需求验证

徐思涵

南开大学

Slides adapted from materials by Prof. Xiaohong Chen (East China Normal University)

需求的形式化表示

广义地说，形式化方法指有严格数学基础的系统开发方法，支持计算机系统及软件的规约、设计、验证与演化等活动。

在需求阶段引入形式化方法，可以将系统的形式化验证提前到需求规约阶段，从而达到尽早发现错误，降低错误修改代价的目的。

在需求阶段进行形式化验证，需要：

（1）确定（需要构建的系统的）系统模型；

（2）确定系统需要满足的性质；

系统需要满足的性质，就是我们需要验证的性质，由性质规约语言描述，这些性质刻画所期望的系统行为。

（3）选择合适的验证工具。

验证工具承载了验证技术，包括模型检测和定理证明。

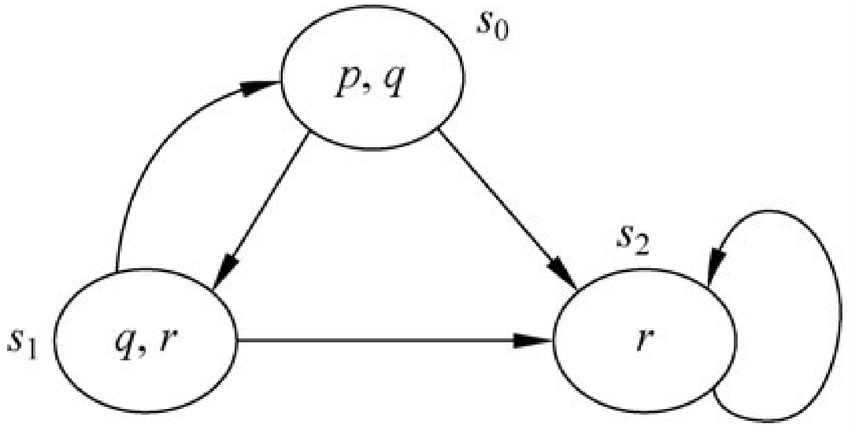
需求形式化建模

状态迁移系统(State Transition Systems)是一种基于状态迁移的计算模型，即通过状态（静态结构）和迁移（动态结构）来建模系统行为。它可定义为一个六元组：

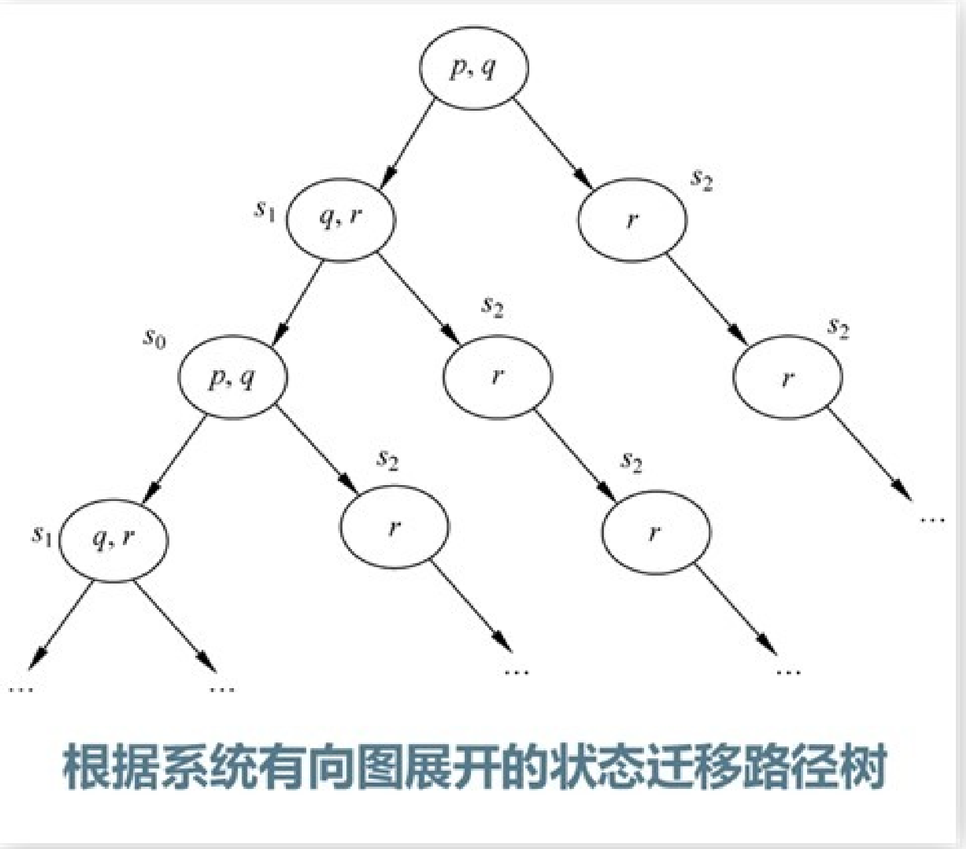
$$M:=(S,Act,\rightarrow,I,AP,L)\ ,$$

其中，S是状态集，包含系统可以处于的所有状态，Act是行为集，包含可以发生的所有事件，$\rightarrow\ \subseteq S\times Act\times S$是迁移函数，$I\subseteq S$ 是初始状态集，AP是命题集，$L{:}S\rightarrow 2^{AP}$是标签函数，表示如果系统处于某个状态，则有和该状态关联的一组命题为真。

状态迁移系统-图形表示



迁移系统作为有向图的简明表示



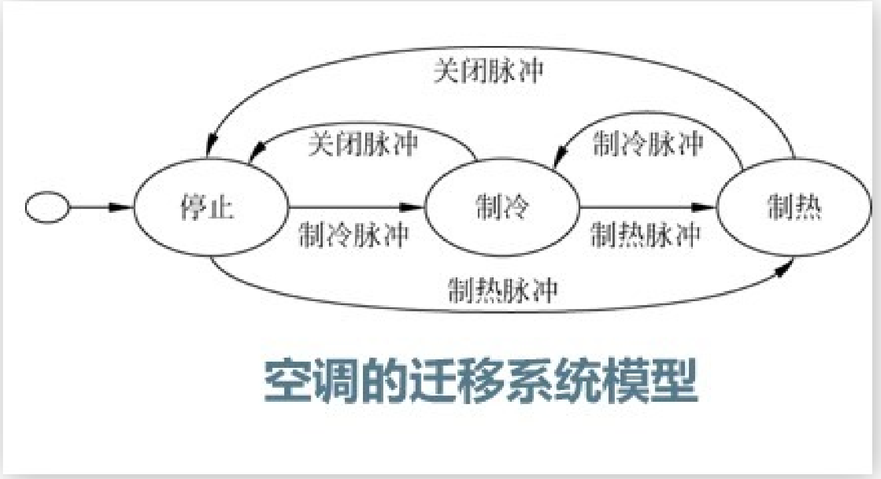
根据系统有向图展开的状态迁移路径树

例如，假设存在一个状态迁移系统，它有三个状态s_0、s_1和s_2，初始状态为s_0，状态间的迁移有$s_0 \rightarrow s_1$，$s_0 \rightarrow s_2$，$s_1 \rightarrow s_0$，$s_1 \rightarrow s_2$和$s_2 \rightarrow s_2$，若有$L(s_0)=\{p,q\}, L(s_1)=\{q,r\}$，$L(s_2)=\{r\}$三个标签函数，其中p,q,r,为三个命题

状态迁移系统的行为解释为： 从初始状态$s_0 \in I$开始，根据当前发生的事件，以及迁移关系→，决定系统状态的演化过程。具体地说，如果系统的当前状态为s，当事件α发生时，则系统不确定地选择迁移关系 $s \rightarrow (a) s'$(即 $(s,\alpha,s') \in \rightarrow$)，并将当前状态从s演化为s'。在当前状态为 s'的时候重复这个过程，直到系统处于某个没有可迁移的状态，则状态演化结束。

状态迁移系统例子

空调的状态变迁系统



空调的迁移系统模型

- 其中，**椭圆代表状态，带标记的边代表状态迁移关系，空心椭圆指向的状态为初始状态。**
- 这里，状态集合S={停止,制冷,制热}。初始状态集合只有一个状态，即I={停止}。
- 行为集Act={关闭脉冲,制热脉冲,制冷脉冲}，其中“关闭脉冲”表示关空调的信号，“制热脉冲”表示让空调制热的信号，“制冷脉冲”表示让空调制冷的信号。
- 带标记的边是空调的状态变迁关系，例如，“停止制冷脉冲制冷”表示当空调处于“停止”状态时，收到“制冷脉冲”信号，则空调进入“制冷”状态。

空调要满足的原子命题可以完全和其状态的含义一致，即对任意状态s，有L(s)={s}。例如L(停止)={停止}，这是一种最简单的情况，即在状态命名的时候能找到和所关注的现实世界性质完全一致的状态名。

需求的状态迁移系统表示

- **首先考虑状态的选取，系统状态的选取跟建模的目的直接相关，对于同一个系统，建模的目的不同，其状态的选取也会不同**。例如，同样是一盏灯，如果只关心灯亮还是不亮，则会将“灯亮”作为一个状态，“灯灭”作为另一个状态； 而如果关心的是灯的亮度是不是适合学习，则假设灯的照度在300~500勒克斯时适合学习，为“亮度适合”状态，照度小于300勒克斯为“亮度偏弱”状态，照度大于500勒克斯作为“亮度偏强”状态。
- 除了状态集合，**还必须描述状态之间的迁移关系**。状态间的迁移关系通常可以这样确定： 从任意一个状态出发，考虑该系统是否可能从这个状态直接变化为其他某个状态，如果可能，则需要进一步确定能触发这个变化的事件，这就确定了该系统的一个状态迁移关系。例如，对于上述仅关心灯亮还是不亮的情形，当灯处于“灯亮”的状态时，收到“关脉冲”，则变为“灯灭”状态； 而当灯处于“灯灭”的状态时，收到“开脉冲”，则变为“灯亮”状态。
- **最后要确定原子命题集合AP，以描述所关注的物理世界情形和系统状态间的对应关系**，即确定一组关于现实世界的原子命题，将它们作为系统状态选择的依据，反过来说，**原子命题就是系统处于某个状态所能确定为真的关于现实世界事实的命题**。如上述例子中，系统处于“亮度偏弱”状态则意味着当前照度小于300勒克斯，处于“亮度偏强”状态则意味着当前照度大于500勒克斯。**如果原子命题只在一个系统状态下满足，也可以直接将原子命题作为状态名**，即对任意状态s，L(s)={s}，如上述将灯亮这个现实世界命题对应于“灯亮”状态，将灯不亮这个现实世界命题对应于“灯灭”状态。

性质规约语言简介

性质规约语言以时态逻辑为基础。时态逻辑有很多种，本节介绍常用的线性时态逻辑和计算树逻辑。

(1) 线性时态逻辑(Linear Temporal Logic，LTL)

线性时态逻辑将时间建模成状态的序列，无限延伸到未来。这个状态序列有时称为计算路径(简称路径)。一般来说，未来是不确定的，因此考虑若干路径，代表未来的不同可能，任何一条都可能是未来的“实际”路径。

线性时态逻辑的基本成分包括： 关于现实世界情形的原子命题(a∈AP)，布尔连接子（如∧），以及两个基本的时态算子O(下一个状态)和U(直到)。其中，原子命题(a∈AP)直接和状态变迁系统的相关联，时态算子为一元前缀算子，其参量为一个LTL公式，其含义是： 公式Oφ在当前时刻成立，则φ在下一时刻成立。U 时态算子为二元时态算子，带两个LTL 公式作为其参量，φ1Uφ2在当前时刻成立，则φ1一直成立直到在未来某个时刻φ2成立。

LTL的语法定义如下。

定义 原子命题公式集AP上的LTL公式由如下语法构成

其中，a∈AP为原子公式。 $\varphi ::= \text{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathrm{O}\varphi \mid \varphi_1 \mathrm{U} \varphi_2$

性质规约语言简介

(2) 计算树逻辑(Computational Tree Logic，CTL)

计算树逻辑是一种分支时间逻辑，它的时间模型是一个树状结构，表明未来是不确定的。在未来的不同路径中的任何一条都可能是“实际”路径。

定义9.2原子命题公式集AP上的CTL公式由如下语法构成：

$$\Phi ::= \text{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg \Phi \mid \exists \varphi \mid \forall \varphi$$

$$\varphi ::= \bigcirc \Phi \mid \Phi_1 \mathrm{U} \Phi_2$$

其中，a∈AP为原子公式，φ是路径公式，Φ、Φ1和Φ2是状态公式。状态公式中的两个带量词的公式分别表示“存在一条路径”和“对所有路径”。

有一些经常需要表达的性质，如下：

存在一个可达状态满足a。可以写为 $\exists \bigcirc \exists \bigcirc \cdots \exists \bigcirc a$

从所有满足a的可达状态出发，可以连续保持a的可满足性，直到到达一个满足b的状态。可以写为 $\forall \Box (a \rightarrow \exists (a \mathrm{U} b))$

如果满足a的状态可达，可以永远连续不断满足b。可以写为 $\forall \Box (a \rightarrow \exists (\text{true} \mathrm{U} \Box b))$

存在一个可达状态，由其出发的所有可达状态都满足b。可以写为： $\forall \Box b$

性质的时态逻辑表示

时态逻辑公式

需求的通用性质和特定性质都要用时态逻辑公式表达出来，才能使用模型检测器进行验证。在需求模型验证中，要验证的性质基本上都转换为**可达性或无死锁性**，但具体怎么转换，不同模型检测器会有不同的方式，具体如何描述也依赖于所使用的语言和工具。

下面列举一些常见性质描述的例子。

（1） 安全性： 两列火车Train1和Train2绝不同时进入同一个轨道区段Track。

$$\Box(\neg \mathrm{Train}_1 \text{ in Track} \vee \neg \mathrm{Train}_2 \text{ in Track})$$

（2） 活性： 每列火车都能无限多次进入同一条轨道区段。

$$\Box\Diamond \text{ Train in Track}$$

（3） 弱活性： 每列等待的火车总能进入某个路段。

$$\Box\Diamond \text{Train waitfor Track} \rightarrow \Box\Diamond \text{Train in Track}$$

需求验证

验证技术：模型检测技术

需求验证中最常见的验证技术是模型检测。模型检测就是对模型的状态空间进行搜索，以确定该系统模型是否满足某些性质，搜索的可中止性依赖于模型的有限性。在模型检测中，模型M一般为迁移系统，性质Φ是时态逻辑公式，而验证过程就是计算模型M是否满足Φ，即：

$$M \models \Phi$$

需求工程中存在需求R、环境E和需求规约S三部分描述，它们之间存在E,S ⊨ R关系。根据待验证的模型和性质的表示，需求验证涉及如下几种类型的验证。

（1） 需求R满足通用性质Φ，Φ可以是一致性、完整性等性质： $R \models \Phi$

（2） 规约S满足通用性质Φ： $S \models \Phi$

（3） 实现规约S的软件部署到环境E中后满足通用性质Φ： $E,S \models \Phi$

（4） 实现规约S的软件部署到环境E后满足需求R： $E,S \models R$

模型检测工具

业界有很多模型检测工具，它们**使用的模型规约语言都有一些不同，所支持的时态逻辑也不尽相同**，所以在模型检测的过程中需要了解所采用的模型检测器的要求。下面是一些在需求阶段常用的模型检测器。

- **UPPAAL**(Uppsala University & Aalborg University)： 时间自动机的模型检测工具，用于建模和模拟及验证实时系统的工具，支持网络化时间自动机和数据类型以及概率模型检验。
- **SMV**(Symbolic Model Verifier)： 符号模型检测器，用来检测有限状态系统是否满足CTL公式。
- **NuSMV**(New Symbolic Model Verifier)： 新符号模型检测工具，重构了SMV，支持用CTL和LTL描述，整合了以SAT为基础的有界模型检测技术。
- **SPIN**(Simple Promela Interpreter)： 适用于验证并发系统，用以检测有限状态系统是否满足PLTL（Propositional Linear Temporal Logic,命题线性时序逻辑）公式及其他一些性质，包括可达性和循环。建模语言为PROMELA(PROcess MEta LAnguage)。
- **MyCCSL**： 基于约束求解器Z3的有界模型检测器，用于支持实时和嵌入式系统的建模和分析，其输入语言为时钟约束规范语言（CCSL），可以建模时钟之间的关系，提供处理逻辑时钟的具体语法。

案例研究

NuSMV简介

本节采用NuSMV作为模型检测器，SMV作为模型规约语言，CTL作为性质规约语言。

NuSMV简介

NuSMV(New Symbolic Model Verifier)针对有限状态自动机系统提供模型检测能力，支持CTL和LTL描述的性质规约。NuSMV 程序由一个或多个模块构成，其中有一个主模块(main)。**模块可以声明变量并赋值，赋值通常给出变量的初始值(initial)，其随后值(next)是关于变量当前值的表达式**。LTL(或CTL)规范由关键词LTLSPEC(或者CTLSPEC)引入。为了方便表示，NuSMV分别用标准键盘上的&、|、→、!、F、G、E和A来分别表示∧、∨、→、◇、□、¬、∃和 ∀。

```
MODULE main
VAR
        request: boolean;
        state :{ready,busy};
ASSIGN
       init(state) :=ready;
       next(state) := case
                request = 1 & state = ready : busy;
                TRUE: ready;
         esac;
LTLSPEC
       G(request->F state=busy)
```

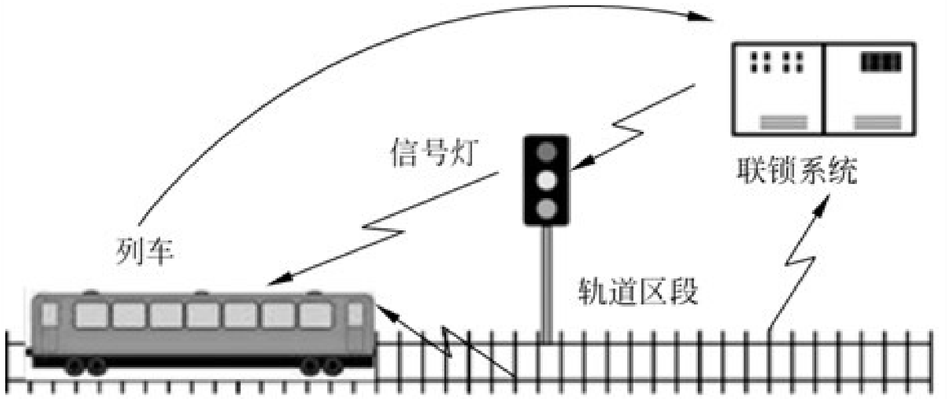
NuSMV程序举例

右图给出NuSMV程序本章中的NuSMV工具采用其2.6.0版本。的例子，该程序有两个变量，布尔(boolean)型的request 和枚举型{ready,busy} 的state，其中0代表“假”，1代表“真”。变量request 的初始值和随后值在这个程序中不确定，表明它的取值是由外部环境决定的。state的初值是ready，当request为1且state为ready时变为busy，否则(即文中的TRUE)state的随后值为ready。**待验证的性质描述为LTLSPEC**，表示一旦有request，则未来state会变成busy。

案例研究

简化的轨道交通联锁系统

在轨道交通系统中，联锁系统是其信号系统的核心设备，用于保证列车行车安全。铁路、地铁车站及车辆段都有很多线路，线路的两端以道岔连接，根据道岔的不同位置组成列车的不同进路，每条进路只允许一辆列车使用。列车能否进入某进路，如何避免发生进路冲突，这些都由联锁系统来协调。通常，一个铁路编组站由四类组件构成：轨道区段、道岔、进路和信号灯。联锁系统的主要功能包括轨道区段的分配、进路控制、道岔控制、信号控制等。



轨道交通联锁系统示意图

一个轨道交通联锁系统的简化场景如下： 一辆列车(train，含车载系统)在进入轨道之前需要向联锁系统(controller)发出请求。联锁系统controller查询轨道的占用状态，决定是否接受其请求。若列车收到红灯信号，则列车等待并重复发送请求； 若收到绿灯信号，则列车等待道岔打开，进入相应的进路。右图给出了轨道交通联锁系统的示意图。

系统模型构建

a: 联锁系统!{forward, backward}
b: 轨道区段!{ocupied, unoccupied}；联锁系统!{query}
c: 联锁系统!{red, green}
d: 列车!{request}；联锁系统!{wait, enter, leave}

联锁系统的上下文图

第一步：结构建模

确定系统的组件及这些组件之间的交互关系，可以用上下文图表示。

识别实体：针对这个简化的联锁系统(controller)，可以识别出该系统有4个环境实体： 轨道区段(track)、道岔(switch)、列车(train)和信号灯(light)。

识别状态：轨道区段在占用状态和未占用状态之间来回转换，道岔在正向与反向状态之间来回转换，信号灯在红灯与绿灯之间转换，列车在发送请求之后需要经历从等待、进入到离开的过程。

第二步：组件建模

上述的组件识别过程已经大致确定了每个组件的行为。

例如，联锁系统的行为如下： 收到列车的请求之后，查看轨道状态，若其当前状态为“占用”，则给信号灯发出红灯脉冲，指示信号灯进入“红灯”状态，达到让列车等待的目的。联锁系统按照一定的时间频率接收列车的请求，若发现轨道状态为“未占用”，则给信号灯发出绿灯脉冲，指示信号灯进入“绿灯”状态，达到通知列车可以进入轨道的目的，同时控制道岔，让列车进入。之后联锁系统处于空闲状态，等待下一次列车的请求信号。

以上述行为分析的结果为依据，定义各组件的状态。

首先考虑各组件的可能状态。

- 对信号灯而言，有红灯(red)和绿灯(green)两个状态;
- 对轨道区段而言，有未占用(unoccupied)和占用(occupied)两个状态;
- 道岔因为有正向和反向两种情况，有正向(forward)和反向(backward)两个状态，假设反向为允许进入;
- 列车有请求已发送(requested)、等待(wait)、进入(enter)和离开(leave)四个状态;
- 系统的状态包括： 已接收请求(receiveRequest)、轨道查询(queryTrack)、已发送等待(sendWait)、已发送进入(sendEnter)和空闲(idle)。

然后，分别定义初始状态。

- 信号灯的初始状态为red，轨道区段为unoccupied，列车为requested，道岔为forward，联锁系统为idle。

第三步：定义求随后状态的函数

（1） next(controller):

如果controller处于idle或sendWait状态且列车处于requested状态，则函数值为receiveRequest；如果controller处于receiveRequest状态，则函数值为queryTrack；

如果controller处于queryTrack状态且轨道处于unoccupied状态，则函数值为sendEnter； 如果controller处于queryTrack状态且轨道处于occupied状态，则函数值为sendWait；

如果controller处于sendEnter状态且列车处于enter状态，则函数值为idle； 其他情况下函数值不变。

（2） next(light):

如果light处于red状态，controller处于sendEnter状态且train处于requested状态，则函数值为green； 其他情况下函数值为red。

（3） next(train):

如果train处于requested状态，light处于green状态且switch处于forward状态，则函数值为enter； 如果train处于requested状态，且controller处于sendWait状态，则函数值为wait； 如果train处于enter状态,且track处于occupied状态，则函数值为leave； 如果train处于leave状态或wait状态，则函数值为requested； 其他情况下函数值不变。

（4） next(track):

如果track处于unoccupied状态，且train处于enter状态，则函数值为occupied； 如果track处于occupied状态，且train处于leave状态，则函数值为unoccupied；

其他情况下函数值不变。

（5） next(switch):

如果switch处于forward状态，且controller处于sendEnter状态，则函数值为backward； 其他情况下函数值为forward。

最后，得到联锁系统的SMV模型，如右图所示。

```
MODULE main
VAR
light : {green,red};
track : {occupied,unoccupied};
switch : {forward,backward};
train : {requested,wait,enter,leave};
controller : {receiveRequest,sendWait,queryTrack,sendEnter,idle};
ASSIGN
init(light) := red;
init(track) := unoccupied;
init(switch) := forward;
init(train) := requested;
init(controller) := idle;
next(controller) := case
          (controller=idle|controller=sendWait) &train=requested : receiveRequest;
          controller=receiveRequest : queryTrack;
          controller=queryTrack & track=unoccupied : sendEnter;
          controller=queryTrack & track=occupied : sendWait;
          controller=sendEnter & track=enter : idle;
          TRUE : controller;
     esac;
next(light) := case
          light=red&controller=sendEnter & train=requested: green;
          TRUE : red;
     esac;
next(train) := case
          train=requested & light=green & switch=forward : enter;
          train=requested & controller=sendWait : wait;
          train=enter & track=occupied : leave;
          train=leave | train=wait : requested;
          TRUE : train;
     esac;
next(track) := case
          track=unoccupied&train=enter : occupied;
          track=occupied&train=leave : unoccupied;
          TRUE: track;
     esac;
next(switch) := case
          switch =forward & controller=sendEnter : backward;
          TRUE: forward;
     esac;
```

联锁系统的NuSMV程序

验证性质

（1） 一致性： 无冲突。

要求联锁系统的需求是一致的，即没有冲突需求。可以表示为在任何情况下，都不可能出现同一组件处于两个不同的状态。

AG!((light=green & light=red)|(track=occupied & track=unoccupied)|(switch=forward & switch=backward)|(train=requested & train=wait)|(train=requested & train=enter)|(train=requested & train=leave)|(train=wait & train=enter)|(train=wait & train=leave)|(train=enter & train=leave)|(controller=idle & controller=receiveRequest)|(controller=idle & controller=queryTrack)|(controller=idle & controller=sendWait)| (controller=idle & controller=sendEnter) |(controller=receiveRequest & controller=queryTrack)|(controller=receiveRequest & controller=sendWait)|(controller=receiveRequest & controller=sendEnter)|(controller=queryTrack & controller=sendWait)|(controller=queryTrack & controller=sendEnter)|(controller=sendWait & controller=sendEnter))

(2) 领域特定的性质： 绿灯，则车辆进入； 红灯，则车辆不能进。

需求可满足性，需求可以表示为如绿灯则列车进入，红灯则列车不能进站等。

AG (light=green AF train=enter)

AG (light=red AF (train!=enter))

将模型与性质输入NuSMV验证器中，即可验证这些性质都是可满足的。

小结与讨论

- 本章介绍了形式化需求的验证方法。
- 尽管形式化方法有诸多好处，但形式化方法要求精确的表示，因此在需求阶段（特别需求的早期阶段）不是很适用。
- 在需求的早期阶段有一些半形式化(或者结构化)的需求建模方法，也能支持一定程度上的需求验证。
  - ✓ 一是直接使用类似UML这样的可视化的建模语言，并在其预定义的形式化语义基础上，支持从所表达的需求到形式化模型的转换，从而支持形式化验证，这类方法的关键在于模型的转换；
  - ✓ 二是定义类自然语言(又称为模式语言)，这类语言在表述需求方面与自然语言类似，易于表达，和上述半形式化方法一样，也为类自然语言赋予形式语义，将其与形式化模型联系起来，从而支持形式验证。其难点在于如何根据问题需求总结其语言表达模式，以及如何建立语言表达模式和形式语义之间的关系。

思考题

1. 需求阶段一般需要验证哪些性质?

2. 假设一个系统的描述中有如下原子命题： 忙(busy)、开始(started)、准备好(ready)、确认(acknowledged)、重启(restart)和请求(request)。如何用时态逻辑描述如下性质?

不可能到达一个started成立但ready不成立的状态;

可能到达一个started成立但ready不成立的状态:

对任何状态，如果一个(对某些资源的)request发生，那么它将最终被acknowledged;

不管发生什么情况，一个特定过程最终被永久死锁(deadlock);

从任何状态出发都可能到达一个restart状态。

3. 请使用NuSMV对如下案例进行建模和验证。

地铁屏蔽门控制系统就是控制站台屏蔽门的开关。当列车到站并停在允许的误差范围内时(如±300mm)，信号系统将向屏蔽门控制系统发送开门指令，打开屏蔽门。当列车驾驶员或站务人员通过就地控制盘发出关门命令时，系统关闭站台屏蔽门。

4. 请选择合适的形式化语言对如下机房自动温度控制系统进行建模，并进行性质的验证。

机房服务器持续运作会导致机房温度过高，产生安全隐患，需要设计一套机房温度自动控制系统来解决这个问题。假设在一个密闭的机房中放置一台服务器、一台空调和一个温度传感器，服务器持续工作会带来房间热量的增加，温度传感器负责检测温度并对空调发出信号，空调负责制冷降温。当机房的温度超过30℃时，将打开空调； 当机房的温度低于20℃时，关闭空调。

参考文献

[1] European Committee for Electrotechnical Standardization.BS EN 50129: Railway application Communications,signaling and processing systems—Safety related electronic systems for signaling [EB/OL] .(2019 05 13) [2022 12 05] .https://www.en standard.eu/bs en 50129 2018 railway applications communication signalling and processing systems safety related electronic systems for si
gnalling/.

[2] RTCA. DO 178C Software Considerations in Airborne Systems and Equipment Certification [M] .Washington，USA： RTCA Inc.,2011.

[3] Yuan Z,Chen X,Liu J，et al.Simplifying the Formal Verification of Safety Requirements in Zone Controllers through Problem Frames and Constraints Based Projection [J] .IEEE Transactions on Intelligent Transportation System，2018,19(11)： 3517 3528.

[4] 刘筱珊,袁正恒,陈小红,等.区域控制器的安全需求建模与自动验证 [J] .软件学报,2020,31(5)： 1374 1391.

[5] Chen X,Wu X,Zhao M,et al.Verifying the Relationship Among Three Descriptions in Problem Frames Using CSP [C] //TASE 2019： 248 255.

[6] Baier C,Katoen J.Principles of Model Checking [M] .Cambridge,MA: MIT Press,2008.

[7] Liu S.Formal Engineering for Industrial Software Development [M] .Berlin: Springer,2004.

[8] Aceituna D,Do H,Srinivasan S.A Systematic Approach to Transforming System Requirements into Model Checking Specifications [C] //ICSE Companion 2014： 165 174.

[9] Bultan T,Heitmeyer C.Analyzing Tabular Requirements Specifications Using Infinite State Model Checking [C] //MEMOCODE 2006： 7 16.

[10] Shrotri U,Bhaduri P,Venkatesh R.Model Checking Visual Specification of Requirements [C] //SEFM 2003： 202 209.

[11] Choi Y,Rayadurgam S,Heimdahl M.Toward Automation for Model Checking Requirements Specifications with Numeric Constraints [J] .Requirement Engineering,2002,7(4)： 225 242.

[12] Fuxman A,Mylopoulos J,Pistore M,et al.Model Checking Early Requirements Specifications in Tropos [C] //RE 2001： 174 181.

[13] Bharadwaj R,Heitmeyer C.Model Checking Complete Requirements Specifications Using Abstraction [J] .Automated Software Engineering,1999,6(1)： 37 68.

[14] Heitmeyer C,Kirby J,Labaw B,et al.Using Abstraction and Model Checking to Detect Safety Violations in Requirements Specifications [J] .IEEE Transactions on Software Engineering,1998,24(11)： 927 948.

[15] Clarke E,Henzinger T,Veith H,et al(eds).Handbook of Model Checking [M] .Cham,Switzerland： Springer International Publishing AG,2018.

[16] 王戟,詹乃军,冯新宇,等.形式化方法概貌 [J] .软件学报,2019,30(1)： 33 61.

[17] Heitmeyer C,Jeffords R,Labaw B.Automated Consistency Checking of Requirements Specifications [J] .ACM Transactions on Software Engineering and Methodology,1996,5(3): 231 261.