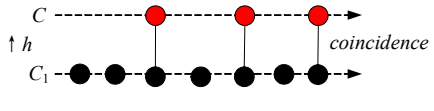

Fig.9 Instant graph of $d.C \triangleq d_1.C_1 \blacktriangle 0(01)^\omega$

图 9 $d.C \triangleq d_1.C_1 \blacktriangle 0(01)^\omega$的时间点图

4.2 组合领域时钟算子

在已知两个或多个子领域的时钟情况下,如何获得组合领域的时钟,与领域的组合相关.根据领域结构的不同,问题领域的组合可以分为两种类型:一种为相同结构领域组合,另一种为不同结构领域组合.所谓相同结构领域,指的是这些领域与机器之间共享相同的现象.

首先,若两个问题领域结构完全不同,则二者的组合对应的时钟时间点集应该是二者的时间点集的并,称为*union*,其具体定义见定义 7.

定义 7. *union*:

$$d.C \triangleq d_1.C_1 \ union \ d_2.C_2.$$

根据已有时钟 $d_1.C_1$ 和 $d_2.C_2$ 创建时钟 $d.C$.d_1,d_2 为时钟 C_1,C_2 所属不同的问题领域,d 为它们的并.每次 $d_1.C_1$ 或 $d_2.C_2$ tick,时钟 $d.C$ 同时 tick,时钟 $d.C$ 与时钟 $d_1.C_1$ 和 $d_2.C_2$ 的关系满足:

- $d_1.C_1$ *subClock* $d.C$;
- $d_2.C_2$ *subClock* $d.C$;
- $\forall i \in I_{d.C}, \exists j \in I_{d_1.C_1} \cup I_{d_2.C_2}, i \equiv j$.

直观上理解,时钟 $d_1.C_1(d_2.C_2)$的任一时间点与 $d.C$ 的某一时间点同时发生,如图 10 所示,其 LTS 语义如图 11 所示.它表示在任一时刻,要么 $d.C$ 与 $d_1.C_1$ 同时 tick,要么 $d.C$ 与 $d_2.C_2$ 同时 tick,要么 3 个时钟都 tick,要么 3 个时钟都不 tick.

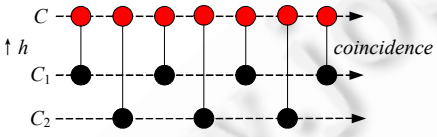

Fig.10 Instant graph for *union*

图 10 *union* 的时间点图

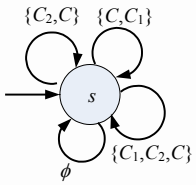

Fig.11 LTS semantics of $d.C \triangleq d_1.C_1$ *union* $d_2.C_2$

图 11 $d.C \triangleq d_1.C_1$ *union* $d_2.C_2$ 的 LTS 语义

当问题领域为相同结构时,我们考虑两种时钟:一种是每个时间点都取二者最慢,一种是每个时间点都取最快,最慢的我们定义为 *sup*,最快的定义为 *inf*.

定义 8. *sup*:

$$d.C \triangleq d_1.C_1 \ sup \ d_2.C_2.$$

要求 d_1,d_2 是相同结构的问题领域,问题领域 d 为它们的并.时钟 $d.C$ 的第 i 个时间点与 $d_1.C_1$ 和 $d_2.C_2$ 的第 i 个时间点中慢的一个同时发生,$\forall k \in \mathbb{N}^*, C[k] \in I_{d.C}$,如下 3 个关系必须满足:

1) $(d_1.C_1[k] \in I_{d_1.C_1}) \wedge$;
2) $(d_2.C_2[k] \in I_{d_2.C_2}) \wedge$;
3) $d.C[k] \equiv d_1.C_1[k]$ if $d_2.C_2[k] \preceq d_1.C_1[k]$;

   $d.C[k] \equiv d_2.C_2[k]$ if $d_1.C_1[k] \preceq d_2.C_2[k]$.

图 12 给出了其直观含义,C 中的每一个时间点都取 C_1 和 C_2 中发生最慢的时间点.我们需要记录时钟 $d_1.C_1$ 和 $d_2.C_2$ 已发生过的时间点的个数的差值δ,以决定某一时刻 $d.C$ 的时间点是否发生以及与谁同时发生.其 LTS 语义如图 13 所示,当时钟 $d_1.C_1$ 比 $d_2.C_2$ 快时,$d.C$ 和 $d_2.C_2$ 相应的时间点同时发生;反之,则与 $d_1.C_1$ 对应的时间点同时发生.

定义 9. *inf*:

$$d.C \triangleq d_1.C_1 \ inf \ d_2.C_2.$$

与 *sup* 相反,$d.C$ 的每个时间点与 $d_1.C_1$ 和 $d_2.C_2$ 对应时间点中较快的那个同时发生,$\forall k\in\mathbb{N}^*, C[k]\in I_{d.C}$ 必须满足如下条件:

1) $(d_1.C_1[k]\in I_{d_1.C_1})\wedge$;

2) $(d_2.C_2[k]\in I_{d_2.C_2})\wedge$;

3) $d.C[k]\equiv d_2.C_2[k]$ if $d_2.C_2[k]\preceq d_1.C_1[k]$;

$d.C[k]\equiv d_1.C_1[k]$ if $d_1.C_1[k]\preceq d_2.C_2[k]$.

与其他算子类似,*inf* 的直观含义如图 14 所示,其 LTS 语义如图 15 所示.

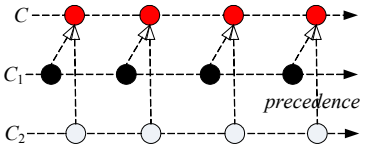

Fig.12 Instant graph for *sup*

图 12 *sup* 的时间点图

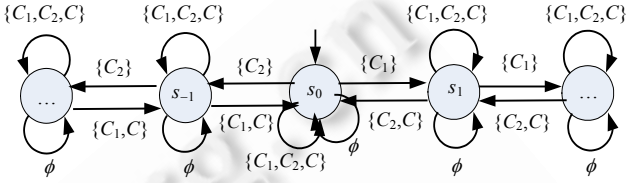

Fig.13 LTS semantics of $d.C\triangleq d_1.C_1$ *sup* $d_2.C_2$

图 13 $d.C\triangleq d_1.C_1$ *sup* $d_2.C_2$ 的 LTS 语义

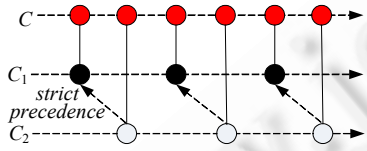

Fig.14 Instant graph for *inf*

图 14 *inf* 的时间点图

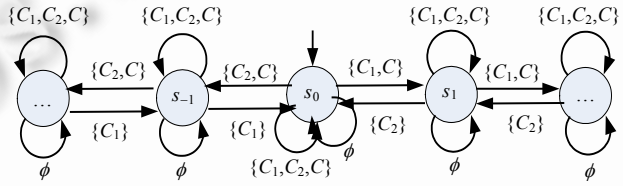

Fig.15 LTS semantics of $d.C\triangleq d_1.C_1$ *inf* $d_2.C_2$

图 15 $d.C\triangleq d_1.C_1$ *inf* $d_2.C_2$ 的 LTS 语义

5 时间需求规约验证

通过前几节的建模,得到的时间需求规约包括各种时间约束(如时间点关系约束、时钟关系约束),这些约束之间可能产生冲突.例如,某个时间点约束要求 *a strictPre b*,而其他约束条件却推导出 *b strictPre a*,由此产生不一致,或者时钟约束发生冲突,整个系统就会互相等待,产生死锁.靠需求分析人员手工分析各个时钟关系相互作用下所有可能的情况容易出错也不现实,因此需要形式化的方法和工具支持.由于我们已经定义了时钟、时钟关系和时钟构造算子的形式化语义,只需要找到合适的形式化工具来支持该分析.本文我们选择使用模型检测工具 NuSMV 来验证时间需求规约,将时间需求规约转换为 NuSMV 模型,用 CTL 公式表达要验证的性质,然后进行规约验证.

5.1 时间需求规约到NuSMV模型的转换

一个时间需求规约转换成一个 NuSMV 的 FSM 模型,对应一个 main MODUEL(main 模块),时间约束(instant relation,clcok relation,clock constructor)转换为 NuSMV 中的 MODULE,规约中出现的时钟在 main MODULE 的 VAR 部分声明为布尔变量,出现的时钟约束创建为对应约束 MODULE 的实例,时钟变量的赋值对应于规约约束中,该时钟是否 tick(TRUE 对应 tick,FALSE 对应不 tick).每个时钟约束的 MODULE 编码是其 LTS 语义的直接映射:状态由 MODULE VAR 中生命的状态变量模拟,迁移由 MODUEL 中 TRANS constraints 实现,MODULE 实例共同作用,形成整个规约的 FSM 模型,决定规约当前的状态以及每个迁移中每个时钟的赋值.每个时钟的赋值即为当前时刻时钟的 tick 情况,它影响算子接下去应该处于其 LTS 语义中的哪一个状态.据此,得到从时间需求规约到 NuSMV 的转换规则,见表 1.

Table 1 Transformation rules from timing requirements specification to NuSMV models

表 1 时间需求规约到 NuSMV 模型的转换规则

Timing specification	NuSMV
Specification	main MODULE
Clock C	*C*: *boolean*
Instant relation	constraint MODULE, select MODULE
Clock relation, *Clock constructor*	constraint MODULE
State *s*	VAR *s*: *boolean*; ...
Transition $s \xrightarrow{\{c_1,c_2,\ldots,c_n\}} s'$	TRANS case *s*=TRUE: *next*(c_1)=TRUE & *next*(c_2)=TRUE & ... & *next*(c_n)=TRUE & *next*(*s*)=FALSE & *next*(*s*′)=TRUE esac;

需要注意的是,从 Instant relation 到 constraint MODULE 的转换,常使用的时间约束大多建立在时钟层面上,而且不需要记数每个时钟的每个时间点(时钟的时间点通常无限大,记数会引入非常大的状态空间).例如,*subClock* 不需要记数时间点而只有 1 个状态,*fasterThan* 只需要记录时钟漂移δ而不是每个时间点的编号,所以我们的 MODULE 都实现了在时钟层面上减少状态空间.但是在这种实现框架下,时间点约束的编码需要特别处理,选出其作用的时钟的某个特定编号的时间点,所以我们引入 *select* 操作,保证时间点约束的正确实现并且尽量缩小状态空间,使 NuSMV 实现仍然在时钟层面上.*select* 操作有两个参数:时钟 *C* 和整数 *i*,*select*(*C*,*k*)选出时钟 *C* 的第 *i* 个时间点,*select* 用计数器 *ct* 来记数时钟的时间点编号,直到 *k* 到达,余下的时间点将不再被记数.在时钟层面的 NuSMV 实现上可以理解为,*select* 创建了一个新的虚拟时钟(这里,虚拟的意思是这个新创建的时钟没有出现在原时间规约中),该时钟只 tick 1 下,并且只能与时钟 *C* 的第 *i* 个时间点同时发生.

另外,由于 NuSMV 模型为有限状态模型,上述规则的使用必须满足一个条件:要转换的时间需求规约具有有限状态空间.有些时钟约束可能引起无界的状态空间,例如,*fasterThan*,*sup*,*inf* 需要记录两时钟间的漂移δ.当一个时间规约含有这样的约束时,它的状态空间也可能是有限的(各个约束相互作用,使得每个漂移δ只可能落在一个有界区间内),而且大多数情况下如此.判断时间约束是否满足上面的情况(互相约束导致有界),需要额外的严格检查.

在本文中,为了避免这样的检查,我们要求如果 *fasterThan*,*sup*,*inf* 出现在一个时间规约中,对应时钟间的漂移要使用 *boundedDiff* 约束显式说明δ的有界区间;而且在实际建模中,通常需求往往要求有界的差值而不是无穷大小的差值.

下面我们用一个小例子来说明上述转换规则.

表 2 左边给出了包含两个 *subClock* 时钟关系和一个时间点关系的时间约束,右边是其转换过去的 NuSMV 模型.该时间约束中的 3 个时钟 $d_1.C_1$,$d_2.C_2$,$d_3.C_3$ 转换为 NuSMV 模型中的 3 个布尔变量 c_1,c_2,c_3,时钟关系 *subClock* 的 LTS 语义编码为 MODULE *subClock*.如图 5 所示,*subClock* 的 LTS 只有 1 个状态,所以在 MODUEL 实现中不需要另外声明状态变量来模拟状态的转变,其 LTS 的 3 个迁移由 TRANS 中的命题公式描述.每次迁移要么两个时钟都 tick,要么父时钟单独 tick,要么两个时钟都不 tick.规约中包含的两个 *subClock* 时钟关系在 main MODULE 中创建为 MODULE *subClock* 的两个实例 ctr_1,ctr_2,时间点关系的 LTS 语义由 instantstrictPre MODUEL 实现(直接映射),它用到的两个时间点 $d_1.C_1[5]$和 $d_2.C_2[3]$由 *select*(c_1,5),*select*(c_2,3)选出,对应为虚拟时钟 c_{15},c_{23} 也声明在 main MODULE 的 VAR 里,时间点关系 $d_1.C_1[5]-d_2.C_2[3]<_{d_3.C_3}$ 4 声明为 instant strictPre MODUEL 的实例,与其他两个 *subClock* 的实例共同作用,决定运行中每个时钟的赋值.

Table 2 An example of transformation

表 2 转换的例子

时间规约	$d_1.C_1$ *subClock* $d_2.C_2$ $d_1.C_1$ *subClock* $d_3.C_3$ $d_1.C_1[5]$–$d_2.C_2[3]<d_3.C_3$ 4

对应的NuSMV模型	MODULE main
	VAR
	c_1: *boolean*; c_2: *boolean*; c_3: *boolean*; c_{15}: *boolean*; c_{23}: *boolean*; ctr_1: *subClock*(c_1,c_2);
	ctr_2: *subClock*(c_2,c_3); $select_1$: *select*(c_1,5); $select_2$: *select*(c_2,3); ctr_3: *instantstrictPre*(c_{15},c_{23},c_3,4);
	INIT c_1=FALSE & c_2=FALSE & c_3=FALSE & c_{15}=FALSE & c_{23}=FALSE;
	MODULE *subClock*(*left*,*right*)
	TRANS
	(*next*(*left*)=TRUE & *next*(*right*)=TRUE)|(*next*(*left*)=FALSE &
	next(*right*)=TRUE)|(*next*(*left*)=FALSE & *next*(*right*)=FALSE);
	MODULE *select*(*instant*,*clock*,*k*)
	VAR *ct*: 0,...,*k*;
	INIT *ct*=0;
	TRANS
	case
	ct+1=*k*: (*next*(*instant*)=TRUE & *next*(*clock*)=TRUE & *next*(*ct*)=*ct*+1)
	|(*next*(*instant*)=FALSE & *next*(*clock*)=FALSE & *next*(*ct*)=*ct*);
	ct+1<*k*: *next*(*instant*)=FALSE & ((*next*(*clock*)=TRUE & *next*(*ct*)=*ct*+1)|(*next*(*clock*)=FALSE & *next*(*ct*)=*ct*));
	TRUE:*next*(*instant*)=FALSE & *next*(*cnt*)=*ct*;
	esac
	MODULE *instantstrictPre*(*left*,*right*,*gc*,*k*)
	VAR *ct*: –2,...,*k*;
	INIT *ct*=–2;
	TRANS
	case
	ct=–2: (*next*(*left*)=FALSE & *next*(*right*)=TRUE & *next*(*gc*)=TRUE & *next*(*ct*)=*ct*+1)
	|(*next*(*left*)=FALSE & *next*(*right*)=FALSE & *next*(*gc*)=FALSE & *next*(*ct*)=*ct*);
	cnt>–2 & *ct*+1<*k*: (*next*(*left*)=FALSE & *next*(*right*)=FALSE & *next*(*gc*)=TRUE & *next*(*ct*)=*ct*+1)
	|(*next*(*left*)=FALSE & *next*(*right*)=FALSE & *next*(*gc*)=FALSE & *next*(*ct*)= *ct*);
	ct+1=*k*: (*next*(*left*)=TRUE & *next*(*right*)=FALSE & *next*(*gc*)=TRUE & *next*(*ct*)=*ct*+1)
	|(*next*(*left*)=FALSE & *next*(*right*)=FALSE & *next*(*gc*)=FALSE & *next*(*ct*)=*ct*);
	TRUE: *next*(*ct*)=*ct*;
	esac;

5.2 时间需求规约的性质描述

CPS 是感知与控制反馈循环的系统,它在环境感知的基础上,实现对物理设备的控制,如此循环往复.这样的系统应该能够保持运行,其系统时间应该持续下去.那么在 CPS 系统的时间需求规约中,所有时钟都应该可以再次 tick.如果规约中约束具有不一致的地方,那么在规约运行中,为了保证每个时间约束都被满足,整个系统(所有时钟)或者其中的一部分(部分时钟)将不能再 tick.我们称这种情况为规约存在不一致,有死锁.

由于 NuSMV 实现中时钟 tick(不 tick)表现为该时钟对应的变量赋值 TRUE(FALSE),若用 CTL 公式来描述一致性属性,C_1 就表示当前状态下 C_1 的值为 1,即上一个时刻 C_1 刚刚 tick 过;相反,$!C_1$ 表示上一个时刻 C_1 没有 tick.规约死锁,即在某时刻起,没有时钟可以再 tick,则应表述为 CTL 公式 $EF(AGp)$,其中,$p=!(C_1|C_2|\ldots|C_n)$表示在某状态没有时钟 tick,那么 AGp 表示该状态出发的所有路径中没有任何一个时钟 tick.由此,我们定义全局死锁.

定义 10. 对于时钟集合为 $T=\{C_1,C_2,\ldots,C_n\}$时间需求规约,我们称该规约存在全局死锁,如果其 NuSMV 模型满足 CTL 公式 $EF(AGp)$.

由于我们假定所有时钟都应该无穷多次 tick,所以如果规约运行到某点后,这个时钟永远不能再 tick 了,我们称这个时钟被死锁了.在系统需求分析中,某个(些)时钟对应的问题领域可能非常重要,设计者需要特别保证其不会被死锁,CTL 公式提供给用户进行该检查的方法.

定义 11. 对于时钟集合为 $T=\{C_1,C_2,\ldots,C_n\}$时间需求规约,我们称该规约存在某一时钟 C_i 的死锁,如果其满足 CTL 公式 $EF(AGq)$,$q=!C_i$.

6 建模验证过程及案例研究

图 16 给出了一个统一的需求建模及验证的过程,过程的输入是一个问题图,它包括问题领域、交互等等.问题图的获取过程可以参考文献[16],过程的输出是一个经过验证的时间需求规约.在图 16 中有 4 大步,我们将使用一个案例来阐述这些步骤.其问题陈述如下:

汽车防抱死制动系统(ABS)是一个小型的 CPS 系统,它的架构包含 4 个传感器(*Sensor ifl*,*ifr*,*irl*,*irr*)和 4 个执行器(*Actuator ofl*,*ofr*,*orr*,*orl*).传感器感应轮子的转动速度,执行器表征施加于轮子上的刹车压力.

ABS 的执行是由 *R* 触发,且必须每 5ms(最大 1ms 的抖动)执行一次.ABS 的 4 个感应器的值在一定延迟内到达(也叫输入同步),输入同步延迟为 0.5ms,一个类似的输出同步延迟出现在执行器.另外,它还允许从输入集合的第 1 个事件到最后一个输出的事件之间存在的延迟必须少于 3ms.

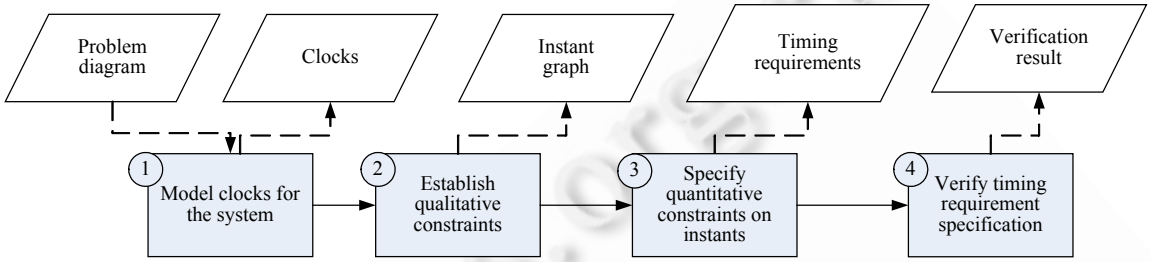

Fig.16 The process of modeling and verifying timing requirements

图 16 时间需求建模验证过程

作为过程的输入,问题图应该给出需求涉及的问题领域、交互以及领域类型(基本领域还是组合领域).

图 17 给出了 ABS 系统的问题图,在这个图中,组合领域 Sensor 有 4 个基本领域:*ifl*,*ifr*,*irl* 和 *irr*.组合领域 Actuator 有 4 个基本领域:*ofl*,*ofr*,*orl* 和 *orr*.显然,*ifl*,*ifr*,*irl*,*irr* 和 Sensor 是相同结构的领域,*ofl*,*ofr*,*orl*,*orr* 和 Actuator 是相同结构的领域.

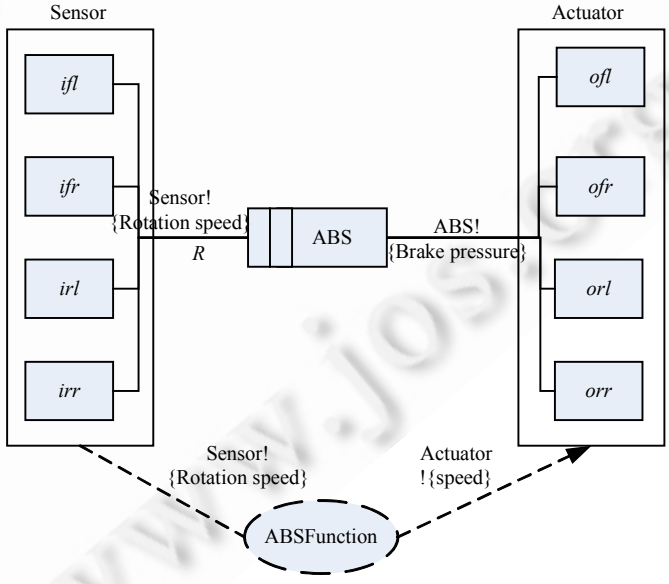

Fig.17 Problem diagram of ABS

图 17 ABS 的问题图

ABS 和 Sensor 之间、ABS 和 Actuator 之间的交互见表 3.为了简化,本文就使用 int_{ij}(i 可以是 1,2,3,or 4,j 是领域的名字)来表示子领域与 ABS 的交互.

Table 3 Declaration

表 3 声明

交互	int_1=⟨*ABS*,*Sensor*,*R*⟩ int_2=⟨*Sensor*,*ABS*,*rotation speed*⟩ int_3=⟨*ABS*,*Actuator*,*break pressure*⟩ int_4=⟨*Actuator*,*ABS*,*speed*⟩	时钟	*Clock Sensor*.C_{sensor}, *Actuator*.$C_{Actuator}$; *Clock ifl*.C_{ifl}, *ifr*.C_{ifr}, *irl*.C_{irl}, *irr*.C_{irr}; Clock *ofl*.C_{ofl}, *ofr*.C_{ofr}, *orl*.C_{orl}, *orr*.C_{orr};
时间点集	$I_{C_{Actuator}}=\{O(int_3),O(int_4)\}$; $I_{C_{ifl}}=\{O(int_{1ifl}),O(int_{2ifl})\}$; $I_{C_{ifr}}=\{O(int_{1ifr}),O(int_{2ifr})\}$; $I_{C_{irl}}=\{O(int_{1irl}),O(int_{2irl})\}$; $I_{C_{irr}}=\{O(int_{1irr}),O(int_{2irr})\}$; $I_{C_{ofl}}=\{O(int_{3ofl}),O(int_{4ofl})\}$; $I_{C_{ofr}}=\{O(int_{3ofr}),O(int_{4ofr})\}$; $I_{C_{orl}}=\{O(int_{3orl}),O(int_{4orl})\}$; $I_{C_{orr}}=\{O(int_{3orr}),O(int_{4orr})\}$;	严格优先关系	$O(int_{1ifl})\prec O(int_{2ifl})$; $O(int_{1ifr})\prec O(int_{2ifr})$; $O(int_{1irl})\prec O(int_{2irl})$; $O(int_{1irr})\prec O(int_{2irr})$; $O(int_{3ofl})\prec O(int_{4ofl})$; $O(int_{3ofr})\prec O(int_{4ofr})$; $O(int_{3orl})\prec O(int_{4orl})$; $O(int_{3orr})\prec O(int_{4orr})$;

第 1 步:为每个问题领域建立时钟.

这一步可以分 3 个子步骤来完成:

1) 为问题图中的每个问题领域声明一个时钟.在问题图中,对每个问题领域建立时钟标签.

回到我们的例子,将如图 17 所示的每个问题领域加以声明,由此得到时钟声明,见表 3,标上时钟,得到了带有时钟的问题图,如图 18 所示.

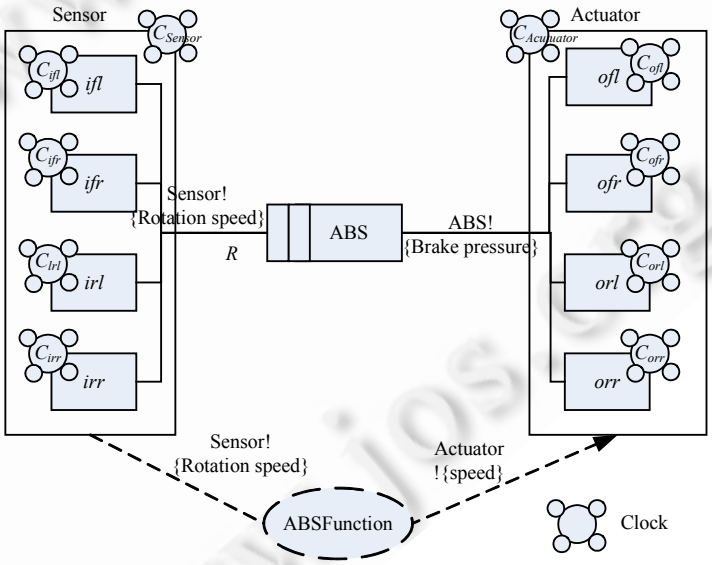

Fig.18 Problem diagram of ABS with clocks

图 18 ABS 带时钟的问题图

2) 为基本领域的时钟建模.

根据时钟的定义,这一步可以分两个子步骤来完成:

(1) 为每个时钟找到时间点.

每个时钟的时间点都是这个领域发起或者接收的交互发生的时刻,所以这一步就是去识别每个交互的发生.对问题图中的该时钟对应的问题领域的每个交互,确定其发生的时刻,标记为该时钟的时间点.

(2) 在每个时钟内确定时间点之间的严格优先关系.

确立时间点之间的严格优先关系,即,明确对应交互发生的先后顺序,可以请需求提供者进行确立.

本文例子中的基本领域包括传感器 *ifl*,*ifr*,*irl*,*irr* 与执行器 *ofl*,*ofr*,*orl*,*orr*.int_{1ifl} 发生的时刻(记为 $O(int_{1ifl})$)和 int_{2ifl} 发生的时刻(记为 $O(int_{2ifl})$)都是 C_{ifl} 的时间点,因此可以有 $I_{C_{ifl}}=\{O(int_{1ifl}),O(int_{2ifl})\}$.类似地,我们得到其他时钟的时间点集,见表 3.

至于时间点关系,例如在时钟 C_{ifl} 中,int_{1ifl} 必须发生在 int_{2ifl} 之前,所以 $O(int_{1ifl})\prec O(int_{2ifl})$.类似地,可以得到其他严格先于关系,见表 3.

3) 为组合领域建模时钟.

这一步是从已有的时钟构造新的时钟,通过使用时钟构造算子来得到.对于每个组合领域,根据它与子领域的关系以及想要表达的意思,选择合适的算子来构造组合领域的时钟.若组合领域是由相同结构的领域组合而成,则可以使用 *sup* 或 *inf* 算子.若想表达最快的时钟,则用 *inf* 算子;若想表达最慢的时钟,则用 *sup* 算子.也可以选择用户自行定义的算子.若组合领域是由不同结构的领域组合而成,可以使用 *union* 算子.

在本例子中,Sensor 是 *ifl*,*ifr*,*irl* 和 *irr* 的组合,而 *ifl*,*ifr*,*irl* 和 *irr* 都有相同结构的领域,而且 Sensor 应该表现最慢,由此得到:

$$Sensor.C_{sensor}=sup(ifl.C_{ifl},ifr.C_{ifr},irl.C_{irl},irr.C_{irr}).$$

类似地,可以得到:

$$Actuator.C_{Actuator}=sup(ofl.C_{ofl},ofr.C_{ofr},orl.C_{orl},orr.C_{orr}).$$

第 2 步:确立定性关系.

第 1 步基本确立了同一个时钟之内的时间点关系,这一步主要是要确立时钟之间的时间点之间的关系.通常,我们确立 3 种关系:precedence,coincidence 和 strict precedence.这些时间点关系的识别还是要从对应的交互出发,请需求提供者确立发生的先后顺序.这样结合第 1 步,我们就可以得到所有时间点之间的定性关系,可以用时间点图表示.

在我们的例子中,在 $Sensor.C_{sensor}$ 和 $Actuator.C_{Actuator}$ 之间,int_2 必须在 int_3 之前发生,因此 $O(int_2)\prec O(int_3)$.在 $Sensor.C_{sensor}$ 和 C_{ifl} 之间,int_1 的发生不会早于 int_{1ifl},所以 $O(int_{1ifl})\preceq\prec O(int_1)$.

类似地,我们可以得到 ABS 内其他的时间点关系,如图 19 所示.

第 3 步:确立时间点之间的定量关系.

时间点之间的定量关系需要由需求提供者给出,定量关系首先考虑涉及到的领域、交互、时钟能否从时钟层面上给出,最后考虑时间点.一般来讲,定量相关的需求主要有 3 类:重复率、延迟需求以及输入输出同步.重复率型的需求一般形式是每几秒执行一次,这种类型首先考虑执行的问题领域,其对应的时钟,然后考虑 *filteredby* 时钟构造算子构造新的时钟.延迟需求和输入输出同步都是要找出执行的问题领域和交互,将领域对应的时钟分解到这个交互对应的时钟(即,时钟的时间点就是这个交互的每次发生,可以虚构).鉴于时钟的虚构性,可以仅用时钟的第 1 个时间点之间的定量关系来表示时间段.结合这一步得到的定量关系以及上一步得到的定性关系,我们可以得到时间需求规约.

比如,本例中重复率(repetition rate)类型的需求通常写成:ABS 的功能必须每 5ms(最大 1ms 的抖动)执行一次.这个功能通过 Sensor 与 Actuator 的组合来周期性地实现,Sensor 与 Actuator 是不同的领域,因此,现有的时钟将是:

$$Sensor.C_{sensor}\ union\ Actuator.C_{Actuator}.$$

根据第 4 节的单领域 *filteredby* 时钟构造算子,其他参数包括初始值为 0,周期为 5.因此,新的时钟 C_{new} 可以通过如下算子构造出来:

$$ABS.C_{new}\triangleq(Sensor.C_{sensor}\ union\ Actuator.C_{Actuator})\blacktriangle(1.0^{5-1})^{\omega}.$$

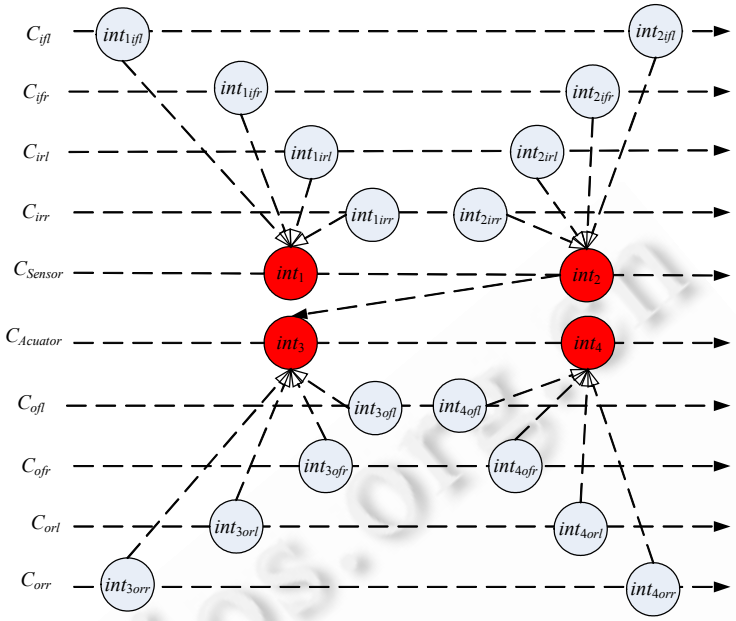

Fig.19 Instant graph for ABS

图 19 ABS 的时间点图

延迟需求类型的需求可以被描述为:从输入集合的第 1 个事件到最后一个输出的事件之间存在的延迟必须少于 3ms.以 Sensor 的 int_1 为例作为输入,以 *Actuator* 的 int_4 为例作为输出,那么最初的输入为 int_{1ifr},int_{1ifl},int_{1irl},int_{1irr} 中反应最快.由于其特殊性,我们做虚拟时钟 $Sensor.C_{input}$ 表示输入中最早的时钟,而为了统一,必须将原先的 $ifr.C_{ifr}$ 拆分为两个:$ifr.C_{ifr}=ifr.C_{ifr1}$ *alternate* $ifr.C_{ifr2}$,其中,$ifr.C_{ifr1}$ 对应交互 int_{1ifr}.其他时钟也类似,在这种情况下有:$Sensor.C_{input}\triangleq inf(ifr.C_{ifr1},ifl.C_{ifl1},irl.C_{irl1},irr.C_{irr1})$.

最慢的输出为 $O(int_4)$,为其构造虚拟时钟 $Actuator.C_{Actuator2}$,则延迟需求表示为

$$Actuator.C_{Actuator2}[1]-Sensor.C_{input}[1]<30.$$

输入输出同步型类似于需要一个 0.5ms 的输入同步.第 1 个输入是 O_{1inf},其作为虚拟时钟 $Sensor.C_{input}$,最后的输入是 $O(int_1)$,对应虚拟时钟 $Sensor.C_{Sensor}$.于是,一个 0.5ms 的输入同步可以表示为

$$Sensor.C_{sensor}[1]-Sensor.C_{input}[1]<5.$$

第 4 步:转换为 Nusmv 描述进行验证.

根据第 5 节的转换规则,将案例时间规约转换为 NuSMV 模型,系统属性描述为 CTL 公式,我们可以检查案例时间规约的一致性,该时间规约存在全局死锁,如图 20 所示.

通过分析 NuSMV 提供的反例路径我们发现,该死锁是由约束 $C_{Actuator2}$ *strictPre* $C_{Sensor1}$ 的不当引入造成的.在 *input* 时钟与 *global* 钟同时 tick 1 下后,*global* 时钟又单独 tick 了 5 下,由约束 $C_{Actuator2}[1]-C_{input}[1]<_{global}5$ 限定,下面需要时钟 $C_{Actuator2}$ tick(与 *global* 一起),又由 $C_{Actuator}\triangleq C_{Actuator1}$ *union* $C_{Actuator2}$ 得到 $C_{Actuator2}$ 需要与 $C_{Actuator}$ 同时 tick.时钟 $C_{Actuator2}$ 和 $C_{Actuator}$ 之前都没有 tick 过,所以 $C_{Actuator}$ 的 tick 可以发生的条件为:之前有 C_{Sensor} 的第一个时间点发生过(约束 C_{Sensor} *strictPre* $C_{Actuator}$).约束 $C_{Sensor}\triangleq C_{Sensor1}$ *union* $C_{Sensor2}$ 和 $C_{Sensor1}$ *strictPre* $C_{Sensor2}$ 限定 C_{Sensor} 的第 1 个时间点的发生一定要与 $C_{Sensor1}$ 的第 1 个时间点的发生同时,所以 $C_{Actuator2}$ 的第 1 个时间点的发生的条件为:之前 $C_{Sensor1}$ 的第 1 个时间点发生过.这与 $C_{Actuator2}$ *strictPre* $C_{Sensor1}$ 矛盾,所以规约这条路径运行到需要 $C_{Actuator2}$ tick 时发生死锁.

分析案例的需求我们发现,$C_{Actuator2}$ *strictPre* $C_{Sensor1}$ 约束应该改为 $C_{Sensor2}$ *strictPre* $C_{Actuator1}$.我们修改了案例的时间规约,再次检查,该规约不再存在全局死锁.

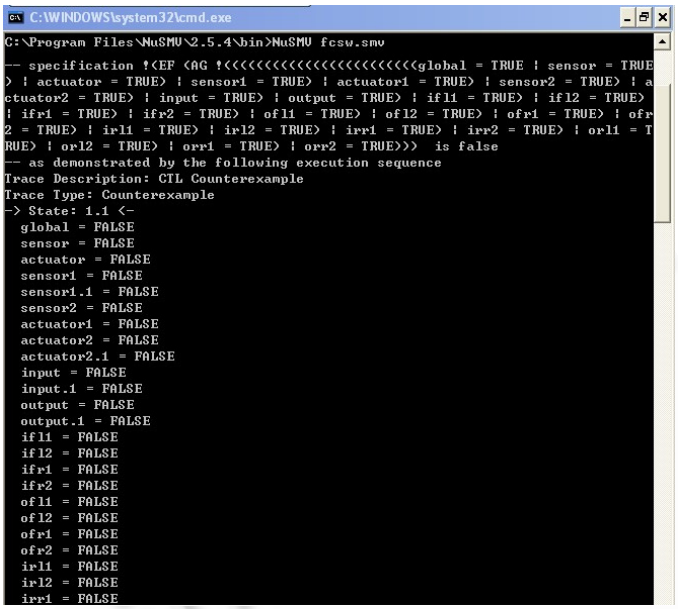

Fig.20 Checking result: Global deadlock

图 20 检测结果:全局死锁

7 相关工作及结束语

基于功能性需求的时间需求建模的工作很多,例如面向目标的方法[4]和面向主体与意图的方法[5,6]:

- 面向目标的方法,比如 KAOS(knowledge acquisition in automated specification)[17],将目标看作是需求的来源,它使用带类型的一阶时序逻辑作为逻辑语言,既包含传统的时序算子,也包含描述涉及实时节点特性的附加实时算子,描述实时特性.它建模实时特性精确,没有显示地参考时间变量.
- 面向主体和意图的方法提供一种基于组织层次上下文的需求获取和建模的思路,其建模理念为刻画有目的的参与者.其代表性工作是 *i**框架[5]和 Formal Tropos[6],它们也是采用带类型的一阶时序逻辑语言来描述时间需求.ALBERT-II(agent-oriented language for building and eliciting real-time requirements)[18]也是一种面向主体和意图的方法,它的描述基础也是一阶时序逻辑,其主体(agent)的状态和行为都通过基于逻辑的符号表示为约束.

上述工作都基于时序逻辑,时序逻辑在描述定性约束的时候没有问题.为了描述定量约束,进行了多种扩充,如 Timed Computation Tree Logic[8]等,这些时序逻辑通常用于表示实时系统的性质和规范,在需求工程中用到的较多的是一阶时序逻辑和谓词时序逻辑[19].由于目标、主体与意图都不与问题领域、交互直接相关,因此上述工作都不适合建模 CPS 系统.

问题框架方法采用问题领域和交互进行建模,更适合建模 CPS 系统.在问题框架相关的时间需求比较有限,Choppy 等人提出时间事件来表示时间[20],Barroca 等人使用 Timer 作为问题领域的一部分[21].他们都没有将交互用时间显式表达出来.我们前期也做过一些基于问题框架的时间需求工程[9,10],但是该方法侧重在多形态时间,致力于解决多样的环境时间描述与单一的软件时间描述的统一.

基于前期工作,本文提出了一种基于逻辑时钟的 CPS 时间需求一致性分析方法,增加了时间需求模型的形式化语法与语义,进行了不一致性属性的验证,是更深入的工作.主要贡献包括:

- 将逻辑时钟引入到问题框架中,构建了 CPS 时间需求模型.通过抽取问题框架方法与时间需求相关工作的基本概念,得到了基于问题框架的时间需求基本概念框架,为形成 CPS 时间需求建模机制奠定了良好的基础.
- 给出了 CPS 时间需求模型概念的语法和操作语义.通过操作语义,给出了 CPS 建模机制中的概念的内涵,方便了后续的形式化验证;
- 给出了 CPS 一致性时间需求验证机制.在形式化语义基础上,将时间需求规约转换成 NuSMV 模型,并定义了不一致性属性,使用 NuSMV 进行了一致性检测.通过检测,为后续的设计与实现奠定了坚实的基础.

目前,仍需要更深入的工作解决一些问题,例如,检验模型的规模偏大,如何减少检测过程中的状态空间,以加速检验的过程,如何在时钟层面上增加定量关系、自动的工具支持等等.

References:

[1] Lee EA. Cyber physical systems: Design challenges. In: Arlat J, ed. Proc. of the 11th IEEE Symp. on Object/Component/Service-Oriented Real-Time Distributed Computing. Washington: IEEE Computer Society, 2008. 363−369. [doi: 10.1109/ISORC.2008.25]

[2] Lee EA. CPS foundations. In: Proc. of the Design Automation Conf. (DAC). New York: ACM Press, 2010.

[3] Jackson M. Problem Frames: Analyzing and Structuring Software Development Problems. New York: Addison-Wesley, 2001.

[4] Van Lamsweerde A. Goal-Oriented requirements engineering: A guided tour. In: Titsworth FM, ed. Proc. of the 5th IEEE Int'l Symp. on Requirements Engineering. Washington: IEEE Computer Society, 2001. 249−263. [doi: 10.1109/ISRE.2001.948567]

[5] Yu E. Agent orientation as a modeling paradigm. Wirtschaftsinformatik, 2001,43(2):123−132. [doi: 10.1007/BF03250789]

[6] Yu E. Towards modeling and reasoning support for early phase requirements engineering. In: Agresti W, ed. Proc. of the 3rd IEEE Int'l Symp. on Requirements Engineering. Washington: IEEE Computer Society, 1997. 226−235. [doi: 10.1109/ISRE.1997.566873]

[7] Hall JG, Rapanotti L, Jackson M. Problem oriented software engineering: Solving the package router control problem. IEEE Trans. on Software Engineering, 2008,34(2):226−241. [doi: 10.1109/TSE.2007.70769]

[8] Alur R, Courcoubetis C, Dill D. Model-Checking in dense real-time. Information and Computation, 1993,104(1):2−34. [doi: 10.1006/inco.1993.1024]

[9] Chen XH, Liu J. Modeling software timing requirements: An environment based approach. Chinese Journal of Computers, 2013, 36(1):88−103 (in Chinese with English abstract).

[10] Mallet F, Peraldi-Frati MA, André C. Marte CCSL to execute East-ADL timing requirements. In: Werner B, ed. Proc. of the 12th Symp. on Object/Component/Service-Oriented Real-Time Distributed Computing. Los Alamitos: IEEE Computer Press, 2009. 249−253. [doi: 10.1109/ISORC.2009.18]

[11] André C. Syntax and semantics of the clock constraint specification language (CCSL). Research Report, 6925, INRIA, 2009. http://hal.inria.fr/inria-00384077

[12] Chen XH, Liu J, Mallet F, Jin Z. Modeling timing requirements in problem frames using CCSL. In: Thu TD, ed. Proc. of the 18th Asia-Pacific Software Engineering Conf. Washington: IEEE Computer Society, 2011. 381−388. [doi: 10.1109/APSEC.2011.30]

[13] Cimatti A, Clarke E, Giunchiglia E, Roveri M. NUSMV: A new symbolic model checker. Int'l Journal on Software Tools for Technology Transfer, 2000,2(4):410−425. [doi: 10.1007/s100090050046]

[14] Jin Z, Chen XH, Zowghi D. Performing projection in problem frames using scenarios. In: Guerrero JE, ed.. Proc. of the 16th Asia-Pacific Software Engineering Conf. Washington: IEEE Computer Society, 2009. 249−256. [doi: 10.1109/APSEC.2009.22]

[15] Uchitel S, Kramer J, Magee J. Synthesis of behavioural models from scenarios. IEEE Trans. on Software Engineering, 2003,29(2): 99−115. [doi: 10.1109/TSE.2003.1178048]

[16] Chen XH, Jin Z. An ontology-guided process for developing problem frame specification: An example. In: Rapanotti L, ed. Proc. of the 3rd Int'l Workshop on Applications and Advances of Problem Frames. New York: ACM Press, 2008. 36−39. [doi: 10.1145/1370811.1370818]

[17] Darimont R, van Lamsweerde A. Formal refinement patterns for goal driven requirements elaboration. In: Garlan D, ed. Proc. of the 4th ACM Symp. on the Foundations of Software Engineering. NewYork: ACM Press, 1996. 179−190. [doi: 10.1145/239098.239131]

[18] Bois P. The albert ii language: On the design and the use of a formal specification language for requirements analysis. [Ph.D. Thesis]. Namur: Dept. of Computer Science, University of Namur, 1995.

[19] Schreiber FA. Is time a real time? An overview of time ontology in informatics. Real Time ComputingSpringer Verlag NATO ASI, 1994,F(127):283−307. [doi: 10.1007/978-3-642-88049-0_14]

[20] Choppy C, Reggio G. A UML-based approach for problem frame oriented software development. Information and Software Technology, 2005,47(14):929−954. [doi: 10.1016/j.infsof.2005.08.006]

[21] Barroca L, Fiadeiro J, Jackson M, Laney R, Nuseibeh B. Problem frames: A case for coordination. In: De Nicola R, ed. Proc. of the 6th Int'l Conf. on Coordination Models and Languages. Berlin, Heidelberg: Springer-Verlag, 2004. 5−19. [doi: 10.1007/978-3-540-24634-3_4]

附中文参考文献:

[9] 陈小红,刘静.基于环境的多形态时间需求建模方法.计算机学报,2013,36(1):88−103.

尹玲(1986－),女,辽宁朝阳人,博士生,主要研究领域为软件工程,模型检测.
E-mail: yinling86@gmail.com

刘静(1964－),女,博士,教授,博士生导师,主要研究领域为高可信计算,形式化方法.
E-mail: jliu@sei.ecnu.edu.cn

陈小红(1982－),女,博士,讲师,CCF 会员,主要研究领域为需求工程,软件工程.
E-mail: xhchen@sei.ecnu.edu.cn