软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(6):1779−1798 [doi: 10.13328/j.cnki.jos.006249]

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

面向 CPS 时空性质验证的混成 AADL 建模与模型转换方法*

陈小颖[1], 祝 义[1], 赵 宇[1], 王金永[2]

[1](江苏师范大学 计算机科学与技术学院,江苏 徐州 221116)

[2](南京航空航天大学 计算机科学与技术学院,江苏 南京 211106)

通讯作者: 祝义, E-mail: zhuy@jsnu.edu.cn

摘 要: 随着信息物理融合系统 CPS(cyber physical system)研究的深入,CPS 的安全性问题越来越受到人们的广泛关注,如何验证CPS时空不一致的安全性问题已经成为研究热点.针对该问题,提出了面向CPS时空性质验证的混成 AADL(architecture analysis & design language)建模与模型转换方法.首先,扩展 AADL 行为附件的时空描述能力,提出了混成 AADL(hybrid architecture analysis & design language),用于建模 CPS 的时空性质;其次,在进程代数中引入微分方程以及位置描述,提出了 HP-TCSP,能够验证 CPS 的时空性质;再次,通过模型转换,将混成 AADL 转换为 HP-TCSP,从而可以将混成 AADL 描述的 CPS 模型在 HP-TCSP 中进行时空一致性验证;最后,通过一个飞机避撞系统实例,验证该方法的有效性.

关键词: 信息物理融合系统;时空性质;进程代数;AADL;形式化验证

中图法分类号: TP311

中文引用格式: 陈小颖,祝义,赵宇,王金永.面向 CPS 时空性质验证的混成 AADL 建模与模型转换方法.软件学报,2021,32(6): 1779−1798. http://www.jos.org.cn/1000-9825/6249.htm

英文引用格式: Chen XY, Zhu Y, Zhao Y, Wang JY. Hybrid AADL modeling and model transformation for CPS time and space properties verification. Ruan Jian Xue Bao/Journal of Software, 2021,32(6):1779−1798 (in Chinese). http://www.jos.org.cn/1000-9825/6249.htm

Hybrid AADL Modeling and Model Transformation for CPS Time and Space Properties Verification

CHEN Xiao-Ying[1], ZHU Yi[1], ZHAO Yu[1], WANG Jin-Yong[2]

[1](School of Computer Science and Technology, Jiangsu Normal University, Xuzhou 221116, China)

[2](College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China)

Abstract: With the in-depth research of CPS (cyber physical system), the security problems of CPS are gradually attracted extensive attention. How to verify the security of space and time inconsistency of CPS has become a research hot spot. A hybrid AADL (architecture analysis & design language) modeling and model transformation method for CPS is proposed to solve this problem. Firstly, the time and space description capability of AADL behavior annex is extended, and Hybrid AADL (hybrid architecture analysis & design language) is proposed. Secondly, the differential equation and the position description are introduced into the process algebra. Thirdly, the hybrid AADL is transformed into HP-TCSP. Finally, the effectiveness of the proposed method is verified by an example of aircraft anti-collision system.

* 基金项目: 国家自然科学基金(62077029); 徐州市应用基础研究计划(KC19004); 江苏省研究生科研创新计划(KYCX20_2380); 江苏省研究生科研创新计划(KYCX20_2384)

Foundation item: National Natural Science Foundation of China (62077029); Applied Basic Research Program of Xuzhou (KC19004); the Graduate Science Research Innovation Program of Jiangsu Province (KYCX20_2380); the Graduate Science Research Innovation Program of Jiangsu Province (KYCX20_2384)

本文由“形式化方法与应用”专题特约编辑姜宇副教授推荐.

收稿时间: 2020-08-30; 修改时间: 2020-10-26; 采用时间: 2020-12-19; jos 在线出版时间: 2021-02-07

Key words: CPS (cyber physical system); time and space properties; process algebra; AADL; formal verification

近年来,随着云计算、物联网与 5G 技术的快速发展,信息物理融合系统 CPS(cyber physical system)的应用已经覆盖到我们生产生活的各个方面,其安全性问题也受到了广泛关注,尤其在很多安全系数较高的 CPS 场景中,如飞机飞行控制系统,汽车控制系统,铁路控制系统等,系统的安全性验证[1]显得尤为重要.2016 年,一名司机驾驶特斯拉 Model S 在自动导航模式下与一辆牵引挂车碰撞发生车祸身亡.究其原因:是在明亮的晴朗天气,牵引挂车白色车体反光严重,汽车组件失灵未能识别而导致碰撞.2018 年,Uber 自动车在夜间环境中未能通过阴影识别行人[2],从而导致撞击行人致死.这些安全关键的 CPS 系统产生错误,将造成严重的后果.因此,需要一种安全性验证的方法对 CPS 系统安全性进行保证.而时间与空间属性是保证安全性的关键因素.所以,如何验证在复杂的运行环境下 CPS 的时空一致性以保障 CPS 的安全性,是当前面临的挑战.

体系结构分析与设计语言(hybrid architecture analysis & design language,简称 AADL)是于 2004 年由自动化工程师学会发布的建模语言[3],AADL 自提出以来便受到广泛关注.该语言基于系统构件层次化的结构设计方法,能够以统一的方式对硬件与软件进行抽象建模,支持高度可演化系统的开发.AADL 语言能提供所需构件之间的分层功能,同时支持文本和图形化的描述方式从而快速建模,其行为附件支持对性质的扩展,在 CPS 建模过程中具有很高的灵活性和实用性.因此,AADL 目前已经成为许多学者对 CPS 建模的首选语言,且在 CPS 领域广泛使用.文献[4]基于 AADL 对云信息物理融合系统建模,进行数据质量分析、实时性能分析和资源消耗分析;文献[5]将模型检查集成到设计过程中,根据 AADL 中的高级规范生成定时自动机模型,从而将模型检查技术集成到 CPS 设计过程中.然而,AADL 并没有足够的能力描述 CPS 的性质以及异构特性与连续行为等特点.因此,相关学者在 AADL 语言上进行相应的扩展,从而更好地对 CPS 进行研究.文献[6]构造了新的 AADL 子语言 AADL+,描述 CPS 连续行为以及网络组件和物理组件之间的交互;文献[7]提出了基于扩展 AADL 的 CPS 集成建模框架描述方法,实现计算实体、物理实体和交互实体的统一描述;文献[8]将 AADL 与 Simulink/Stateflow 结合,提供了统一的图形化协同建模形式,支持 CPS 软件、硬件和物理这 3 个角度统一设计;文献[9]针对 AADL 嵌入式系统体系结构进行可靠性建模,实现了 AADL 可靠性模型到广义随机 Petri 网可靠性计算模型的转换,并基于 GSPN 可靠性计算模型对嵌入式系统进行可靠性评估,设计并实现了 AADL 可靠性评估工具.此外,AADL 本身还是一种半形式化的建模语言,无法保证所建立的 CPS 模型的安全性,因此有必要对使用 AADL 建模的模型进行安全性验证.

通信顺序进程 CSP(communicating sequential process)是 Hoare[10]在 1978 年建立的一种适合于分布式并发软件规格和设计的形式化方法.1986 年,牛津的 Reed 和 Roscoe 对 CSP 进行了实时扩展,提出了时间通信顺序进程 TCSP(timed communicating sequential process)[11].进程代数是一种解决并发系统通信问题的形式化方法,能够描述 CPS 中的事件的并发、同步、异步等问题.但是 TCSP 是离散模型,且只有时间性质的描述能力,因此有必要对 TCSP 进行空间描述能力与连续行为能力的扩展,扩展后的 TCSP 能够对 CPS 的时空一致性进行验证.

通过以上分析,本文针对 CPS 时空不一致的安全性验证问题,提出了面向 CPS 时空性质验证的混成 AADL 建模与模型转换方法(如图 1 所示).

- 该方法首先在工业界广泛使用的建模语言 AADL 上扩展时空性质,使其具有建模 CPS 时空性质的能力,提出了混成 AADL(hybrid AADL,简称 HAADL);
- 其次,扩展了 TCSP 的条件执行算子与条件中断算子,提出了混成位置时间通信顺序进程 HP-TCSP (hybrid position TCSP),使其具有描述时空性质的能力;
- 再次,通过模型转换方法,将扩展后 AADL 所建立的模型转换成进程代数模型进行验证,对没有通过验证的 HAADL 模型修改其输入/输出参数,直至通过时空一致性验证;
- 最后,通过一个飞机避撞系统的实例来说明该方法在验证 CPS 时空安全性方面的有效性.

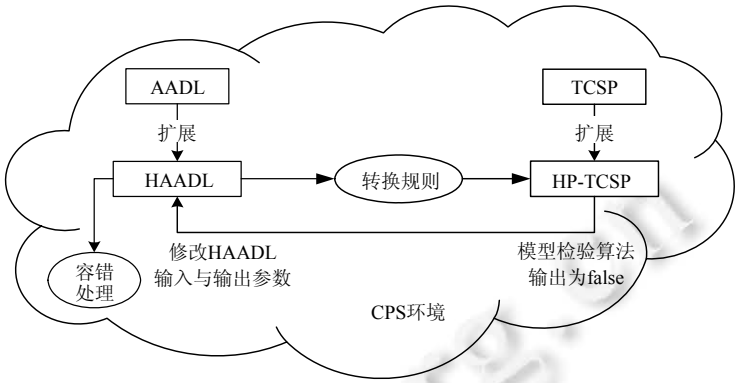

Fig.1 Technology roadmap

图 1 技术路线

1 混成 AADL

1.1 AADL的基础知识

体系结构分析与设计语言 AADL 定义了用于描述系统软硬件体系结构的构件和描述系统特性的特征、属性等语义,此外,还通过行为附件(behavior annex)支持 AADL 的扩展.AADL 的行为附件是一个简单的状态转换关系系统,包括状态与状态的迁移.完整的 AADL 描述包括软/硬件体系结构、执行模型以及行为附件.AADL 线程构件使用的是“Input-Compute-Output”计算模型,输入/输出行为是由线程的基本执行语义和线程通信语义决定的,而行为附件则是对计算行为进行细化和求精[12].

目前,AADL 建模工具都支持行为附件的描述,针对 AADL 不同的应用层次,各种仿真工具包括集成开发工具 OSATE、可调度分析工具 Cheddar、自动代码生成工具 Ocarina 等.

1.2 AADL组件的行为附件以及行为附件的扩展

本文的 HAADL 到 HP-TCSP 的模型转换是在元模型层次上的转换.元模型是对模型的抽象,是对模型的语法解释,需要在语法层次上建立 HAADL 到 HP-TCSP 的对应关系.在 TCSP 的进程代数方法上做扩展,其有单独的时间和空间位置变量.因此在 AADL 中,需要将 *times* 和 *positions* 抽象出来,进而可以将其变为元模型的抽象元素.

行为附件主要包括 3 部分:变量、状态和转换.变量部分声明当前行为附件中使用的所有局部变量,局部变量可以用来保存当前行为附件范围内的中间结果;状态迁移包括原状态、目的状态以及转换条件和执行动作,状态迁移定义了触发条件及迁移后的执行动作,条件和动作主要包括接收/发送数据、子程序调用、异步访问、执行时间、延迟时间等,且通过层次、并发状态来支持更复杂的行为描述.AADL 的行为附件可表示为:

annex behavior_specification

{**

⟨*variables*⟩

⟨*states*⟩

⟨*transitions*⟩

**}

AADL 的行为附件的抽象描述.

定义 1. 行为附件可以定义为三元组 *Action*:={*Vars*,*States*,*Transition*},其中:*Vars* 为变量集合;*States* 为状态集合;*Transition* 为状态迁移的集合,状态迁移 *Transition* 包含 *SourceState*,*Destination*,*Guard*,*Action*:

$$Transition:=SourceState[\langle guard\rangle]\rightarrow DestinationState[\{\langle action\rangle\}].$$

其中,*SourceState* 表示状态迁移之前的状态,目的状态 *DestinationState* 是状态 *SourceState* 在执行描述迁移后的状态,*guard* 表示执行该状态迁移的前提条件,*action* 为执行该状态迁移后完成的动作.

本文在原行为附件基础上扩展时间与空间的相关因素.扩展后的 AADL 行为附件可表示为:

annex behavior_specification(*extend*)

{**

⟨*variables*⟩

⟨*states*⟩

⟨*times*⟩

⟨*positions*⟩

⟨*transitions*⟩

**}

我们在原行为附件的基础上增加了时间⟨*times*⟩与位置⟨*positions*⟩.⟨*times*⟩部分,共涉及 5 个时间变量:*wait* 为执行该迁移的等待时间,即为使用 CPU 的时间;*compute* 为迁移的计算时间及迁移的持续时间,是任务被挂起或中断的时间;*ex_compute* 为当前迁移的期望执行时间;*ex_wait* 为当前迁移的期望等待时间;*worst_execute* 表示该行为附件计算的最坏执行时间,当执行模型端口被分配的时候可以获得.

我们在扩展中引入 3 个运算符.

- *vara*?*varb*:表示将变量 *varb* 的值赋给变量 *vara*,如 *speed*?*v* 表示将变量 *v* 的值赋给变量 *speed*;
- *vara*!*varb*:表示取出变量 *vara* 值放在 *varb* 中,如 *v*!*speed* 表示将变量 *v* 的值取出赋给变量 *speed*;
- |:该符号表示两个赋值动作互不影响,同时进行.

定义 2. 我们将包含时间与空间因素的行为附件定义为以下五元组:

$$ExAction:=\{Vars,States,Transition,Time,Position\},$$

其中,

- $Time:=\{wait,compute,ex\_compute,ex\_wait,worst\_execute\}$;
- $Position:=(x,y,z),x\sim c|y\sim c|z\sim c,c$ 是一个实数,$\sim\in\{>,\geqslant,<,\leqslant,=\}$,且三维坐标 x,y,z 按照$\varphi(x),\varphi(y),\varphi(z)$的微分方程变化.

该过程中涉及的赋值动作为 *action*,接下来我们用 *act* 代替:

$$Trans\_sequence:=SourceState[\langle act_1\rangle\langle guard\rangle]\rightarrow\{DesS_1[\{\langle act_2\rangle\}],DesS_2[\{\langle act_3\rangle\}],...,error]\},$$

其中,

- $\{DesS_1[\{\langle act_2\rangle\}],DesS_2[\{\langle act_3\rangle\},...,error]\}$为迁移序列,当 $DesS_1$ 在 *ex_wait* 时间未响应,则会执行迁移到状态 $DesS_2$;$DesS_2$ 未在期望等待时间响应,则执行迁移至状态 $DesS_3$.以此类推,直到执行迁移到最后一个 *error* 状态,进行容错处理;
- $act_i:=act.act$ 进行系统变量、时间变量与位置变量的赋值操作以及输入/输出操作,$act_i\in\{act_1,act_2,act_3,...\}$;
- $guard:=con\vee(wait\sim ex\_wait)\vee(compute\sim ex\_compute).\sim\in\{>,\geqslant,<,\leqslant,=\}$.*guard* 表示执行该迁移的前提条件,如果 *wait* 在指定等待时间 *ex_wait* 没有执行当前迁移,或执行时间 *compute* 不满足当前 *ex_compute* 值,或 *con* 中有条件不满足,则迁移至迁移序列的下一个状态直至迁移至容错状态 *error*;
- *error* 为错误状态,并执行预定的处理异常的操作进行容错处理.

1.3 AADL组件的物理附件的扩展

对 AADL 的物理组件进行扩展,从而对 CPS 中的物理行为进行建模,实现物理设备之间的通信.对传感器和执行器的连续行为建模可以作为抽象组件实现.

定义 3. 物理设备的混成附件定义为以下五元组:

$$PY:=\{process,var,interface,time,position\},$$

其中,*process* 为物理设备的进程;*var* 为物理设备的变量;*interface* 为物理设备的接口;*time* 为执行该物理设备之间的信息交换过程的期望时间;*position*:=(x,y,z),x~c|y~c|z~c,c 是一个实数,~∈{>,≥,<,≤,=}.

1.4 HAADL实例

带警报的水箱控制系统,其控制器是通过传感器感知水位.当水位处于最低值 *bottom* 时,若 t 时间内进水系统未响应时将会启动报警装置.当水位处于最高值 *top* 时,若在 t'时间内进水系统未响应时,将响应系统报警.水位随着时间变化,水位值 y 逐渐递增 k 或递减 k.

如图 2 为带警报的水箱控制系统 AADL 模型,该进程构件包含一个线程子构件,行为附件为其中线程子构件的计算部分的处理.对 AADL 行为附件的扩展,主要是线程中计算过程的扩展.传感器传过来的数据 *sensor_data* 通过连接端口传输输入给行为附件中的位置变量 y,通过行为附件的计算后,将命令数据 *command_data* 通过端口传给相应的响应系统进行响应.

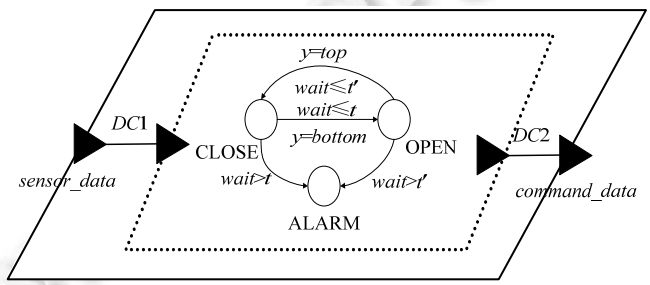

Fig.2 Graphical AADL model of water tank control system with alarm

图 2 带警报的水箱控制系统的图形化 AADL 模型

其中的 HAADL 行为附件描述为:

thread implementation example.impl
annex behavior_specification{**
variables
top:*Base_Types*::*Integer*;
bottom:*Base_Types*::*Integer*;
times
t:*Base_Types*::*float*;
t':*Base_Types*::*float*;
wait:*Base_Types*::*float*;
position
y:*Base_Types*::*Integer*;
states
open:*initial state*;
close:*complete state*;
alarm:*complete state*;
transitions
*T*_0:
close[⟨*y*=*bottom*⟩⟨(*wait*≤*t*)⟩]→{*open*[{*y*=*y*+*k*}],*alarm*]}
*T*_1:
open[⟨*y*=*top*⟩⟨(*wait*≤*t'*)⟩]→{*close*[{*y*=*y*−*k*}],*alarm*]}
**};
end *example.impl*;

2 混成位置-时间通信顺序进程 HP-TCSP

2.1 混成位置-时间通信顺序进程语法

定义 4. 对于一个系统 S,进程变量集合 PV:={X,Y,P,Q,...}表示 HP-TCSP 中的进程,离散变量集合 DV:={d,t,m,n,x,y,z,...}包括系统变量、时间变量以及位置中的坐标变量(默认 x,y,z 这 3 个符号代表三维坐标变量,其他变量中不使用该符号).

混成位置-时间通信顺序进程 HP-TCSP 可以定义为:

$$P ::= STOP \mid SKIP \mid WAIT\ t \mid a \to Q \mid P;Q \mid P \square Q \mid P \sqcap Q \mid P \overset{d}{\triangleright} Q \mid P \Delta Q \mid f(P) \mid P \setminus A \mid P\ {}_{A}\|_{B}\ Q \mid P \,|||\, Q \mid \mu X \cdot f(X) \mid$$
$$Con >> P(Con := Pf\ \&\&Place[\varphi(x),\varphi(y),\varphi(z)]\ \&\ Pcon \mid \text{true}) \mid P \blacktriangleright Fin\_Con(Fin\_Con := Fcon \mid \text{true}) \vdash Q$$

其中,

- $STOP$:停止,表示一个进程的中断,该进程不与外部发生通信,可表示死锁或进程不收敛;
- $SKIP$:跳过,表示一个进程除终止外不做任何事情;
- $WAIT\ t$:等待,表示进程在 t 时间后终止,其间不做任何事情;
- $a \to Q$:前缀操作,表示事件 a 执行完后执行进程 Q;
- $P;Q$:顺序复合,表示进程 P 执行完后执行进程 Q;
- $P \square Q$:外部选择,表示执行进程 P 或 Q 依赖于进程执行的第 1 个事件;
- $P \sqcap Q$:内部选择,表示执行进程 P 或 Q 由进程内部决定;
- $P \overset{d}{\triangleright} Q$:超时,如果 d 时间内两个进程未发生通信则认为超时,控制权由 P 交给 Q;
- $P \Delta Q$:中断,Q 的任何事件执行都能导致 P 的中断;
- $f(P)$:换标,$f(P)$和进程 P 有着同样的结构,只是 P 中的事件经过函数 f 映射为另一个名字;
- $P \setminus A$:集合隐藏,表示不显示进程 P 中任何属于 A 的事件;
- $P_A \|_B Q$:同步并发,P 和 Q 在集合 $A \cap B$ 的事件并发,在其他集合的事件交叉进行;
- $P \,|||\, Q$:异步并发,进程所执行的每个事件为进程 P 或 Q 中的一个事件;
- $\mu X \cdot f(X)$:$f(X)$是包含进程变量 X 的一个前缀表达式;
- $Con >> P(Con := Pf \&\& Place[\varphi(x),\varphi(y),\varphi(z)] \& Pcon \mid \text{true})$称为条件执行算子,在 Con 条件成立时,继续执行进程 P.Con 条件变量按照微分方程 Pf 连续变化,位置变量的三维坐标分别按照微分方程$\varphi(x)$,$\varphi(y)$, $\varphi(z)$ 的微分方程变化,并且微分方程 $Pf,\varphi(x),\varphi(y),\varphi(z)$中的变量满足谓词公式 $Pcon$,即 $Pcon$:=$x \sim c \mid y \sim c \mid z \sim c \mid Pcon_1 \wedge Pcon_2 \mid \text{true}$,其中,$c$ 是一个实数,$\sim \in \{>,\geqslant,<,\leqslant,=\}$.当条件中无需有位置约束时,则该条件默认为 true,始终成立.例如$(m=2n+1 \&\& Place[x=x+1] \& n>1 \wedge x>2) >> P$,即:当前 m 变量满足微分方程 $2n+1$,位置的 x 坐标按照 $x+1$ 变化.其中,变量 $n>1$,位置坐标 $x>2$ 时执行进程 P,否则不执行 P.以下 $Place[\varphi(x),\varphi(y),\varphi(z)]$也可简写为 $Place(x,y,z)$;
- $P \blacktriangleright Fin\_Con(Fin\_Con := Fcon \mid \text{true}) \vdash Q$ 称为条件中断算子,$Fcon$ 为谓词公式,是进程 P 终止执行时所涉及的变量的条件,默认为 false,可省略.如果条件 Fin_Con 满足,则中断正在执行的进程 P 进而执行进程 Q.如:$P \blacktriangleright (x<2) \vdash Q$ 表示当进程 P 执行时,x 坐标小于 2 时,终止 P 进程,执行 Q 进程,且在条件同时满足用“∧”,条件至少一个满足用“∨”.

TCSP 的基本操作 $a?x$ 表示进程通过通道 channel a 接收 x 的输入,TCSP 中提供通道表示一类事件的集合,例如:channel a:Int,表示通道 a 能够与任何带有整型数据的事件通信,事件 $a.2$ 是通道 a 声明的一个元素.$a!x$ 表示向通道 a 中输出消息 x.

2.2 HP-TCSP的混成位置-时间迁移系统

下面讨论 HP-TCSP 的混成位置-时间迁移系统 HP-TCSP 的语义.文献[13]将 TCSP 的语义定义为时间迁移系统,进程中一个事件的执行在时间迁移系统里可以看作为一个迁移,我们在研究 HP-TCSP 时引入混成位置-

时间迁移系统的概念.

定义 5. 一个时间通信顺序进程 TCSP 的语义描述为一个时间迁移系统 $TTS_{TCSP}=\langle NODES,\Sigma_T,\rightarrow\rangle$,$NODES$ 为节点集合,表示各个进程.Σ_T 是带延迟时间的事件集,即$\{(t_0,a_0),(t_1,a_1),\ldots,(t_n,a_n)\}$,$\rightarrow$是迁移关系,是一个三元关系,$\rightarrow\subseteq NODES\times\Sigma_T\times NODES$, $N_1\xrightarrow{(t,a)}N_2$,表示 N_1 执行事件 a,延迟 t 个时间单元变成 N_2 代表的进程.

定义 6. 一个混成位置-时间迁移系统 HP-TCSP 为 $HPTTS_{HP\text{-}TCSP}=\langle NODES,\Sigma_{(T,C,P)},\rightarrow\rangle$,$NODES$ 为节点集合,表示各个进程.$\Sigma_{(T,C,P)}$是带延迟时间、条件执行以及位置赋值操作的事件集,即$\{(p_0,c_0,t_0,a_0),(p_1,c_1,t_1,a_1),\ldots,(p_n,c_n,t_n,a_n)\}$,且$\Sigma_T\subseteq\Sigma_{(T,C,P)}$.当 $p_n=\varepsilon\wedge c_n=\text{true}$ 时,$\Sigma_T=\Sigma_{(T,C,P)}$.$\rightarrow$是迁移关系,是一个三元关系,$\rightarrow\subseteq NODES\times\Sigma_{(T,C,P)}\times NODES$, $N_1\xrightarrow{(p,c,t,a)}N_2$,表示 N_1 执行的进程进行 p 的位置赋值操作且在条件 c 满足的情况下,执行事件 a,延迟 t 个时间单元,变为 N_2 所代表的进程.

定义 7. 一个混成位置-时间通信顺序进程可以描述为一个混成位置-时间迁移系统:

$$HPTTS_{HP\text{-}TCSP}=\langle NODES,\Sigma_{(T,C,P)},\rightarrow\rangle.$$

2.3 HP-TCSP的操作语义

定义 8. HP-TCSP 的操作语义:

- 下面给出上面条件执行算子 $Con>>P(Con:=Pf\&\&Place[\varphi(x)][\varphi(y)][\varphi(z)]\&Pcon|true)$操作语义:

$$\frac{v=\langle v\mid v\neq pf(v)\rangle\vee x=\langle x\mid x\neq\varphi(x)\rangle\vee y=\langle y\mid y\neq\varphi(y)\rangle\vee z=\langle z\mid z\neq\varphi(z)\rangle\vee(Pcon=\text{false})}{s\xrightarrow{(p,c,t,a)}s} \tag{2.1}$$

$$\frac{v=\langle v\mid v=pf(v)\rangle\wedge x=\langle x\mid x=\varphi(x)\rangle\wedge y=\langle y\mid y=\varphi(y)\rangle\wedge z=\langle z\mid z=\varphi(z)\rangle\wedge(Pcon=\text{true})}{s\xrightarrow{(p,c,t,a)}s'} \tag{2.2}$$

只有当变量 v 是按照微分方程 $pf(v)$变化的,并且三维坐标 x,y,z 按照$\varphi(x),\varphi(y),\varphi(z)$的微分方程变化,且 $Pcon$ 为 true 时,状态 s 则在进行 p 的位置赋值操作且在条件 c 满足的情况下,执行事件 a,延迟 t 个时间单元,变为 s' 状态(2.2),否则不发生状态的迁移(2.1).

- 位置赋值算子 $P\blacktriangleright Fin\_Con(Fin\_Con:=Fcon|true)\vdash Q$ 的操作语义:

$$\frac{Fcon=\text{false}}{s\xrightarrow{(p,c,t,a)}s} \tag{2.3}$$

$$\frac{Fcon=\text{true}}{s\xrightarrow{(p,c,t,a)}s'} \tag{2.4}$$

只有当 $Fcon$ 为 true 时,即中断条件满足时,才会中断当前进程,当前状态从 s 状态迁移为 s'状态(2.4);否则,状态不发生改变(2.3).

2.4 精化关系

下面证明 TCSP 的语义模型是 HP-TCSP 的子语义模型:首先,对子语义模型进行定义;其次给出 HP-TCSP 包含 TCSP 语义的定理证明,证明 HP-TCSP 是在 TCSP 的基础上进行扩展.

定义 9. 设一个 A 类具有语义 M_A 的通信顺序进程 P_A 能够通过一个精化关系得到另一个 B 类具有语义 M_B 的通信顺序进程 P_B,那么称语义 M_B 是语义 M_A 的子语义模型[14].P_A 与 P_B 的精化模型如下:

$$\frac{P_B\ \text{sat}\ S_B\ \text{in}\ M_B}{P_A\ \text{sat}\ S_A\ \text{in}\ M_A}(P_B\sqsubseteq P_A).$$

S_A 是 P_A 的语义,S_B 是 P_B 的语义.

定义 10. 混成位置-时间通信顺序进程 HP-TCSP 所有可接受的语言是混成位置-时间迁移序列的集合.

定理 1. 时间通信顺序进程 TCSP 的语义模型是混成位置-时间通信顺序进程 HP-TCSP 的子语义模型.

证明:令 P_{TCSP} 是一个时间通信顺序进程.由定义 5 知,TCSP 是一个时间迁移系统 $TTS_{TCSP}=\langle NODES,\Sigma_T,\rightarrow\rangle$ 其接受的语言为 L,根据时间通信顺序进程构造一个混成位置-时间通信顺序进程,由定义 7 知,$P_{HP\text{-}TCSP}$ 是一个 $HPTTS_{HP\text{-}TCSP}=\langle NODES,\Sigma_{(T,C,P)},\rightarrow\rangle$.设 $P_{HP\text{-}TCSP}$ 能够接受的位置时间语言为 L'.此时,在 L 中任取一个时间迁移序列 $R=\langle(t_0,a_0),(t_1,a_1),\ldots,(t_{n-1},a_{n-1})\rangle$,在 L'中都有唯一一个 $R'=\langle(p_0,c_0,t_0,a_0),(p_1,c_1,t_1,a_1),\ldots,(p_{n-1},c_{n-1},t_{n-1},a_{n-1})\rangle$与之对

应,因此$\Sigma_T \subseteq \Sigma_{(T,C,P)}$.当 $p_n=\varepsilon \wedge c_n=$true 时,$\Sigma_T=\Sigma_{(T,C,P)}$,是 $P_{HP\text{-}TCSP}$ 到 P_{TCSP} 的精化关系.

那么,由定理 1 得,HP-TCSP 上所有与功能性质相关的模型检测都可以通过 TCSP 的模型检测工具完成.针对扩展的时空性质的一致性,需要进行时空一致性分析来判断是否满足时空一致性.在产生时空不一致时,进行一定的容错措施的处理.

2.5 时空属性的分析

HP-TCSP 的位置时间迁移系统是在 TCSP 的时间迁移系统上扩展位置因素而得到的,其状态空间可构造一个可达图 *G*.*G*:=(*NODES*,*EDGES*,*TIMES*,*POSITIONS*)其中,

- *NODES*={N_i|N_i 为进程执行的节点,0≤i≤n};
- *EDGES*={$e(i,j)$|表示从 N_i 到 N_j 的一条有向边,0≤i≤n,0≤j≤n,且 $i\neq j$};
- *TIMES*={$t(i,j)$|表示从 N_i 到 N_j 边 $e(i,j)$上的时间,若从 N_i 到 N_j 没有边,则 $t(i,j)=\infty$,0≤i≤n,0≤j≤n,且 $i\neq j$};
- *POSITIONS*={⟨$dis(i,j),p(j)$⟩|⟨$dis(i,j),p(j)$⟩中包含两个元素,$dis(i,j)$表示边 $e(i,j)$从 N_i 到 N_j 经过的距离,若从 N_i 到 N_j 没有边,则 $dis(i,j)=\infty$,0≤i≤n,0≤j≤n,且 $i\neq j$.$p(j)=(x_i,y_i,z_i)$表示在节点 N_j 时所处的位置,其中,x_i,y_i,z_i 为位置的三维坐标}.

对时空一致性的检验需要检查两个方面.一是行为模型端口分配时给出的最坏执行时间(worst-case execution time,简称 WCET)是否满足,以保证行为附件可以在期望的时间内完成任务;二是针对时空不一致问题的验证,即:在 *t* 时间本应该在 *B* 位置,而经过验证发现在 *t* 时间还处于 *A* 位置,此时的时空不一致问题将会产生严重的后果.

对于第 1 个问题,本文针对状态空间图 *G*,首先判断 *G* 是否为规范的可达图,即,所有的状态空间的路径是否都可以在 *WCET* 时间内完成.本文通过一个深度优先的最坏时间可满足性检查算法判断 *G* 是否为规范的可达图.设定 max 的值为最坏执行时间,算法中设置 *cur_path* 存储当前已访问路径,*abnormal* 存储异常节点(即从 N_0 到当前节点的迁移总时间不满足 *WCET*).该算法的输入是图 *G*,输出为图 *G* 是否为规范可达图:若是规范可达图,则该算法返回值为 true;否则,返回值为 false.最坏时间可满足性检查算法如图 3 所示.

```
算法 1. 最坏时间可满足性检查算法.
max:=WCET; abnormal:=∅; cur_path:={N0}; totalt:=0;
repeat
  ln:=last node in cur_path;   //取当前路径的最后一个节点
  if successor nodes of last node have been visited   //删除已经访问的子节点
    then delete last node of cur_path;
    totalt:=totalt-curr_t;   //删除最后一个节点时总时间减去相关的边
  else
    begin
      bn:=take a unvisited successor node of ln;   //取一个未被访问 ln 的孩子节点 bn
      totalt:=totalt+curr_t   //将当前路径与该孩子节点迁移边的时间加入 totalt
      result:=true;
      if totalt>max then result:=false;   //从源节点到当前节点 bn 之间的执行时间值大于最坏时间 max 时 result 值为 false
      if result=false
        then abnormal:=abnormal∪{bn};   //如果 result 值为 false,将异常节点记下来
      else
        cur_path:=cur_path∪{bn};
    end
until cur_path=∅;
if abnormal=∅ then
  return true;
else return false;
```

Fig.3 Algorithm for checking the worst-time satisfyability

图 3 最坏时间可满足性检查算法

通过算法 1,当返回值为 false 时,说明图 *G* 并不是一个规范的可达图,图中存在执行总时间大于 *totalt* 的路径,此时删除仅属于 *abnormal* 节点所在路径的节点与边,从而可以获得一个规范的可达图 *G*′.接下来对规范可达图 *G*′的时空一致性进行检验(如图 4 所示).*consist_abnormal* 用来存放执行路径中产生时空不一致的异常节

点.该算法的输入为规范的可达图 $G'.restrict[i]$用来表示从源点 N_0 到终点 N_i 的限制时间,通过对比每个节点的限制时间与当前 N_0 到终点 N_i 执行时间,对比目标位置与当前位置间的关系来检查其时空一致性,即,在指定的时间里是否到达了期望位置.当算法的返回值为 true 时,说明图 G'满足时空一致性;当返回值为 false 时,说明有时空不一致的异常点.我们获取异常点所在的位置并修改HAADL的输入/输出值来对HAADL进行一定的容错处理.

算法 2. 时空一致性检查算法.

```
consist_abnormal:=∅; cur_path:={N0};
restrict[i];   //从源点 N0 到终点 Ni 的限制时间
current:=0   //从源点 N0 到当前的时间
repeat
  Nb:=last node in cur_path;   //取当前路径的最后一个节点
  if successor nodes of last node have been visited   //删除已经访问的节点
  then delete last node of cur_path;
    current:=current-curr_t;   //删除最后一个节点时 current 减去相关的边
  else
    begin
      Nk:=take a unvisited successor node of Nb;   //取一个 Nb 未被访问的孩子节点 Nk
      current:=current+curr_t   //将当前路径与该孩子节点迁移边的时间加入 current
      consist:=true;
      if current>restrict[k] then consist:=false;   //从源节点 N0 到当前节点 Nk 的时间超过该节点时间限制
      if consist=false
        then consist_abnormal:=consist_abnormal∪{Nk};   //如果 consist 值为 false,将异常节点记下来
      else
        cur_path:=cur_path∪{Nk};
    end
until cur_path=∅;
if consist_abnormal=∅ then
  return true;
else return false;
```

Fig.4 Algorithm for checking space and time consistency

图 4 时空一致性检查算法

2.6 感应灯实例

下面通过一个感应灯控制系统来使用 HP-TCSP 建模,以说明 HP-TCSP 的描述能力.

感应灯控制器控制感应灯是否点亮.由两个变量控制,分别为是否有声音与是否光线较弱,即:只有当光线弱并且有人经过发出声音时,感应灯才点亮(ON).以感应灯的垂直方向为 z 坐标,当人走到以 z 轴为中心半径为 2m 的范围内,如图 5 所示,当人走在圆柱形范围内,则会触发该位置的上下两个楼层(设每层楼层之间为 6m)的感应灯设置.点亮后,当持续时间超过 2 分钟会自动熄灭(OFF).如果感应灯在 10s 内没有亮,则启用备用灯.备用灯的执行过程与非备用灯相同,不同的是:非备用灯在 10s 内没有点亮,则报告错误 error.

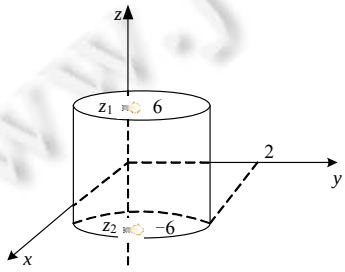

Fig.5 Example of induction lamp

图 5 感应灯实例

设置声音变量为 sv,光亮变量为 lv.当 sv=1 时,即有声音;lv=1 时,即为黑暗.相反的,sv=0 即为无声,lv=0 即为光亮.只有 sv=1&&lv=1 时,感应灯会亮.其余 sv=1&&lv=0,sv=0&&lv=1,sv=0&&lv=0,感应灯不会亮,即不执行任

何操作.时间的单位为秒,时间按照 $t=t+1$ 变化.上下两层灯的坐标分别为 z 轴的 z_1,z_2.该感应灯系统中,包含声音控制系统 *SOCON* 与感光系统 *LICON* 和人体感应系统 *PER*.

- 整个系统进程 *SYSAGENT* 由 *SENSE*,*STANDBY* 与 *ERROR* 进程组成,即:
  - $PV=\{SOCON,LICON,PER,SENSE,STANDBY,ERROR,SYSAGENT\}$;
  - $DV=\{sv,lv,x,y,z,t,z_1,z_2\}$.
- 进程进行通信的信道 $CHANNEL=\{sound,light,cx,cy,cz\}$.

其中,

- $SOCON=sound?sv\rightarrow STOP$;
- $LICON=light?lv\rightarrow STOP$;
- $PER=cx?x\rightarrow STOP||cy?y\rightarrow STOP||cz?z\rightarrow STOP$;
- $ON=turnon\rightarrow STOP$;
- $OFF=turnoff\rightarrow STOP$;
- $SENSE=sound!sv\rightarrow STOP||light!lv\rightarrow STOP||cx!x\rightarrow STOP||cy!y\rightarrow STOP||cz!z\rightarrow STOP;(t:=0)\&\&(t=t+1)\&\&Place(x,y,z)\&t\leqslant 120\&(sv=1)\&(lv=1)>>ON\blacktriangleright(t>120)\vdash OFF;SENSE$.

  其中,$Place(x,y,z)$中的坐标需满足$(x^2+y^2\leqslant 4)\&\&(z-6>z_2)\&\&(z+6<z_1)$;
- $STANDBY=sound!sv\rightarrow STOP||light!lv\rightarrow STOP||cx!x\rightarrow STOP||cy!y\rightarrow STOP||cz!z\rightarrow STOP;(t:=0)\&\&(t=t+1)\&\&Place(x,y,z)\&t\leqslant 120\&(sv=1)\&(lv=1)>>ON\blacktriangleright(t>120)\vdash OFF;STANDBY$.

  其中,$Place(x,y,z)$中坐标需满足$(x^2+y^2\leqslant 4)\&\&(z-6>z_2)\&\&(z+6<z_1)$;
- $ERROR=call\rightarrow STOP$;
- $SYSAGENT=SOCON||LICON||PER||SENSE \overset{d}{\triangleright} STANDBY \overset{d}{\triangleright} ERROR(d=10s)$.

3 扩展后的 AADL 到 HP-TCSP 的转换规则(验证)

为了验证扩展后的混成 AADL 的时空一致性,我们采用形式化的方法进行验证.接下来,先给出 HAADL 到 HP-TCSP 模型转换框架,再定义从 HAADL 到 HP-TCSP 的映射规则.

3.1 HAADL到HP-TCSP的模型转换框架

HAADL 到 HP-TCSP 模型转换框架的核心模块为同构化、语义映射和语法转换.其中,语法转换和语义映射都是构建在所产生的元模型的基础上.对于 HAADL 模型和 HP-TCSP 模型而言,同构化的结果是建立了相互理解的 HAADL 元模型和 HP-TCSP 元模型.语义映射模块针对同构化所产生的 HAADL 元模型和 HP-TCSP 元模型构造语义映射规则,通过执行该转换规则,可以将符合 HAADL 元模型的 HAADL 模型转换为符合 HP-TCSP 元模型的 HP-TCSP 模型(反之亦然).语法转换同样是基于 HAADL 元模型和 HP-TCSP 元模型构造 HAADL 语法规则和 HP-TCSP 语法规则,通过应用该语法规则,就可以将 HAADL 模型和 HAADL 文本进行相互转换、HP-TCSP 模型和 HP-TCSP 文本进行相互转换.值得注意的是:图 6 中的 HAADL 模型和 HP-TCSP 模型相互转换所应用的语义映射相当于是语义映射规则的实例;而 HAADL 语法转换和 HP-TCSP 语法转换分别为 HAADL 语法规则和 HP-TCSP 语法规则的运行实例.

3.2 混成AADL到HP-TCSP的转换规则

扩展的混成 AADL 的行为附件 $ExAction:=\{Vars,States,Transition,Time,Position\}$可以对应到转换过程如下.

(1) 行为附件中的 HAADL 系统变量、时间变量、位置变量中的 *Interger*,*Float*,*Boolean* 和 *String* 的基本类型分别直接映射到 HP-TCSP 离散变量集合中的 *InterType*,*RealType*,*BooleanType* 和 *StringType*;

(2) 将状态集合中的状态对应于 TCSP 中的一个进程;

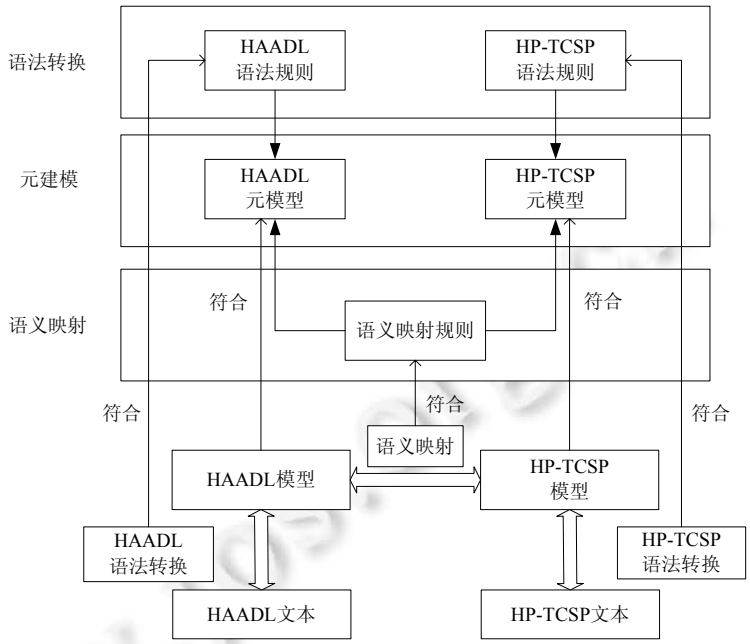

Fig.6 Framework of model transformation from HAADL to HP-TCSP

图 6 HAADL 到 HP-TCSP 模型转换框架

(3) 行为附件中的状态迁移映射:

$Trans\_sequence$:=$SourceState[\langle act_1\rangle\langle guard\rangle]\rightarrow\{DesS_1[\{\langle act_2\rangle\}],DesS_2[\{\langle act_3\rangle\}],...,error]\}$.

guard 对应两种形式:周期执行 on dispatch 与条件执行.接下来,我们对 *guard*、act_1、act_2(迁移序列中的 act_1 与 act_2 形式转换相同)、*error* 与迁移序列分别转换.

(3.1) *guard* 是变迁触发的条件,一般包括定时触发、条件触发与端口发生的输入/输出事件触发.定时触发(time trigger)如 $t=d$ 转换为 $P\overset{d}{\triangleright}Q$,P 为一个不会与任何进程通信的进程,则在 d 时间后执行进程 Q;条件触发(condition trigger)对应于 HP-TCSP 中的条件执行算子 $Con>>P$ 算子 *Con* 中与 $P\blacktriangleright Fin\_Con(Fin\_Con:=Fcon|true)\vdash Q$ 算子 *Fcon* 的条件;端口输入/输出事件触发用 HP-TCSP 中的输入/输出事件表示;

(3.2) act_1,act_2 等中动作可以转换为 HP-TCSP 的条件执行算子中的微分方程变量值的变化.引入 3 个运算符 *vara*?*varb*,*vara*!*varb* 转换为微分方程中变量的赋值变化,多个动作同时进行'|',每个动作转换为只有一个事件的进程,多个动作同时进行表示成进程之间的并发;

(3.3) *error* 对应于 HP-TCSP 中的 *ERROR* 进程;

(3.4) 迁移序列即未在条件 *interrupt* 执行的迁移状态,会执行下一个状态.将其转换为:

$$P\blacktriangleright(interrupt)\vdash Q.$$

其中,P 为迁移序列的前一个状态转换的进程,Q 为迁移序列的后一个状态转换的进程.

3.3 HAADL到HP-TCSP的具体映射规则

根据第 3.2 节的转换策略,我们定义了从 HAADL 到 HP-TCSP 的转换规则,见表 1.

在实现 HAADL 到 HP-TCSP 的转换后,即可得到位置时间迁移系统的状态空间图,再通过第 2.5 节的时空属性分析的算法对其进行时空属性的检查,并对 HAADL 做相应的修改与容错.

Table 1 Mapping rules from HAADL to HP-TCSP

表 1 HAADL 到 HP-TCSP 映射规则

Name	Description
Integer2IntegerType	Integer2IntegerType Match *HAADL.IntegerType* to *HP-TCSP.Integer*
Float2RealType	Float2RealType Match *HAADL.floatType* to *HP-TCSP.Real*
Boolean2BooleanType	Boolean2BooleanType Match *HAADL.BooleanType* to *HP-TCSP.Boolean*
String2StringType	String2StringType Match *HAADL.StringType* to *HP-TCSP.String*
state2process	state2process Match *HAADL.state* to *HP-TCSP.process*
timetrigger2overtime	timetrigger2overtime Match *HAADL.timetrigger* to *HP-TCSP*. $P \overset{d}{\triangleright} Q$
conditiontrigger2ConFcon	conditiontrigger2ConFcon Match *HAADL.conditiontrigger* to *HP-TCSP.Con* and *HP-TCSP.Fcon*
Inoutput2Inoutput	Inoutput2Inoutput Match *HAADL.input* and *HAADL.output* to *HP-TCSP.input* and *HP-TCSP.output*
act2assignment	act2assignment Match *HAADL.act* to *HP-TCSP.assignment*
assignment2assignment	assignment2assignment Match *HAADL.assignment* to *HP-TCSP.assignment*
sameact2and	sameact2and Match *HAADL.*| to *HP-TCSP*||
interrupt2interrupt	interrupt2interrupt Match *HAADL.interrupt* to *HP-TCSP.condition interrupt*

3.4 转换实例

接下来将第 1.4 节带警报的水箱控制系统扩展的行为附件部分转换成相应的 HP-TCSP 模型.

首先,将 *variables*,*times*,*position* 中的 *Integer* 与 *float* 分别转换为 HP-TCSP 离散变量集合中的 *InterType* 与 *RealType* 类型.状态 *open*,*close*,*alarm* 转换为进程变量 *OPEN*,*CLOSE*,*ALARM* 以及其他条件执行与条件中断等.具体转换见表 2.

Table 2 Mapping rules from HAADL to HP-TCSP for water tank control system

表 2 水箱控制系统 HAADL 到 HP-TCSP 映射规则

HAADL	HP-TCSP	HAADL	HP-TCSP
Integer top	*Integer top*	*open*	*OPEN*
Integer bottom	*Integer bottom*	*close*	*CLOSE*
float t	*Real t*	*alarm*	*ALARM*
float t'	*Real t'*	*y=y+1*	*y?(y+1)*
float wait	*Real wait*	*y≠bottom*	*(y≠bottom&wait<t)>>OPEN*
Integer y	*Integer y*	*y≠top*	*(y≠top&wait<t')>>CLOSE*
conditioninterrupt	*OPEN▶(y=top)&&(wait>t')⊢ALARM*	*conditioninterrupt*	*CLOSE▶(y=bottom&&wait>t)⊢ALARM*

转换后,HP-TCSP 如下:

- 离散变量集合 *DV*:={*top*,*bottom*,*t*,*t'*,*wait*,*y*};
- 进程变量集合 *PV*:={*OPEN*,*CLOSE*,*ALARM*}.

其中,

WATER=OPEN▶(y=top)&&(wait>t')⊢ALARM||CLOSE▶(y=bottom&&wait>t)⊢ALARM;

OPEN=y=(y+k)→STOP;

CLOSE=y=(y−k)→STOP;

ALARM=alarm→STOP.

4 实例分析

飞机避撞系统是典型的 CPS 应用实例,对飞机避撞系统的安全性研究也是研究热点之一.2000 年~2014 年间,美国空军的 F-16 战斗机因可控飞行撞地事故而损失的飞行员人数占到了所有非发动机失效相关事故的 75%.美国空军研究实验室(AFRL)研究发现,飞机发生与地面或其他飞机相撞事故的起因包括加速度致飞行员意识丧失(G-LOC)、空间迷向、飞行员过于盯住目标、注意力分散和任务饱和等.因此,保证飞机避撞系统的安全性十分必要.下面我们通过一个飞机避撞系统的实例,给出 AADL 的行为附件的描述并且利用转换规则转换为形式化的 HP-TCSP 验证其时空的一致性,从而验证避撞系统的安全性.

现研究一个竖直平面的飞机避撞系统,假设飞机处于同一竖直平面位置.如图 7 所示:假设两架飞机 p_1 与 p_2

的飞行速度分别为 v_1 与 v_2.假设飞机在 t 时刻于某点 k 相撞,则 $x_1+t\times v_1=x_2+t\times v_2$.在存在碰撞危险的距离处,飞机可采取垂直解脱(爬升或下降)或水平解脱(左飞或右飞)进行避撞.

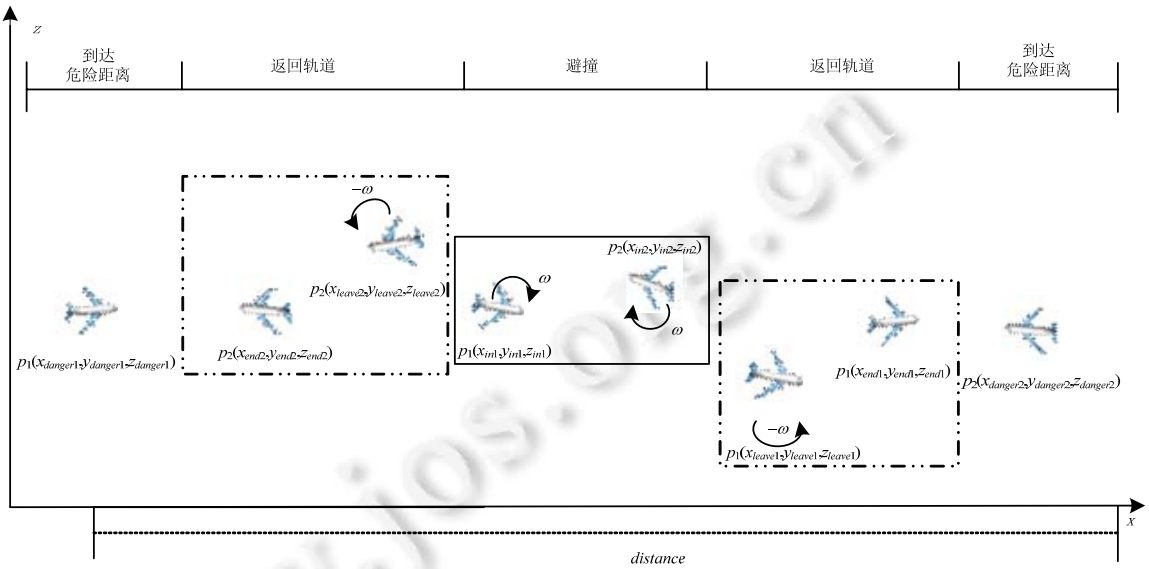

Fig.7 Aircraft anticollision

图 7 飞机避撞图

图 7 中两飞机避撞过程具体场景如下.

(1) 两架飞机正常航行,即将到达危险距离;

(2) 到达危险距离 *distance* 时,p_1 飞机位置为$(x_{danger1},y_{danger1},z_{danger1})$与 p_2 飞机位置为$(x_{danger2},y_{danger2},z_{danger2})$.在响应时间 *danger_wait* 时需要做出反应,即:在相距距离 $distance=x_2-x_1=danger_wait\times(v_1+v_2)$时,飞机要采取避撞措施.避撞措施为指定一定的方向爬升/下降/左/右/平飞.旋转飞机以碰撞中心为圆心,分别在以 r_1 与 r_2 的半径同心圆上,以ω的角度调节方向,平滑进入各自的避撞轨道,进入点为$(x_{in1},y_{in1},z_{in1})$与$(x_{in2},y_{in2},z_{in2})$,且从处于危险距离处到到达避撞轨道需 *prepare*(*pr*)时间完成,否则将启用自动避撞系统或容错措施;

(3) 两飞机以相同大小的角速度ω在同心圆上旋转飞行,且该过程中,线速度(角速度与半径的乘积)的大小与原飞机的运行速度大小相同,即 $(r_1\omega)^2=v_1^2\wedge(r_2\omega)^2=v_2^2$,且从到达避撞轨道到离开避撞轨道时间需为 *complete*(*ct*),在等待时间 *in_wait* 时需要做出反应,否则启用自动避撞系统或容错措施;

(4) 旋转飞行结束后,两飞机以$-\omega$的角速度在点$(x_{leave1},y_{leave1},z_{leave1})$与$(x_{leave2},y_{leave2},z_{leave2})$处离开避撞轨道,在点$(x_{end1},y_{end1},z_{end1})$与$(x_{end2},y_{end2},z_{end2})$处到达原航行轨道,此时角速度$\omega$为 0.该驶出避撞轨道到达正常轨道执行时间为 *back*(*bk*),在等待时间 *end_wait* 时需要做出反应,否则将启用自动避撞系统或者容错措施;

(5) 恢复正常航行.

整个过程的最坏执行时间为 *worst_execute*,即:该避障的过程必须在 *worst_execute* 前完成,以保证飞机的安全性.该过程的 4 个关键处:到达危险距离、开始进入避撞轨道、开始离开避撞轨道、结束离开避撞轨道正常航行.

接下来,我们用扩展的 AADL 的行为附件描述该避撞过程.

thread implementation example.impl

annex behavior_specification{**

variables

v_1:*Base_Types*::*float*; v_2:*Base_Types* :: *float*;

r_1:*Base_Types*::*float*; r_2:*Base_Types*::*float*;

ω:*Base_Types*::*float*; *distance*:*Base_Types*::*float*;

direction:*Base_Types*:*String*;

times

danger_wait:*Base_Types*::*float*; *in_wait*:*Base_Types*::*float*;

end_wait:*Base_Types*::*float*; *pr*:*Base_Types*::*float*;

ct:*Base_Types*::*float*; *bk*:*Base_Types*::*float*;

wait:*Base_Types*::*float*; *compute*:*Base_Types*::*float*;

position

$x_{danger1}$:*Base_Types*::*float*; $x_{danger2}$:*Base_Types*::*float*;

$y_{danger1}$:*Base_Types*::*float*; $y_{danger2}$:*Base_Types*::*float*;

$z_{danger1}$:*Base_Types*::*float*; $z_{danger2}$:*Base_Types*::*float*;

x_{in1}:*Base_Types*::*float*; x_{in2}:*Base_Types*::*float*;

y_{in1}:*Base_Types*::*float*; y_{in2}:*Base_Types*::*float*;

z_{in1}:*Base_Types*::*float*; z_{in2}:*Base_Types*::*float*;

x_{leave1}:*Base_Types*::*float*; x_{leave2}:*Base_Types*::*float*;

y_{leave1}:*Base_Types*::*float*; y_{leave2}:*Base_Types*::*float*;

z_{leave1}:*Base_Types*::*float*; z_{leave2}:*Base_Types*::*float*;

x_{end1}:*Base_Types*::*float*; x_{end2}:*Base_Types*::*float*;

y_{end1}:*Base_Types*::*float*; y_{end2}:*Base_Types*::*float*;

z_{end1}:*Base_Types*::*float*; z_{end2}:*Base_Types*::*float*;

states

s_1:*initial state*;

s_2:*state*;

s_3:*state*;

s_4:*complete state*;

error:*state*;

transitions

*T*_0:

$s_1[\langle p_1v?v_1|p_2v?v_2\rangle\langle(distance=danger\_wait*(p_1v!v_1+p_2v!v_2))\wedge(wait=danger\_wait)\wedge(compute=pr)\rangle]\rightarrow$
$\{s_2[\{(x_{danger1},y_{danger1},z_{danger1})!|(x_{danger2},y_{danger2},z_{danger2})!\rangle\}],error]\}$

*T*_1:

$s_2[\langle direction?|(x_{in1},y_{in1},z_{in1})!|(x_{in2},y_{in2},z_{in2})!|p_1v?v_1|p_2v?v_2\rangle\langle(r_1\omega)^2=v_1^2\wedge(r_2\omega)^2=v_2^2\wedge(wait=in\_wait)\wedge$
$(compute=ct)\rangle]\rightarrow\{s_3[\{\langle p_1w:=\omega|p_2w:=\omega\rangle\}],error]\}$

*T*_2:

$s_3[\langle direction?|(x_{leave1},y_{leave1},z_{leave1})!|(x_{leave2},y_{leave2},z_{leave2})!|p_1v?v_1|p_2v?v_2\rangle\langle(r_1\times-\omega)^2=v_1^2\wedge(r_2\times-\omega)^2=v_2^2\wedge$
$(wait=end\_wait)\wedge(compute=bk)\wedge(\omega\neq0)\rangle]\rightarrow\{s_4[\{\langle\omega=0|(x_{end1},y_{end1},z_{end1})!|(x_{end2},y_{end2},z_{end2})!\rangle\}],error]\}$

**};

end *example.impl*;

接下来,我们将 HAADL 的行为附件通过定义的转换规则转为 HP-TCSP.

其中的状态 s_1~s_4,*error* 对应于进程 *NORMAL*,*ENTRY*,*ROTATION*,*EXIT*,*ERROR*,且 AADL 行为附件中的

variables 对应于 HP-TCSP 中的离散变量集合,其中涉及的执行与位置的变化对应于条件执行算子的赋值变化,条件对应于算子中的 *Pcon* 与 *Fin_Con*(见表 3).

Table 3 Mapping rules from HAADL to HP-TCSP for aircraft collision avoidance

表 3 飞机避撞系统 HAADL 到 HP-TCSP 映射规则

HAADL	HP-TCSP	HAADL	HP-TCSP
float v_1	*Real* v_1	s_1	*NORMAL*
float v_2	*Real* v_2	s_2	*ENTRY*
float r_1	*Real* r_1	s_3	*ROTATION*
float r_2	*Real* r_2	s_4	*EXIT*
float ω	*Real* ω	*error*	*ERROR*
float distance	*Real distance*	$p_1v?v_1$	$p_1v?v_1\rightarrow STOP$
String direction	*String direction*	$p_2v?v_2$	$p_2v?v_2\rightarrow STOP$
float danger_wait	*Real danger_wait*	*direction*?	$direction?\rightarrow STOP$
float in_wait	*Real in_wait*	$(x_{in1},y_{in1},z_{in1})!$	$(x_{in1},y_{in1},z_{in1})!\rightarrow STOP$
float end_wait	*Real end_wait*	$(x_{in2},y_{in2},z_{in2})!$	$(x_{in2},y_{in2},z_{in2})!\rightarrow STOP$
float pr	*Real pr*	$(x_{danger2},y_{danger2},z_{danger2})!$	$(x_{danger2},y_{danger2},z_{danger2})!\rightarrow STOP$
float ct	*Real ct*	$(x_{danger1},y_{danger1},z_{danger1})!$	$(x_{danger1},y_{danger1},z_{danger1})!\rightarrow STOP$
float wait	*Real wait*	$(x_{leave1},y_{leave1},z_{leave1})!$	$(x_{leave1},y_{leave1},z_{leave1})!\rightarrow STOP$
float compute	*Real compute*	$(x_{leave2},y_{leave2},z_{leave2})!$	$(x_{leave2},y_{leave2},z_{leave2})!\rightarrow STOP$
float $x_{danger1}$	*Real* $x_{danger1}$	$p_1w?$	$p_1w?\rightarrow STOP$
float $y_{danger1}$	*Real* $y_{danger1}$	$p_2w?$	$p_2w?\rightarrow STOP$
float $z_{danger1}$	*Real* $z_{danger1}$	$\omega=0$	$\omega=0\rightarrow STOP$
float $x_{danger2}$	*Real* $x_{danger2}$	$(x_{end1},y_{end1},z_{end1})!$	$(x_{end1},y_{end1},z_{end1})!\rightarrow STOP$
float $y_{danger2}$	*Real* $y_{danger2}$	$(x_{end2},y_{end2},z_{end2})!$	$(x_{end2},y_{end2},z_{end2})!\rightarrow STOP$
float $z_{danger2}$	*Real* $z_{danger2}$	$\wedge$	&
float x_{in1}	*Real* x_{in1}	$s_1\rightarrow s_2$ 的迁移序列	$ENTRY\blacktriangleright(distance\neq danger\_wait*(p_1v!v_1+p_2v!v_2)\vee wait\neq danger\_wait\vee compute\neq pr)\vdash ERROR$
float y_{in1}	*Real* y_{in1}	$s_2\rightarrow s_3$ 的迁移序列	$ROTATION\blacktriangleright((r_1\omega)^2\neq v_1^2\vee(r_2\omega)^2\neq v_2^2\vee wait\neq in\_wait\vee compute\neq ct)\vdash ERROR$
float z_{in1}	*Real* z_{in1}	$s_3\rightarrow s_4$ 的迁移序列	$EXIT\blacktriangleright((r_1-\omega)^2\neq v_1^2\vee(r_2-\omega)^2\neq v_2^2\vee wait\neq end\_wait\vee compute\neq bk)\vdash ERROR$
float x_{in2}	*Real* x_{in2}	*float* z_{leave2}	*Real* z_{leave2}
float y_{in2}	*Real* y_{in2}	*float* x_{end1}	*Real* x_{end1}
float z_{in2}	*Real* z_{in2}	*float* y_{end1}	*Real* y_{end1}
float x_{leave1}	*Real* x_{leave1}	*float* z_{end1}	*Real* z_{end1}
float y_{leave1}	*Real* y_{leave1}	*float* x_{end2}	*Real* x_{end2}
float z_{leave1}	*Real* z_{leave1}	*float* y_{end2}	*Real* y_{end2}
float x_{leave2}	*Real* x_{leave2}	*float* z_{end2}	*Real* z_{end2}
float y_{leave2}	*Real* y_{leave2}		

由此得到的 HP-TCSP 的描述如下.

- 进程变量集合 $PV:=\{NORMAL,ENTRY,ROTATION,EXIT,ERROR\}$;
- 离散变量集合 $DV:=\{v_1,v_2,r_1,r_2,\omega,distance,direction,danger\_wait,in\_wait,end\_wait,pr,ct,wait,compute,x_{danger1},y_{danger1},z_{danger1},x_{danger2},y_{danger2},z_{danger2},x_{in1},y_{in1},z_{in1},x_{in2},y_{in2},z_{in2},x_{leave1},y_{leave1},z_{leave1},x_{leave2},y_{leave2},z_{leave2},x_{end1},y_{end1},z_{end1},x_{end2},y_{end2},z_{end2}\}$.

其中,

- $NORMAL=p_1v?v_1\rightarrow STOP||p_2v?v_2\rightarrow STOP||Place[distance=danger\_wait*(p_1v!v_1+p_2v!v_2)]\&(wait=danger\_wait)\&(compute=pr)>>ENTRY\blacktriangleright(distance\neq danger\_wait*(p_1v!v_1+p_2v!v_2)\vee wait\neq danger\_wait\vee compute\neq pr)\vdash ERROR$;
- $ENTRY=direction?\rightarrow STOP||(x_{in1},y_{in1},z_{in1})!\rightarrow STOP||(x_{in2},y_{in2},z_{in2})!\rightarrow STOP||p_1v?v_1\rightarrow STOP||p_2v?v_2\rightarrow STOP||(x_{danger1},y_{danger1},z_{danger1})!\rightarrow STOP||(x_{danger2},y_{danger2},z_{danger2})!\rightarrow STOP||((r_1\omega)^2=v_1^2)\&\&((r_2\omega)^2=v_2^2)\&\&(wait=$

$in\_wait)\&(compute=ct)>>ROTATION\blacktriangleright((r_1\omega)^2\neq v_1^2\vee(r_2\omega)^2\neq v_2^2\vee wait\neq in\_wait\vee compute\neq ct)\vdash ERROR$;

- $ROTATION=direction?\rightarrow STOP||(x_{leave1},y_{leave1},z_{leave1})!\rightarrow STOP||(x_{leave2},y_{leave2},z_{leave2})!\rightarrow STOP||p_1v?v_1\rightarrow STOP||p_2v?v_2\rightarrow STOP||p_1w?\rightarrow STOP||p_2w?\rightarrow STOP||((r_1\times-\omega)^2=v_1^2)\,\&\,\&((r_2\times-\omega)^2=v_2^2)\,\&\&(wait=end\_wait)\&(compute=bk)\&\omega\neq0>>EXIT\blacktriangleright((r_1\times-\omega)^2\neq v_1^2\vee(r_2\times-\omega)^2\neq v_2^2\vee wait\neq end\_wait\vee compute\neq bk)\vdash ERROR$;
- $EXIT=(\omega=0\rightarrow STOP||(x_{end1},y_{end1},z_{end1})!\rightarrow STOP||(x_{end2},y_{end2},z_{end2})!\rightarrow STOP)>>EXIT\blacktriangleright(\omega\neq0)\vdash ERROR$;
- *ERROR* 定义为一系列容错措施.

HP-TCSP 的状态空间对应于一个位置时间迁移系统,针对于不同的指令,其所需的等待时间与执行时间不同.表 4 展示了事件等待时间与执行时间.

Table 4 Time for aircraft collision avoidance

表 4 飞机避撞的时间

事件	等待时间	执行时间	所处位置(p_1 飞机/p_2 飞机)
进入轨道(*et*)	2	5	$(x_{danger1},y_{danger1},z_{danger1})/(x_{danger2},y_{danger2},z_{danger2})$
左转(*lf*)	3	8	$(x_{in1},y_{in1},z_{in1})/(x_{in2},y_{in2},z_{in2})$
右转(*rt*)	3	8	$(x_{in1},y_{in1},z_{in1})/(x_{in2},y_{in2},z_{in2})$
爬升(*cb*)	2	8	$(x_{in1},y_{in1},z_{in1})/(x_{in2},y_{in2},z_{in2})$
下降(*dsd*)	2	6	$(x_{in1},y_{in1},z_{in1})/(x_{in2},y_{in2},z_{in2})$
平飞(*mt*)	1	5	$(x_{in1},y_{in1},z_{in1})/(x_{in2},y_{in2},z_{in2})$
离开(*lf*/*rt*/*cb*/*dsd*/*mt* 与以上避撞时的方向相反)	3	8	$(x_{leave1},y_{leave1},z_{leave1})/(x_{end2},y_{end2},z_{end2})$

其状态空间图 *G* 如图 8 所示:首先,先通过算法 1 获取一个规范的可达图,若本次最坏执行时间 *worst_execute* 为 36.我们对比每一条路径的执行时间.通过最坏时间可满足性算法,得到返回值为 false,获取异常节点集合 $abnormal=\{N_{12},N_{13}\}$.由于$\{N_0N_1N_2N_7N_{12}\}$与$\{N_0N_1N_3N_8N_{13}\}$的执行时间 *totalt* 值为 38,则该两条路径的总执行时间不满足最坏执行时间的要求,说明在当前状态下,采取左转右转方式避撞会有碰撞危险,因此删除仅属于这两条路径的节点与迁移边,然后得到状态空间图 *G*′.此外,修改 HAADL 中的 *direction* 的输入为爬升、下降或平飞.

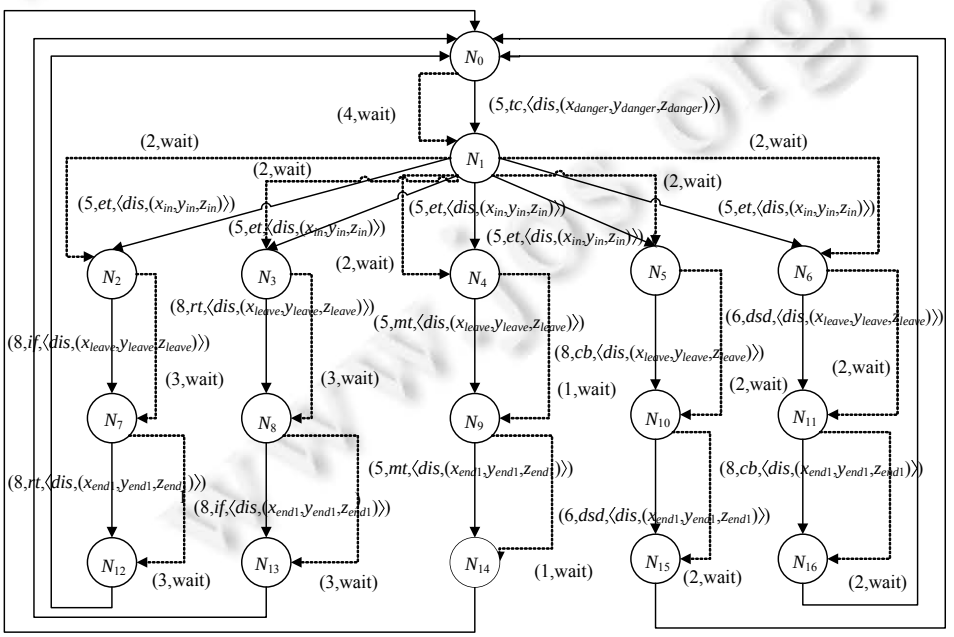

Fig.8 Position-time transition system *G* of aircraft collision avoidance

图 8 飞机避撞的位置-时间迁移系统 *G*

接下来,针对图 G'进行时空一致性的算法验证(如图 9 所示).

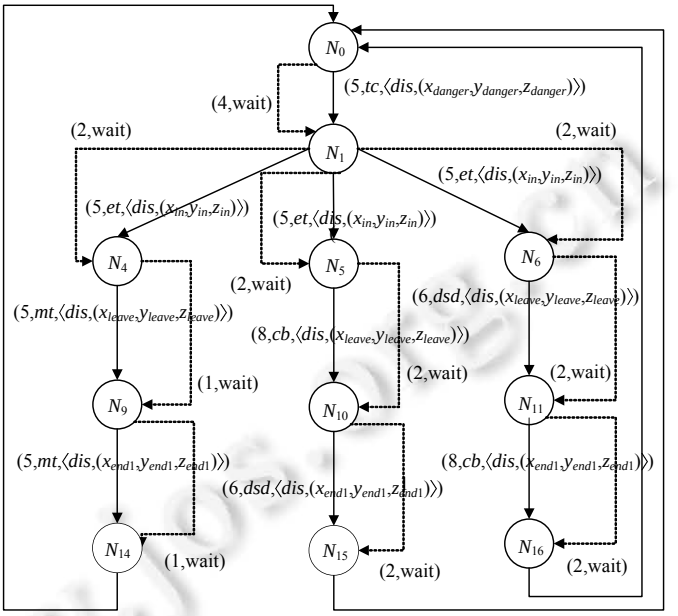

Fig.9 Normal reachable graph G'

图 9 规范的可达图 G'

各个节点的限制 *restrict*[ ]={0,9,16,16,16,16,16,28,29,22,**27**,24,40,40,**27**,34,34},表示从源节点 N_0 到当前节点 N_i 的时间限制.该限制数组可以从以往的知识库中获取.通过时空一致性检查算法,返回值为 false,存在时间不一致的问题,且 *consist_abnormal* 值为{N_{10},N_{14}},在 N_{10} 处的时空不一致是本应在时间 26 到达(x_{leave},y_{leave},z_{leave})处,当前避撞飞机预计在时间 27 到达,存在时空不一致现象,此时提高 HAADL 中输入的飞机速度,以便在规定时间 26 到达(x_{leave},y_{leave},z_{leave})处,或启用自动避撞系统进行容错处理,并输出一定的反馈信号.在 N_{14} 处的时空不一致是本应 28 时间到达(x_{end},y_{end},z_{end})处,但预计飞机在时间 27 到达,此时减小 HAADL 中输入的飞机速度,或启用自动避撞系统进行容错处理,并输出一定的反馈信号,例如协调两架飞机、指示另一架飞机加速等措施.

5 相关工作比较

CPS 系统是一个综合计算、网络和物理环境的多维复杂系统,对 CPS 的建模应该考虑系统的时空一致性.一方面,近年来对 AADL 扩展以及利用 AADL 对 CPS 进行建模与验证,国内外已有相关研究.文献[6]针对 CPS 的连续行为以及网络组件和物理组件之间的交互进行建模,提出了构造新的 AADL 子语言 AADL+的方法.文献[7]通过扩展 AADL 的标准和附录,实现对计算实体、物理实体和交互实体的统一描述.文献[8]将 AADL 和 Simulink/Stateflow 的结合起来,提供了统一的图形化协同建模形式,从软件、硬件和物理这 3 个角度支持 CPS 的设计.这部分工作主要侧重于研究 CPS 的计算、网络和物理环境之间的交互与融合方面.文献[15]在 AADL 行为附件上进行层次化扩展,从而使行为附件中具有表达层次自动机的机制.文献[16]等将 AADL 与 Z 结合,对嵌入式软件可靠性进行建模与评估,并在此基础上扩展了概率进行可靠性评估.文献[17]扩展了进程代数 CCS,提出 HPCCS 进行建模并扩展 AADL 的行为附件描述随机动作.文献[18]针对混成 AADL 在不确定环境下的建模与分析问题,提出了面向不确定环境的混成 AADL 设计与量化分析方法,从而支持混成 AADL 的不确定性建模以及对混成 AADL 模型的定量分析.该部分研究主要在 AADL 上进行扩展,从而进行可靠性分析,但其更注重系统的随机性,对时空的一致性考虑较少.文献[19]基于带有时间约束的 AADL 行为模型时序验证问题,提出了基于节点转换规则的 AADL 行为模型分解规则,从而支持实时系统对隐式时间约束和显式时间约束的验

证.这部分工作主要考虑时间的约束,而对位置因素考虑的较少.文献[20]中,用 AADL 对中国列车控制系统(CTCS-3)进行行为建模与验证.其使用核心 AADL 结构对系统的结构进行建模,利用 BLESS 附件对控制系统的离散行为进行了建模和验证,利用混合附件对列车的连续行为和网络-物理交互进行建模,并使用定理证明的方法进行验证.该方法具有一定的针对性,主要通过定理证明的方法对列车控制系统的离散行为、连续行为和网络-物理交互进行建模;本文方法主要是从模型检测的角度对 CPS 验证,两种方法具有一定的互补性.不同方法研究 CPS 的侧重点各不相同,与以上工作不同,本文更加注重 CPS 的非功能属性,并且从 AADL 进行建模入手,通过模型转换,将其转换成形式化模型进行模型检测.该方法是对已有方法的有效补充,能够在实际应用上发挥一定的效用.

另一方面,关于 CPS 的时间或空间性质建模与验证问题,国内外已有相关研究.文献[21]提出了基于时间状态迁移矩阵的形式化建模方法,在迁移矩阵形式语义基础上,引入时钟函数与时间约束,从而实现在软件设计层面逻辑功能与时间性能的一体化建模与验证.文献[22]通过定义一系列的规则,将领域环境模型组合到运行时验证过程中.该方法研究环境的变化对系统参数的实时影响,但未将系统本身的空间随着时间的变化考虑在内.文献[23]把 CPS 系统分为物理实体、计算实体和控制实体,并给出基于动态行为建模的 UML 建模方法对 CPS 系统计算实体进行建模,从而完整地把系统计算实体的逻辑关系和时序关系描述出来.文献[24,25]在时间条件约束下,对 CPS 系统各部分之间如何安全交互进行建模研究.这些传统的 CPS 建模与验证方法大多局限于时间域内的分析,对于时间与空间统一变化对 CPS 的影响考虑的较少,因此可能会存在一些 CPS 时空一致的安全性问题.

文献[26]将 CPS 时空状态转移融合成一个状态转移实时时空事件,并在时间 Petri 网基础上引入空间标签,建立时空 Petri 网模型,利用时空 Petri 网对物理实体状态转移过程进行分析.文献[27]针对运输信息物理融合系统(T-CPS)缺乏时空建模与分析问题,提出了有色时空 Petri 网(CSTPN),并用 CSTPN 交通路口协调控制系统的开发、交通仿真分析以及基于 T-CPS 的 CSTPN 的实现.这部分方法更加注重于 CPS 的时空性质的建模方面,本文是已有工作的延续工作,本文中加入了时空一致性的验证,从而能够保证 CPS 的时空安全性.

与以上研究工作相比,本文重点研究非功能属性中的时间与空间属性,即时空不一致对系统安全性的影响,重点关注 CPS 时空一致性的建模与验证.本文方法是对已有工作的有效补充.本文首先在半形式化的 AADL 上作时空性质的扩展;其次,在形式化 TCSP 上扩展空间属性;再次,将扩展的 HAADL 转化成扩展后的 HP-TCSP 并进行时空一致性的验证,从而保证 CPS 的时空安全性.

6 结束语

CPS 的安全性问题受到人们的广泛关注.针对验证 CPS 时空不一致的安全性问题,本文提出了面向 CPS 时空性质验证的混成 AADL 建模与模型转换方法.首先,在 AADL 的行为附件中扩展时间与空间因素;其次,在 TCSP 中引入微分方程以及位置描述,提出了 HP-TCSP,能够验证 CPS 的时空性质;再次,通过模型转换,将 HAADL 转换为 HP-TCSP,从而可以将 HAADL 描述的 CPS 模型在 HP-TCSP 中进行时空一致性验证;最后,通过一个飞机避撞系统实例验证该方法的有效性,从而保障 CPS 的时空安全性.

但是在转换的过程中,大多是根据转换规则进行手动转换,接下来将会完成一个自动转换系统,实现 HAADL 到 HP-TCSP 的自动转换.

本文主要研究关于 AADL 行为附件的扩展,对于 AADL 组件的物理附件的扩展并没有过多讨论,关于 AADL 物理附件的扩展模块将会在后续的研究工作中继续展开研究.

CPS 系统场景复杂,对于时间不一致的容错处理部分,我们也将继续展开研究.此外,时间与空间的变化也伴随着空间拓扑关系的变化,如何解决物理空间与时间变化约束的 CPS 安全问题,将会是下一步的研究重点.

CPS 运行环境复杂,且在目前发展环境下应用场景较为广泛,例如自动驾驶等,这些具体复杂的场景中不仅有整体的时空一致性,其内部也包含了一定的时空一致性.因此,如何实现时空一致性的整体与部分的组合形式化验证,也是接下来的一个研究方面.

References:

[1] Zheng X, Julien C, Kim M, Khurshid S. Perceptions on the State of the art in verification and validation in cyber-physical systems. IEEE Systems Journal, 2017,11(4):2614−2627.

[2] Banks VA, Plant KL, Stanton NA. Driver error or designer error: Using the perceptual cycle model to explore the circumstances surrounding the fatal tesla crash on 7th May 2016. Safety Science, 2018,108:278−285.

[3] Feiler PH, Lewis B, Vestal S, Colbert E. An overview of the sae architecture analysis & design language (AADL) standard: A basis for model-based architecture-driven embedded systems engineering. Architecture Description Languages, 2005,176:3−15.

[4] Zhang L. Specifying and modeling cloud cyber physical systems based on AADL. In: Proc. of the 17th Int'l Symp. on Distributed Computing and Appkications for Business Engineering and Science. Wuxi: Institute of Electrical and Electronics Engineers Inc, 2018. 26−29.

[5] Goncalves FS, Pereira D, Tovar E, Becker LB. Formal verification of AADL models using UPPAAL. In: Proc. of the VII Brazilian Symp. on Computing Systems Engineering. Curitiba: IEEE Computer Society, 2017. 117−124.

[6] Liu J, Li T, Ding Z. AADL+: A simulation-based methodology for cyber-physical systems. Frontiers of Computer Science, 2019, 13(3):516−538.

[7] He R, Zhang W, Wu L. AADL-Based reliability modeling method of cyber-physical systems. In: Proc. of the WSSE2019. Wuhan: Association for Computing Machinery, 2019. 47−58.

[8] Zhan H, Lin Q, Wang S. Unified graphical co-modelling of cyber-physical systems using AADL and simulink/stateflow. In: Ribeiro P, Sampaion A, eds. Proc. of the UTP 2019. Porto: Springer-Verlag, 2019. 109−129.

[9] Dong YW, Wang GR, Zhang F, Gao L. AADL model reliability analysis and evaluation tool. Ruan Jian Xue Bao/Journal of Software, 2011,22(06):1252−1266 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4014.htm [doi: 10.3724/SP.J.1001.2011.04014]

[10] Hoare CAR. Communicating sequential processes. Communications of the ACM, 1978,21(8):666−677

[11] George M, Reed, Roscoe AW. A timed model for communicating sequential processes. In: Proc. of the 13th Int'l Colloquium on Automata, Languages and Programming (ICALP'86). Rennes, 1986.

[12] Yang ZB, Hu K, Zhao YW, Ma DF, Jean-Paul BODEVEIX. Validation of AADL model based on time abstract state machine. Ruan Jian Xue Bao/Journal of Software, 2015,26(2):202−222 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4776.htm [doi: 10.13328/j.cnki.jos.004776]

[13] Schneider S. An operational semantics for timed CPS. Information and Computation, 1995,116(2):193−213

[14] Zhu Y, Huang ZQ, Cao ZN, Zhou H, Liu YP. A method of generating software architecture model based on formal specifications. Ruan Jian Xue Bao/Journal of Software, 2010,21(11):2738−2751 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3701.htm [doi: 10.3724/SP.J.1001.2010.03701]

[15] Xu JM, Yang ZB, Huang ZQ, Xie J, Zhou Y. Functional behavior modeling extension of architecture analysis & design language (AADL). Computer Science and exploration, 2019,13(10):1638−1653 (in Chinese with English abstract).

[16] Li M, Zhuang Y, Hu TW. An embedded software reliability modeling and evaluation method combining AADL and Z. Computer Science, 2019,46(08):217−223 (in Chinese with English abstract).

[17] Cao XY, Cao ZN, Bu XC. Hybrid AADL modeling and model transformation for CPS. Computer Technology and Development, 2019,29(10):35−40 (in Chinese with English abstract).

[18] Bao YX, Chen MS, Zhu Q, Wei TQ, Frederic M, Zhou TL. Quantitative performance evaluation of uncertainty-aware hybrid AADL designs using statistical model checking. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, 2017, 36(12):1989−2002.

[19] Liu X, Xie HM. AADL behavior model time consistency verification method. Computer Technology and Development, 2017, 27(7):1−5 (in Chinese with English abstract).

[20] Ahmad E, Dong YW, Larson B, Lü JD, Tang T, Zhan NJ. Behavior modeling and verification of movement authority scenario of Chinese train control system using AADL. Science China (Information Sciences), 2015,58(11):125−144.

[21] Hou G. Modeling, verification and analysis of cyber physical system software [Ph.D. Thesis]. Dalian: Dalian University of Technology, 2018 (in Chinese with English abstract).

[22] Luo CX, Wang R, Guan Y, Li XJ, Shi ZP, Song XY. CPS integrated modeling method for real-time data. Ruan Jian Xue Bao/ Journal of Software, 2019,30(7):1966−1979 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/30/1966.htm

[23] Ye F, Xia Y, Shen ZX, Zhu YC. CPS modeling method based on dynamic behavior modeling. Journal of System Simulation, 2016, 28(05):1003−1008,1016 (in Chinese with English abstract).

[24] Zhang ZH, Zhu Y, Xiao FX. Modelling and analysis of real-time and reliability for WSN-based CPS. Int'l Journal of Internet Protocol Technology (IJIPT), 2019,12(2).

[25] Su Q, Wang T, Chen TM, Chen RR. CPS security modeling and validation based on time automaton. Information Security Research, 2017,3(7):601−609 (in Chinese with English abstract).

[26] Zhang J, Wang L, Fan HB. Modeling and analysis of CPS physical entity spatio-temporal state. Computer Engineering and Applications, 2018,54(14):41−44 (in Chinese with English abstract).

[27] Zhao HZ, Cheng S, Yue H. Using CSTPNs to model traffic control CPS. IET Software, 2017,11(3):116−125.

附中文参考文献:

[9] 董云卫,王广仁,张凡,高磊.AADL 模型可靠性分析评估工具.软件学报,2011,22(6):1252−1266. http://www.jos.org.cn/1000-9825/4014.htm [doi: 10.3724/SP.J.1001.2011.04014]

[12] 杨志斌,胡凯,赵永望,马殿富,Jean-Paul BODEVEIX.基于时间抽象状态机的 AADL 模型验证.软件学报,2015,26(2):202−222. http://www.jos.org.cn/1000-9825/4776.htm [doi: 10.13328/j.cnki.jos.004776]

[14] 祝义,黄志球,曹子宁,周航,刘亚萍.一种基于形式化规约生成软件体系结构模型的方法.软件学报,2010,21(11):2738−2751. http://www.jos.org.cn/1000-9825/3701.htm [doi: 10.3724/SP.J.1001.2010.03701]

[15] 许金淼,杨志斌,黄志球,谢健,周勇.系统架构描述语言 AADL 的功能行为建模扩展.计算机科学与探索,2019,13(10):1638−1653.

[16] 李蜜,庄毅,胡镡文.一种结合 AADL 与 Z 的嵌入式软件可靠性建模与评估方法.计算机科学,2019,46(8):217−223.

[17] 曹雪岳,曹子宁,卜星晨.面向 CPS 的混成 AADL 建模与模型转换.计算机技术与发展,2019,29(10):35−40.

[19] 刘骁,谢红梅.AADL 行为模型时间一致性验证方法.计算机技术与发展,2017,27(7):1−5.

[21] 侯刚.信息物理系统软件的形式建模、验证与分析[博士学位论文].大连:大连理工大学,2018.

[22] 罗晨霞,王瑞,关永,李晓娟,施智平,SONG XY.面向实时数据的 CPS 一体化建模方法.软件学报,2019,30(7):1966−1979. http://www. jos.org.cn/1000-9825/5753.htm [doi: 10.13328/j.cnki.jos.005753]

[23] 叶枫,夏阳,申朝祥,朱云超.基于动态行为建模的 CPS 计算实体建模方法.系统仿真学报,2016,28(5):1003−1008,1016.

[25] 苏琪,王婷,陈铁明,陈蓉蓉.基于时间自动机的 CPS 安全建模和验证.信息安全研究,2017,3(7):601−609.

[26] 张晶,王亮,范洪博.CPS 系统物理实体时空一致性建模与分析.计算机工程与应用,2018,54(14):41−44.

陈小颖(1997−),女,硕士,CCF 学生会员,主要研究领域为软件工程,形式化方法,CPS 系统.

赵宇(1997−),男,硕士,CCF 学生会员,主要研究领域为软件工程,软件缺陷预测,机器学习.

祝义(1976−),男,博士,教授,CCF 专业会员,主要研究领域为软件工程,形式化方法,CPS 系统,智能软件开发.

王金永(1983−),男,博士生,CCF 学生会员,主要研究领域为时空约束系统规约,协同无人驾驶安全性分析,形式化模型检测.