基于微分动态逻辑的CPS建模与属性验证

朱 敏,李必信,陈乔乔,吉顺慧,李加凯
(东南大学计算机科学与工程学院,江苏南京 211189)

摘 要: 随着信息物理融合系统(Cyber-Physical Systems,CPS)应用的越来越普及,CPS的设计和实现能否满足实际需求显得至关重要.本文提出了一种CPS建模与属性验证框架.在框架中,首先使用HybridUML对CPS进行建模,然后将该通用模型转换为形式化模型,进而进行形式化验证.本文采用的形式化验证方法为dL(Differential Dynamic Logic),其操作模型为hybrid program.将HybridUML模型转换为hybrid program时,基于语义一致性的原则定义转换规则.转换完成后,结合得到的hybrid program对验证的CPS属性进行规约,最后使用定理证明器KeYmaera对属性进行自动化验证.

关键词: 信息物理融合系统;微分动态逻辑;HybridUML;模型转换;验证
中图分类号: TP301 **文献标识码:** A **文章编号:** 0372-2112 (2012) 06-1126-07
电子学报URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2012.06.010

Transforming HybridUML to Hybrid Program for CPS Property Verification

ZHU Min, LI Bi-xin, CHEN Qiao-qiao, JI Shun-hui, LI Jia-kai
(*School of Computer Science and Engineering, Southeast University, Nanjing, Jiangsu* 211189, *China*)

Abstract: With the wide application of cyber-physical systems(CPS), it is of vital importance to ensure that the design and implementation of CPS meet critical performance requirements (e.g., security, reliability). This paper presents a framework for CPS modeling and verification. In the framework, CPS are modelled by HybridUML and then the generic model is transformed to a formal model that is suitable for reasoning with the help of formal methods. The formal method adopted is differential dynamic logic (dL) whose operational model is hybrid program(HP). When transforming a HybridUML model to its corresponding HP presentation, transformation rules are defined according to semantic consistency. After transformation, CPS properties are specified on the basis of the resulting HP, and finally, the properties are automatically verified through the theorem prover KeYmaera.

Key words: CPS; differential dynamic logic; HybridUML; model transformation; verification

1 引言

在信息化和网络化的浪潮中,物理设备的计算,通信和控制能力日益增强,将三者融为一体的信息物理融合系统(Cyber-Physical Systems, CPS)逐渐成为人们关注的热点.CPS将物理过程与计算相结合,通过嵌入式系统和网络对物理设备进行监测与控制,计算与物理过程通过反馈机制相互影响[1].随着信息与物理融合越来越深入,CPS将会深远地影响着人们的生活,生产等各个方面.CPS能否满足设计需求,对系统设计实施来说至关重要,验证技术可以确定系统是否满足某些属性,弥补了传统的测试技术无法证明系统不存在缺陷的不足,在提高和保障系统安全性,可靠性等方面起到了关键的作用.形式化验证技术利用数学方法来验证规约的属性是否正确.其中基于模型检验技术的方法更多的是应用于有限状态的系统,而CPS具有离散与连续的行为,存在无限的状态,虽然有很多学者对其进行了扩展以验证无限状态系统[2,3],但由于在解决可达性问题方面的普遍限制[4,5]等原因,模型检验应用于验伪而非验真.基于推理的验证方法如,周巢尘等通过在状态变量的导数中引入数学表达式对时段演算进行扩展[6],利用积分规则和外部数学方法对导数和连续性进行推理,但该方法难以进行自动推理.Platzer提出的微分动态逻辑(Differential Dynamic Logic, dL)[7]的方法有着较强的推理能力,并且可以实现推理自动化,有效地应用于CPS验证.该形式化方法操作模型为混合程序(Hybrid Program, HP)将

收稿日期:2011-09-22;修回日期:2012-03-14
基金项目:国家自然科学基金(No.60973149);博士点基金(No.20100092110022);中科院计算机科学国家重点实验室开放基金基金(No.SYSKF1110)

通用的模型转换为形式化的模型进行验证的方法是软件工程领域研究的热点[8],它既能对系统建立直观易懂模型又可采用形式化方法进行推理验证.由于 UML 没有精确语义[9]且不能对连续状态建模,HybridUML[10]是基于 UML profile 机制对 UML 的一种扩展,弥补了 UML 在对 CPS 建模方面的不足,同时又保持了通用直观的特性以及 UML 状态机的基本语义[11].本文使用 HybridUML 对 CPS 进行建模,并提出一种将 HybridUML 模型转换为 HP 的模型转换方法,进而对转换后的 HP 进行推理验证.

2　基于 dL 方法对 CPS 建模与验证的框架

基于 dL 方法进行 CPS 建模验证分为两个部分.一是将 HybridUML 所建模型转换为 HP,二是利用得到的 HP 生成 KeYmaera[12]的输入代码.KeYmaera 是可交互的自动化定理证明器,支持 dL 等逻辑.本文主要讨论第一部分,从 HP 生成 KeYmaera 输入代码仅涉及到代码格式调整以及操作符替换等操作则不详细阐述.首先根据模型转换一致性的要求建立模型转换的规则,然后生成应用规则的模板,按照模板生成 HP,利用 dL 规约属性公式,最后依据 KeYmaera 的输入格式,格式化属性公式到 KeYmaera 进行验证.

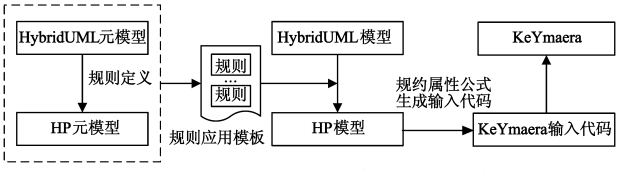


图1　基于微分动态逻辑方法验证CPS的框架

3　HybridUML 建模及扩展

在 HybridUML 中,Agent 和 Mode 分别是静态结构和动态行为的基本建模元素.Bisanz 在文献[9]中定义了表示 HybridUML 元模型的形式化描述,我们基于该形式化描述进行模型转换工作,增加以下几个定义:

(1)Mode 与 Agent 的分类,根据 Mode 是否有子 Mode 将 Mode 分为复合 Mode 和原子 Mode.子 Mode 不为空定义为复合 Mode,反之定义为原子 Mode.Agent 分类类似.以下点号(·)前后分别为前提和结论.

$\mathrm{kind}_M: M \to \{\mathrm{CompositeMode}, \mathrm{PrimitiveMode}\}$

$\mathrm{kind}_A: A \to \{\mathrm{CompositeAgent}, \mathrm{PrimitiveAgent}\}$

$\forall\, m \in M, \mathrm{submode}_M(m) \neq \varnothing \cdot (\mathrm{kind}_M(m) := \mathrm{CompositeMode}$

$\forall\, a \in A, |\mathrm{behavior}_A| = \varnothing \cdot \mathrm{kind}_A(a) := \mathrm{CompositeAgent}$

$\forall\, m \in M, \mathrm{submode}_M(m) = \varnothing \cdot \mathrm{kind}_M(m) := \mathrm{PrimitiveMode}$

$\forall\, a \in A, |\mathrm{behavior}_A| = 1 \cdot \mathrm{kind}_A(a) := \mathrm{PrimitiveAgent}$

(2)只有原子 Agent 才包含顶层 Mode,用来表示其行为 $\mathrm{TM}_A: A \to \mathrm{behavior}_A(A)$, $\forall\, a \in A \cdot \mathrm{kind}_A(a) = \mathrm{PrimitiveAgent}$

(3)src_T 和 tar_T 分别表示变迁的源和目标控制点,我们新增变迁的源 Mode 和目标 Mode 的表示:$\mathrm{srcMode}_T: T \to \{M(\mathrm{MI}\}$ $\mathrm{tarMode}_T: T \to \{M(\mathrm{MI}\}$变迁不仅在 Mode 与其子 Mode 实例间交换控制,子 Mode 实例之间也有控制交换,所以 $\mathrm{srcMode}_T$ 和 $\mathrm{tarMode}_T$ 必需满足:$\forall\, t \in T \cdot \mathrm{srcMode}_T(t) \in M \wedge \mathrm{tarMode}_T(t) \in \mathrm{MI} \vee \mathrm{srcMode}_T(t) \in \mathrm{MI} \wedge \mathrm{tarMode}_T(t) \in \mathrm{MI} \vee \mathrm{srcMode}_T(t) \in \mathrm{MI} \wedge \mathrm{tarMode}_T(t) \in M$

(4)根据变迁源和目标控制点的种类将 Mode 的变迁分三种类型:进入变迁集,退出变迁集和内部变迁集.

■ 进入变迁集(EntryTransitions),表示变迁源为 Mode 的进入控制点,变迁目标为子 Mode 实例的进入控制点的变迁的集合.

$\mathrm{EntryTransitions}_T: M \to T$

$\forall\, t \in T \cdot \mathrm{src}_T(t) \in \mathrm{CP} \wedge \mathrm{kind}_{\mathrm{CP}}(\mathrm{src}_T(t))$

$= \mathrm{entry} \wedge \mathrm{tar}_T(t) \in \mathrm{CPI} \wedge \mathrm{kind}_{\mathrm{CP}}(\mathrm{cp}_{\mathrm{CPI}}(\mathrm{tar}_T(t)))$

$= \mathrm{entry}$

■ 内部变迁集(InternalTranstions),表示复合 Mode 内部子 Mode 实例之间变迁的集合.

$\mathrm{InternalTransitions}_T: M \to T$

$\forall\, t \in T \cdot \mathrm{src}_T(t) \in \mathrm{CPI} \wedge \mathrm{kind}_{\mathrm{CP}}(\mathrm{cp}_{\mathrm{CPI}}(\mathrm{src}_T(t)))$

$= \mathrm{exit} \wedge \mathrm{tar}_T(t) \in \mathrm{CPI} \wedge \mathrm{kind}_{\mathrm{CP}}(\mathrm{cp}_{\mathrm{CPI}}(\mathrm{tar}_T(t)))$

$= \mathrm{entry}$

■ 退出变迁集(ExitTransitions),表示变迁源为子 Mode 实例的退出控制点,变迁目标为 Mode 的退出控制点的变迁的集合.

$\mathrm{ExitTransitions}_T: M \to T$

$\forall\, t \in T \cdot \mathrm{src}_T(t) \in \mathrm{CPI} \wedge \mathrm{kind}_{\mathrm{CP}}(\mathrm{cp}_{\mathrm{CPI}}(\mathrm{src}_T(t)))$

$= \mathrm{exit} \wedge \mathrm{tar}_T(t) \in \mathrm{CP} \wedge \mathrm{kind}_{\mathrm{CP}}(\mathrm{tar}_T(t))$

$= \mathrm{exit}$

4　dL 验证方法与 Hybrid Program

利用 dL 对混成系统进行验证的方法是基于定理证明的形式化方法,其工作机理与状态空间规模无关,因而避免了状态空间爆炸问题.dL 实际上是带模态的一阶动态逻辑,使用 HP 作为操作模型对系统进行建模,系统属性用 dL 公式表示,Platzer 给出了 dL 的演算规则以及规则的正确性证明,通过演算规则对属性公式进行推理验证.该方法同样可应用于 CPS[13].本文给出基本的 HP 表示:

HPModel = (InitBlock, DJ, CE, HPSkeleton, HPContent)

InitBlock 表示初始化块,包含 HP 所用到变量的声明以及初始化;DJ 表示离散变迁的集合,每个离散变迁由离散跳集的元素组成;CE 表示连续变化的集合,每个连续变化行为由微分方程组成;HPSkeleton 表示 HP 的

框架,HP 的具体实现包含于 HPContent;HPContent 由控制结构连接的 DJ 和 CE 集合的元素组成.

5 模型转换一致性保证

保持 HybridUML 模型与 HP 语义一致是进行模型转换工作的前提.本文从宏观和微观两个角度说明.从宏观上保证语义一致性,即要求 HP 必须能够满足 HybridUML 模型语义的要求,HP 和 HybridUML 的 Mode 都表现为连续变迁和离散变迁的相互交替,两者宏观语义一致.从微观上保证语义一致性,即应用于模型映射的形式化语言 S 和 D 应该满足如下约束:S 和 D 在语义上的对等,即 S 的概念集在 D 中具有语义对等的概念集,反之亦然[14].所以在进行 HybridUML 模型向 HP 转换之前,须确保两者概念集对等.由于并非所有的 HybridUML 模型元素都能映射到 HP,必须对 HybridUML 元模型进行重构以保证重构后的源元模型概念集在目标元模型都能找到对应的非空子集.图 2(a)和(b)分别是本文根据 MOF 标准定义的 HybridUML 和 HP 的元模型,Mode 是分层的状态机扩展自 UML 的 State 和 StateMachine,对应于 HPState 和 HP,所以在进行转换时必须将 Mode 的层次展开;HybridUML 元模型中 ModeTransition 表示离散变迁,变迁源和目标都是 Mode,而 HP 中的离散变迁(HP_D_Transition)的变迁源和目标皆为原子状态,转换时必须找到 ModeTransition 的变迁的原子状态才可与 HP 相对应,两者离散变迁中赋值行为(UpdateActivity),改变事件(ChangeEvent)都一一对应,Mode 的微分约束(Constraints)对应于 HP 中的 HP_C_Transition 表示连续变迁;HybridUML 中约束的表达式有三种:微分表达式(DifferentiaExpression),代数表达式(AlgebraicExpression)和不变量表达式(InvariantExpression).HybridUML 并不严格限制所用的表达式语言[9],为了与 HP 表达类型一致,在 HybridUML 建模时直接使用一阶逻辑相应的表达式.综上,在模型转换时必须根据上述概念集重构要求建立转换规则才可保证语义一致.

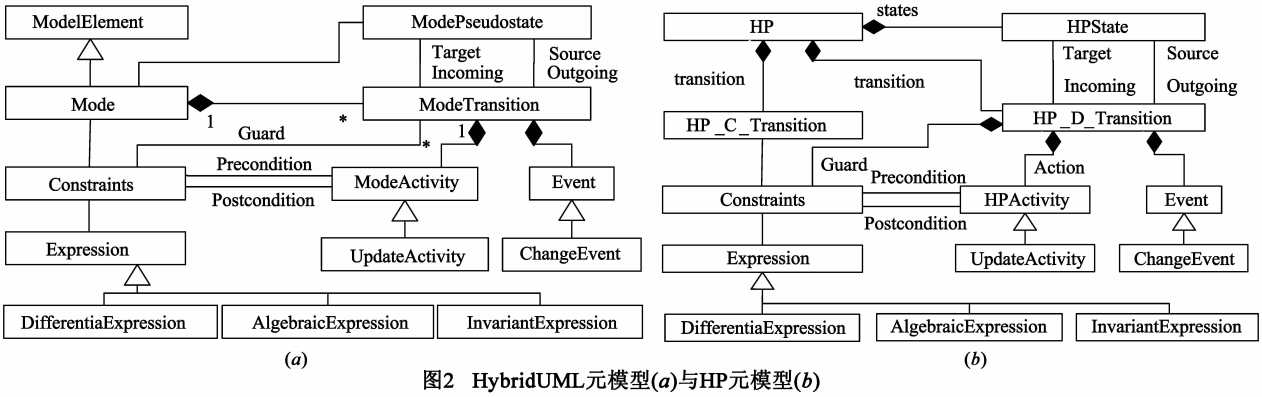


图2 HybridUML元模型(a)与HP元模型(b)

6 建立模型转换规则

上节模型转换一致性保证是以下模型转换工作的前提,在保证转换前后概念集语义对等的基础上建立转换规则保证模型转换前后的一致性.本文采用模型驱动架构 MDA 中模型转换的思想[15],根据系统需求选择具体的建模与模型转换方法以指导模型驱动的开发过程[16].在元模型层次根据源元模型和目标元模型元素间的关联定义转换规则,在模型层次应用规则进行模型转换.按照功能的不同将规则分为映射规则和处理规则两类.映射规则将源元模型中的元素在满足一定约束情况下映射到目标元模型;处理规则是处理源模型得到转换媒介或对源模型优化以便于进行转换.从 HybridUML 模型到 HP 的转换包括静态结构和动态行为规则的建立.

6.1 静态结构转换规则

在 HybridUML 模型中,类图和组成结构图用来表示系统静态结构.类图描述了系统 Agent 的定义及其之间的相互关系,而组成结构图包含了 Agent 的内部实现,如定义的属性,事件等.我们建立一个共享变量表,用来标明不同 Agent 之间共享的变量,在转换后的 HP 中进行共享变量统一命名处理.建立共享变量表的规则命名为 CreateShareVariableTable.本文考虑所有原子 Agent 中至多只有一个拥有连续动态行为的情况.假设 A_P 为原子 Agent 的有限集合,$A_P \in A \cup AI$,$\forall a \in A_P \cdot \mathrm{kind}_A(a) = \mathrm{PrimitiveAgent}$.Agent 之间是并行的,在 HP 中这样处理并行:设 $a_0 \in A_P$ 为具有连续动态行为的原子 Agent,a_0 的状态的连续变化不考虑通信过程,只受其它 Agent 在离散时间点上与其通信的影响.通过不确定选择操作符($\cup$)和重复操作符($*$)连结 a_0 和其他原子 Agent 的状态,对每个可能的时间点建模,而不需要考虑通信过程[13].映射的规则命名为 MappingStructureToHP.

6.2 动态行为转换规则

在 HP 中,原子 Mode 才是 HP 中变迁的源和目标状态,对于 HybridUML 模型的每个变迁,如果变迁的源或

目标是复合 Mode,必须找到其内的原子 Mode.根据转换语义一致性的要求,将 Mode 的层次展开,得到原子 Mode 组成的状态图再进行转换.首先将原子 Mode 继承其上层 Mode 的约束,并递归向上继承直至顶层 Mode.继承的规则命名为 MergeConstraints.

6.2.1　层次状态图展开

在 HybridUML 里变迁的源和目标控制点所属 Mode 可能是原子 Mode、复合 Mode 或交叉点(junction),而 HP 中变迁的源和目标都为原子状态,所以需要将层次状态图展开成原子 Mode 组成的简单状态图.经过 Mode 的层次约束继承后,HybridUML 状态图中的每个 Mode 都已继承其上层 Mode 的约束.对于状态图内的每个变迁,若变迁的源或目标 Mode 为复合 Mode,则需要向其上层或下层寻找变迁,直到变迁的源和目标都为原子 Mode.在寻找变迁路径过程中,如果变迁 t_2 以变迁 t_1 的目标控制点为源控制点,则在变迁集 T 中加入以 t_1 的源控制点为源控制点,t_2 的目标控制点为目标控制点的变迁,待处理完毕后将 t_1、t_2 从变迁集中删除,处理的规则命名为 CreateTransitionPath.对于交叉点处理类似,处理的规则命名为 EliminateJunction.

最后,进行层次状态图[17]的展开,根据变迁的几种不同类型分别处理.处理的规则命名为 FlatHierarchyMode,如图 3 所示.

```
ProcessingRule FlatHierarchyMode {
[Rule Input]
Mode tm
[Declaration]
TransitionSet:存储处理内部变迁时生成的临时变迁的集合,Add()方法向集合中添加项,Clear()清除集合元素,
ModeQueue:存储 mode 的队列,EnQueue()方法插入元素到队尾,GetHead()方法返回队列队首元素,DeQueue()方法删除队首元素
[ProcessingRule]
// 将 tm 的所有 submode 入队
ModeQueue.EnQueue(submode_M(tm))
// 处理所有进入变迁
for each t_M.ET in EntryTransition_M(tm)
  // 合并每个进入变迁的目标 mode 的进入变迁
  for each t_SM.ET in EntryTransition(tarMode_M(t_M.ET))
      if src_T(t_SM.ET) = tar_T(t_M.ET)
        then Transition t_Merge←MergeTransition(t_M.ET, t_SM.ET)
            // 将 t_SM.ET从变迁集中删除
            T = T \ {t_SM.ET}
  T = T \ {t_M.ET}
//处理所有退出变迁
for each t_M.XT in ExitTransition_M(tm)
  // 合并每个退出变迁的源 mode 的退出变迁
  for each t_SM.XT in ExitTransition(srcMode_M(t_M.XT))
      then Transition t_merge←MergeTransition(t_SM.XT, t_M.XT)
          // 将 t_SM.XT,从变迁集中删除
          T = T \ {t_SM.XT}
  T = T \ {t_M.XT}
// 处理所有内部变迁
for each t_M.IT in InternalTransition(tm)
  // 处理内部变迁的目标 mode 的进入变迁
  for each t_SM.ET in EntryTransition(tarMode_T(t_M.IT))
      if src_T(t_SM.ET) = tar_T(t_M.IT)
        then Transition t_merge←MergeTransition(t_SM.ET, t_M.IT)
            // 将得到的新变迁加入到变迁集合中
            TransitionSet.Add(t_Merge)
            T = T \ {t_SM.ET}
// 处理内部变迁的源 mode 的退出变迁
for each t_SM.XT in ExitTransition(srcMode_T(t_M.IT))
    if tar_T(t_SM.XT) = src_T(t_M.IT)
        then for each t_new in TransititonSet
            Transition t_merge←MergeTransition(t_SM.XT, t_new)
            T = T \ {t_M.IT}
    T = T \ {t_M.IT}
    // 清空临时变迁集合
    TransitionSet.Clear()
// 队列非空时,处理队列
while ModeQueue.NotEmpty
    do FlatHierarchyMode(ModeQueue.GetHead())
        // 删除队列头
        ModeQueue.DeQueue()
[Return Result]
    return tm
}
```

图 3　规则 FlatHierarchyMode

6.2.2　变迁转换

展开后顶层 Mode 其内部仅含原子 Mode 及其之间的变迁,顶层 Mode 的进入变迁转换为 HP 的 InitBlock,原子 Mode 及其间的变迁分别对应于 HP 中的 CE 和 DJ.建立映射规则 MappingETtoInitBlock 将顶层 Mode 的进入变迁映射为 HP 中的 InitBlock.我们将原子 Mode 及其间的变迁关系组成的图称之为 TransitionGraph,对应于 HP 中的 HPContent,定义如下:

定义　TransitionGraph(M_{TG}, T_{TG})是状态图展开后由原子 Mode 及其之间的变迁组成的有向图,其中图的顶点集 M_{TG}为 HybridUML 模型中原子 Mode 的集合,$\forall m \in M_{TG} \cdot \mathrm{kind}_M(m) = \mathrm{PrimitiveMode}$, T_{TG}为边的集合 $\forall t \in T_{TG} \cdot \mathrm{kind}_M(\mathrm{srcMode}_T(t)) = \mathrm{PrimitiveMode} \wedge \mathrm{kind}_M(\mathrm{tarMode}_T(t)) = \mathrm{PrimitiveMode}$,边之间关系 $t = \{ < v, w > | t(T_{TG} \cdot v = \mathrm{srcMode}_M(t), w = \mathrm{tarMode}_M(t)\}$,表示以 v 为源 Mode,w 为目标 Mode 的变迁.

经过上述处理后,得到 TransitionGraph,其包含的 HybridUML 的两类动态行为:离散变迁和连续变化,分别对应于 HP 的离散变迁和连续变迁.HybridUML 离散

变迁中的触发事件,变迁条件,以及变迁执行动作分别与 HP 离散变迁相应元素对应,为了标识 HP 中当前活动的状态,在 InitBlock 中新增一个标记变量 ActiveState. 变迁的转换规则命名为 MappingTGtoHP,如图 4 所示.

```
MappingRule MappingTGtoHP {
[Rule Input]
    TransitionGraph tg
[Declaration]
    HPContent hpcontent
    // Trans_TG表示 TransitionGraph 中离散变迁的集合
    Trans_TG:TG→T_TG
    // Mode_TG表示 TransitionGraph 中原子 Mode 的集合
    Mode_TG:TG→M_TG
[Mapping]
// 离散变迁转换
for each t in Trans_TG(tg)
    hpcontent←(? hp.ActiveState = srcMode_T(t);? grd_T(t);
    act_T(t);hp.ActiveState: = tarMode(t))(hpcontent
// 连续变化转换
for each m in Mode_TG(tg)
    hpcontent←(? hp.ActiveState = m;flow_M(m) & inv_M(m))(hpcon-
tent
[Return Result]
    return hpcontent
}
```

图 4 规则 MappingTGtoHP

7 实例分析

欧洲列车控制系统(European Train Control System, ETCS)遵循分段行驶原则[18],列车只允许在其当前指定的移动授权 MA 内行驶,无线闭塞控制器 RBC 根据其管辖范围内列车运行情况动态分配 MA. 图 5[18]是 MA 的动态分配示意图. 本例验证动态分配 MA 协议的安全性,即列车是否始终在分配给其的 MA 内行驶. 首先用 HybridUML 建模,得到 ETCS 的类组成图,系统组成结构图以及状态图(图 6, 7). ETCS 由 Train 和 RBC 两个 Agent 组成,其中 Train 具有连续的动态行为,变量 A, b 分别表示列车最大的加速度和最大的减速度; m 表示当前所在的 MA; z 和 v 表示列车在当前 MA 的位置和速度; t 是自动列车保护单元动态确定的安全运行时间,在此时间内列车可安全行驶; ε 表示列车最多可以运行的时间,超过则需要重要调整列车行驶状态; message 表示当前行驶是否有紧急情况,如果有则列车就减速; recommedspeed 表示在 MA 内的推荐速度,超过该速度即应减速. ETCS 的组成结构图由类结构图中的 Agent 的实例组成, Train 和 RBC 之间共享了 message 和 recommedspeed 变量. 当列车超出 SB 时,假设当前以最大加速度 A 进行行驶,通过 KeYmaera 可计算式(1)得到 SB 的最小值.

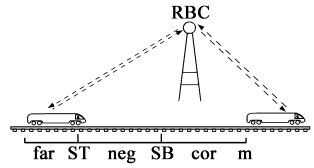


图5 ETCS 列车MA动态分配示意图

公式 Λ 如下:

$$(\forall m \forall z(m-z) \geqslant \mathrm{SB} \wedge v^2 \leqslant 2b(m-z) \wedge v \geqslant 0 \wedge b \geqslant 0 \wedge A \geqslant 0) \rightarrow [a:=A; z'=v, v'=a, t'=1 \,\&\, v \geqslant 0 \wedge t \leqslant \varepsilon](z \leqslant m) \equiv \mathrm{SB} \geqslant \frac{v^2}{2b} + \left(\frac{A}{b}+1\right)\left(\frac{A}{2}\varepsilon^2 + \varepsilon * v\right) \quad (1)$$

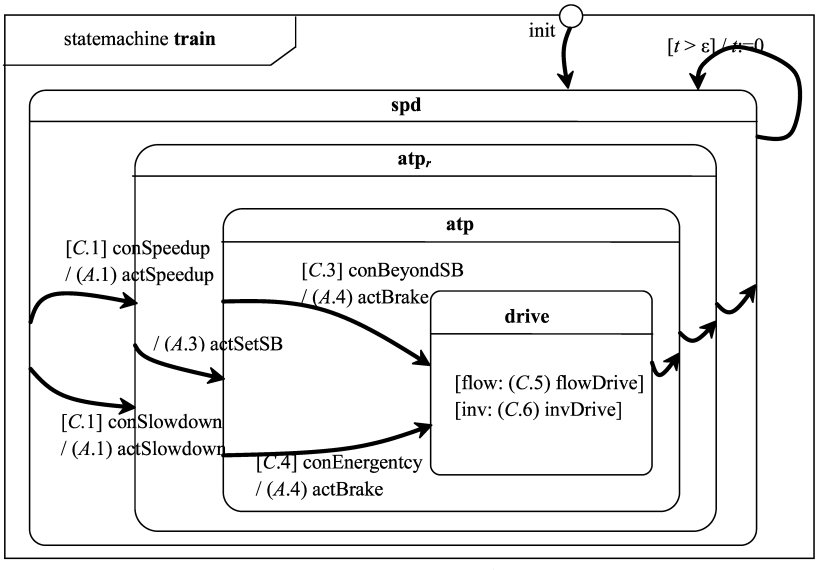


图6 Agent Train的状态图

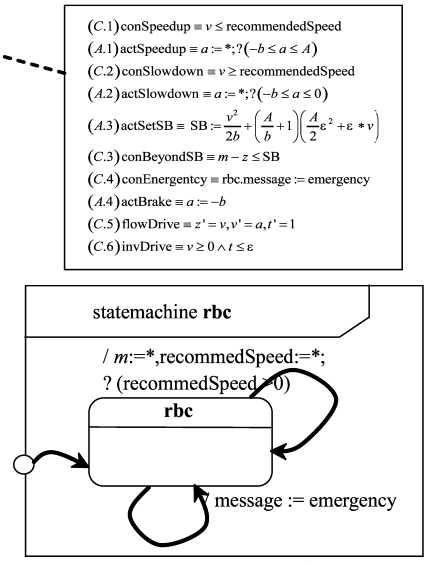


图7 Agent RBC的状态图

利用 HybridUML 对 ETCS 建模后,经转换到 HP 如下:

$\mathrm{ETCS} \equiv (\mathrm{train} \cup \mathrm{rbc})^*$

$\mathrm{train} \equiv \mathrm{ctrl}; \mathrm{drive}$

$\mathrm{ctrl} \equiv \sigma_1 \cup \sigma_2 \cup \sigma_3 \cup \sigma_4$

$\boldsymbol{\sigma}_1 \equiv (?\ \mathrm{ActiveState} = \mathrm{drive}; ?\ (v \leqslant \mathrm{recommedspeed});$

$a := *; ?\ (-b \leqslant a \leqslant A);$

$\mathrm{SB} := \frac{v^2}{2b} + (\frac{A}{b} + 1)(\frac{A}{2}\varepsilon^2 + \varepsilon * v);$

$(?\ (m - z \leqslant \mathrm{SB} \vee \mathrm{message} = \mathrm{emergency});$

$a := -b; \mathrm{ActiveState} := \mathrm{drive})$

$\boldsymbol{\sigma}_1 \equiv ?\ \mathrm{ActiveState} = \mathrm{drive}; ?\ (v \leqslant \mathrm{recommedspeed});$

$a := *; ?\ (-b \leqslant a \leqslant A);$

$\mathrm{SB} := \frac{v^2}{2b} + (\frac{A}{b} + 1)(\frac{A}{2}\varepsilon^2 + \varepsilon * v);$

$(?\ (m - z \leqslant \mathrm{SB} \wedge \mathrm{message}!\ = \mathrm{emergency});$

$a := -b; \mathrm{ActiveState} := \mathrm{drive})$

$\boldsymbol{\sigma}_1 \equiv ?\ \mathrm{ActiveState} = \mathrm{drive}; ?\ (v \geqslant \mathrm{recommedspeed});$

$a := *; ?\ (-b \leqslant a \leqslant 0);$

$\mathrm{SB} := \frac{v^2}{2b} + (\frac{A}{b} + 1)(\frac{A}{2}\varepsilon^2 + \varepsilon * v);$

$(?\ (m - z \leqslant \mathrm{SB} \vee \mathrm{message} = \mathrm{emergency});$

$a := -b; \mathrm{ActiveState} := \mathrm{drive})$

$\boldsymbol{\sigma}_1 \equiv ?\ \mathrm{ActiveState} = \mathrm{drive}; ?\ (v \geqslant \mathrm{recommedspeed});$

$a := *; ?\ (-b \leqslant a \leqslant 0);$

$\mathrm{SB} := \frac{v^2}{2b} + (\frac{A}{b} + 1)(\frac{A}{2}\varepsilon^2 + \varepsilon * v);$

$(?\ (m - z \leqslant \mathrm{SB} \wedge \mathrm{message}!\ = \mathrm{emergency});$

$a := -b; \mathrm{ActiveState} := \mathrm{drive})$

$\mathrm{drive} \equiv ?\ \mathrm{ActiveState} = \mathrm{drive};$

$t := 0; (z' = v, v' = a, t' = 1\ \&\ v \geqslant 0 \wedge t \leqslant \varepsilon)$

$\mathrm{rbc} \equiv \mathrm{message} := \mathrm{emergency}$

$\cup (m := *, \mathrm{recommedSpeed} := *; ?\ (\mathrm{recommedSpeed} > 0))$

再用 dL 公式进行系统属性规约,如下:

$$\Psi \rightarrow [\mathrm{ETCS}^*]z \leqslant m \qquad (2)$$

其中:$\Psi \equiv \mathrm{ActiveState} = \mathrm{drive} \wedge v^2 \leqslant 2b(m - z) \wedge b > 0 \wedge A \geqslant 0$; $\mathrm{ETCS} \equiv (\mathrm{ctrl}; \mathrm{drive}) \cup \mathrm{rbc}$, $\mathrm{ctrl} \equiv \sigma_1 \cup \sigma_2 \cup \sigma_3 \cup \sigma_4$.

Ψ 为初始条件.推理过程会产生推理分支,如果每个推理分支都能推出显而易见的表达式则推理结束.在 KeYmaera 中进行验证用了 236 步,产生 10 个推理分支,得到在满足初始条件 Ψ 下式(2)成立,即在满足 Ψ 的约束下,列车在行驶的任何状态都不会超出分配给其的 MA,每列列车都满足该属性列车之间不会发生碰撞.

8　结束语

本文提出一种基于 dL 的 CPS 建模与属性验证方法,利用 HybridUML 对 CPS 建模,提出一种模型转换的方法,将 HybridUML 模型转换为 dL 方法的操作模型 HP,再利用 dL 公式对 CPS 属性进行规约,将得到的属性作为定理证明器 KeYmaera 的输入,从而进行验证.下一步工作将主要包括模型转换的完善,CPS 其他属性的验证,以及为转换的自动化提供有效的工具支持等.

参考文献

[1] Lee EA. Cyber physical systems: Design challenges[A]. Proceedings of 11th IEEE Symposium on Object Oriented Real-time Distributed Computing (ISORC)[C]. Washington: IEEE Computer Society, 2008. 363 – 369.

[2] Chutinan A, Krogh, BH. Computational techniques for hybrid system verification[J]. IEEE Transactions on Automatic Control, 2003, 48(1): 64 – 75.

[3] Tiwari A. Approximate reachability for linear systems[A]. Proceedings of Hybrid Systems: Computation and Control[C]. Verlag: Springer, 2003. 514 – 525.

[4] Platzer A, Clarke, EM. The image computation problem in hybrid systems model checking[A]. 10workshop on Hybrid System: Computation and control[C]. Heidelberg: Springer, 2007. 473 – 486.

[5] Collins P, Lygeros J. Computability of finite-time reachable sets for hybrid systems[A]. Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference[C]. New Jersey: Piscataway, 2005. 4688 – 4693.

[6] Zhou CC, Hansen MR. Duration Calculus: A Formal Approach to Real-Time Systems[M]. Heidelberg: Springer, 2004. 41 – 62.

[7] Platzer A. Differential dynamic logic for hybrid systems[J]. Journal of Automated Reasoning, 2008, 41(2): 143 – 189.

[8] 刘亚萍,黄志球,祝义. 基于元建模的实时系统模型转换方法研究[J]. 小型微型计算机系统, 2010, 31(11): 2146 – 2153.
Liu Yaping, Huang Zhiqiu, Zhu Yi. Research on model transformation method of real-time system based on metamodeling [J]. Journal of Chinese Computer Systems, 2010, 31(11): 2146 – 2153. (in Chinese)

[9] Stefan Bisanz. Executable HybridUML Semantics: A Transformation Definition[D]. Bremen: University of Bremen, 2005.

[10] Kirsten Berkenkötter, Stefan Bisanz, Ulrich Hannemann, Jan Peleska. The HybridUML profile for UML 2.0[J]. International Journal on Software Tools for Technology Transfer (STTT), 2006, 8(2): 167 – 176.

[11] 周颖,郑国梁,等. 面向模型检验的 UML 状态机语义[J]. 电子学报, 2004, 51(12): 2091 – 2095.
ZHOU Ying, ZHENG Guo-liang, et al. An operational semantics for UML State Machines in model checking context[J].

Acta Electronica Sinica, 2004, 51(12): 2091 – 2095. (in Chinese)

[12] Platzer A, Quesel JD. KeYmaera: A hybrid theorem prover for hybrid systems[A]. International Joint Conference on Automated Reasoning (IJCAR)[C]. Heidelberg: Springer, 2008. 171 – 178.

[13] Platzer, A. Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics[M]. Heidelberg: Springer, 2010.

[14] Caplat G, Sourrouille J-L. Model mapping using formalism extensions[J]. IEEE Software, 2005, 22(2): 44 – 51.

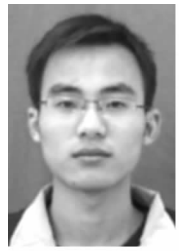
[15] Czarnecki K, Helsen S. Classification of model transformation approaches[A]. OOPSLA'03 Workshop on Generative Techniques in the Context of Model-driven Architecture[C]. New York: ACM Press, 2003.

[16] 战德臣,冯锦丹,等. ICEMDA:一种可互操作可配置可执行的模型驱动体系结构[J]. 电子学报, 2008, 36(12A): 120 – 127.
ZHAN De-chen, FENG Jin-dan, et al. ICEMDA: An interoperable configurable executable model driven architecture[J]. Acta Electronica Sinica, 2008, 36(12A): 120 – 127. (in Chinese)

[17] 赖明志,尤晋元. 从 UML 状态图到 PVS 规范的自动转换、验证[J]. 电子学报, 2002, 30(12A): 2122 – 2125.
LAI Ming-zhi, YOU Jin-yuan. Automatic transform UML statechart into PVS[J]. Acta Electronica Sinica, 2002, 30(12A): 2122 – 2125. (in Chinese)

[18] Platzer A, Quesel, J. D. European Train Control System: A case study in formal verification[A]. Formal Methods and Software Engineering: 11th International Conference on Formal Engineering Methods[C]. Heidelberg: Springer, 2009. 246 – 265.

作者简介



朱 敏 男,1988 年出生于安徽省庐江县,现为东南大学计算机科学与工程学院在读硕士研究生,主要研究方向为 CPS 验证.
E-mail: kong@seu.edu.cn

李必信(通信作者) 男,1969 年出生于安徽省庐江县,博士/博士后,现任东南大学计算机科学与工程学院教授、博士生导师, CCF 高级会员,主要研究领域为软件分析、测试、验证,实证软件工程.
E-mail: bx.li@seu.edu.cn

陈乔乔 男,1987 年出生湖北省荆门市,现为东南大学计算机科学与工程学院在读硕士研究生,主要研究方向为 CPS 验证.
E-mail: joe_0701@126.com

吉顺慧 女,1987 年出生江苏省海安市,现为东南大学计算机科学与工程学院在读博士研究生,主要研究方向为 Web 服务验证.
E-mail: shunhuiji@seu.edu.cn

李加凯 男,1987 年出生山东省潍坊市,现为东南大学计算机科学与工程学院在读硕士研究生,主要研究方向为 CPS 验证.
E-mail: jiakai_li@seu.edu.cn