基于 Object-Z 的形式化验证方法*)

文志诚　缪淮扣　张新林
(上海大学计算机学院　上海 200072)

摘　要　定理证明是一种形式化验证技术,也是形式化方法的重要组成部分,它能从形式规格说明中推理出应具备的性质与属性,从而可以对规格说明进行形式验证。Object-Z 是形式规格说明语言 Z 的面向对象扩充,基于集合论与数理逻辑,具有严密的逻辑性,适合精确地描述大型软件系统,并且可以对其形式规格说明进行推理。本文首先给出了基于 Object-Z 规格说明的定理证明验证方法,接着用 Object-Z 描述了一个电梯操作系统的实例,在此基础上给出了其形式规格说明的定理证明方法来进行形式化验证。

关键词　Object-Z,形式化验证,前后置条件,状态空间,电梯操作系统

Formal Verification Based on Object-Z Specification

WEN Zhi-Cheng　MIAO Huai-Kou　ZHANG Xin-Lin
(School of Computer Engineering and Science, Shanghai University, Shanghai 200072)

Abstract　Theorem proof, an important compositive part of formal method, is one kind of formal verification, which can reason about the characters that the formal specification should hold for formal specification. Therefore, it can verify the formal specification. Object-Z, an extension to formal specification language Z, is apt to describing large scale object-oriented software specification and can reason about the formal specification. This paper presents a verification approach with theorem proof for Object-Z specification and describes an example of elevator operation system, whose relevant verification methods are given.

Keywords　Object-Z, Formal verification, Pre-&Post-condition, State space, Operation system of elevator

1　前言

形式化方法能使软件开发人员创建比那些使用传统方法或面向对象方法产生的需求规格说明更为完整、一致和无二义性的规格说明,能帮助开发人员发现用其它方法不容易发现的系统描述不一致性、不明确性或不完整性,有助于增加软件开发人员对系统的理解。因此,形式化方法是提高软件系统,特别是 safety-critical 系统的安全性与可靠性的重要手段。

形式化规格说明是用形式化说明语言来描述软件系统。形式软件开发是用形式语言来描述软件的需求规格,并且通过推理验证来保证最终的软件产品满足需求规格。形式化方法可以用于程序的正确性验证,来保证程序的正确性。因此,在开发过程中可以采用形式验证来及时检查出错误。开发过程中尽早地消除错误是改善软件质量的关键之一。

定理证明就是形式化验证技术的一种,它从形式规格说明中推理应具备的性质,看是否符合需求规格。若能推出应有的性质,则形式规格说明符合设计需求;否则,形式规格说明不完全或存在不一致性。定理证明是形式化方法的一个重要组成部分,没有证明就没有核心。

形式规格说明语言有许多种,如 VDM、B、Z、Object-Z 等。Object-Z[1]是形式规格说明语言 Z 的面向对象扩充,基于集合论与数理逻辑,具有严密的逻辑性,适合精确地描述大型软件系统,并且可以对其规格说明进行形式推理,文[2]已给出了其相应的推理规则与验证方法,可以推理出规格说明应该具有的性质与属性。标准的 Object-Z 是采用历史语义[3]把一个对象的状态演变过程看成是一个历史,由状态及操作事件构成。

本文给出了基于 Object-Z 的一般形式验证的定理证明方法,能验证所书写的形式规格说明,并用 Object-Z 描述了一个电梯操作系统的实例,在此基础上给出了其定理证明的形式化验证。

2　Object-Z 简介

Object-Z 是 Z 的面向对象扩展,一个形式规格说明包括一些全局定义和类定义。全局定义可以被系统中所有的类共享,它定义在所有的类的外面,不属于任何一个类。全局定义和在一个类中的其余部分使用 Z 的定义方式。类是它的一个重要的特征,它的结构有如下形式:

```
Classname[Parameters]
  Inherited classes
Type definitions
Constant definitions
State schema
Initialization schema
 Operation schema
```

一个类能够带有参数,可以继承它的父类,与父类同名的变量及操作采取谓词合取的方式。父类的操作在子类继承过

*)国家自然科学基金(60373072)和上海市教委第四期重点学科建设基金资助。文志诚　博士生,讲师,主要研究领域为面向对象技术与形式化方法。

程中被改名，可作为另一个操作被继承下来，而子类中可以重新定义一个同名的操作。类中还涉及常量定义、变量定义、无名状态模式、初始状态模式及操作模式。

其中，状态变量定义在无名状态模式中，可以被此类中所有的操作使用；可以有其约束条件，即状态不变式，它定义了状态空间，对象的每个状态是其中的一个状态。初始化操作模式用来给状态变量赋初值。操作中可以有变量定义部分及谓词部分，变量定义部分可以声明在本操作中要改变的状态变量及可以定义局部的输入与输出变量，这些局部输入与输出变量只能在本操作中起作用，其它的操作不能更改这些局部变量；谓词部分隐式定义了其前置条件及后置条件。全局变量、状态变量及局部输入与输出变量都有其相应的类型。

3 形式验证方法

3.1 初始状态存在验证

初始状态是状态空间里的一个状态，表示一个对象在进化过程中所处的初始状态。在 Object-Z 中，初始状态用一个名为 Init 模式表示。书写完一个形式规格说明，必须验证其初始状态存在，即是状态空间中的一个状态。有表达式为：

$$Context \vdash \exists x: X \bullet Invariant(x) \wedge Init(x)$$

Context 表示上下文环境，包括全局变量、类中的局部类型、函数及常量定义等。$x: X$ 表示类中状态空间中定义的状态变量 x 及其相应的类型 X。Invariant(x)表示状态变量的不变式，不变式定义在无名状态模式中。Init(x)表示相应的状态变量的初始状态。

如果此表达式为真，说明初始状态存在。否则，说明形式规格说明不完全或存在矛盾，因此初始状态存在性验证是必要的。

3.2 前后置条件验证

设一个类的不变式为 Invariant(x)，其中 $x: X$ 是所涉及的状态变量及其类型；一个操作中可能的输入与输出变量及类型为 input : INPUT，output : OUTPUT。根据 Object-Z 前置条件与后置条件的求法，状态不变式含蓄地包含在每一个操作中，因此有

$$OP.pre = \exists x': X; output: OUTPUT \bullet OP(x, x', input, output) \wedge Invariant(x) \wedge Invariant(x')$$

$$OP.post = OP(x', output) \wedge Invariant(x')$$

其中 Invariant(x′)表示不变式中的状态变量 x 用其相应的后状态变量 x′代替；OP(x′, output)表示操作部分，与后状态变量 x′和输出变量 output 有关。后置条件只保留含有后状态变量的谓词表达式，前置条件和后置条件表达式里包含前状态和后状态变量。由求法可知，前状态和后状态变量可能会出现在同一个谓词表达式中。如果一个操作施用于它的一个对象，那么此对象的当前状态一定要满足它的前置条件。执行完毕，该对象的状态得到修正。

在书写完一个形式规格说明后，必须求出类中操作的前置条件和相应的后置条件。对前置条件来说，可以检验一个操作前置条件是否满足我们的需求规格。对于后置条件，操作实施后可能会修改状态变量，因此可以检验是否满足需求规格。

3.3 推理类中的性质

当用 Object-Z 书写了一个形式规格说明，我们可以验证此规格说明的一些性质是否满足我们的需求。如果满足，说明此部分的规格说明是合理的，否则不完善。为了验证类中的性质，我们可以用 Graeme Smith[2] 的扩展的 W 逻辑，它有形式为：A::d | Ψ ⊢ Φ，其中 A 为一个类，d 为声明表，Ψ 为谓词集合，称为“前提”，Φ 也是谓词集合，称为“结果”。在 d 的环境条件下，当 Ψ 的所有谓词均为真时，Φ 里的谓词合取也为真。

用此方法，我们可以推理一个操作的性质，可以检验是否与需求规格相符。如果推理出来的性质与需求不相符，说明此部分形式规格说明不完全或不一致。

4 实例：电梯操作系统

在本节中，我们将给出一个电梯操作系统的实例来具体说明。实例分两个部分：首先用 Object-Z 形式语言来描述电梯操作系统(4.1)；其次，用前面给出的验证方法来验证我们所描述的形式规格说明(4.2)。

4.1 描 述

对于一个电梯操作系统，可以把一台电梯看成是一个对象，它是由一个电梯类(Elevator)实例化而成，所有的功能及性质都在电梯类中描述。电梯类有关的操作为：初始化操作(Init)、申请按键操作(Request)、电梯运行操作(Run)、进出人员操作(Add_dec_man)。其结构如下所示：

Elevator	
Type definitions	//4.1.1
Constant definitions	//4.1.1
State schema	//4.1.2
Init	//4.1.3
Request	//4.1.4
Run	//4.1.5
Add_dec_man	//4.1.6

为了方便起见，我们将把电梯类中的各个部分分别描述。

4.1.1 类型定义和常量定义

常量定义(Constant definitions)和类型定义(Type definitions)可以放在一起，定义所需要的常量、类型及函数，可以作为初始状态的存在性验证的上下文环境(Context)。一台电梯能够上升到的最大高度 maxfloor 及电梯里最多能容纳的人数 maxsize 是常量。在常量定义部分，可以定义相关能用到的函数，这里有取最大及最小值的函数为 max 和 min、电梯上升一层或下降一层所用到的函数 add 和 dec 及计算电梯方向函数 cal。在电梯类中会用到一些自定义的类型，它们可以看作是一个自定义的集合，如楼层的集合(A)、电梯里最多人数的集合(B)、电梯门的状态集合(State)及电梯运行方向标志集合(Flag)等。

| maxfloor, maxsize: N

A = 1..maxfloor

B = 0..maxsize

State ::= open | close

Flag ::= up | 0 | down (0 表示电梯目前可以上也可以下，运行方向未定)

电梯目前运行方向向上(T = up)且向上申请集合不为空(#s1 > 0)，或者电梯目前方向未定(T = 0)且向上申请集合里的元素比向下申请集合里的元素要多(#s1 > #s2)，则经过函数计算(cal)，电梯运行方向为上升(T = up)。向上申请集合表示要求上升所到的层数的集合，由电梯里面及外面按键情况构成，每一按键就进入具体的集合。这里 T，s1，s2 是参数，由计算时具体决定。其他两个类似。

max, min: $\mathbb{F}_1\mathbb{N}\rightarrow\mathbb{N}$

$max=\{s:\mathbb{F}_1\mathbb{N};\ m:\mathbb{N} \mid m\in s\wedge(\forall\ n:s\cdot m\geq n)\ \cdot\ s\mapsto m\}$

$min=\{s:\mathbb{F}_1\mathbb{N};\ m:\mathbb{N} \mid m\in s\wedge(\forall\ n:s\cdot m\leq n)\ \cdot\ s\mapsto m\}$

add, dec: $\mathbb{N}\rightarrow\mathbb{N}$

$\forall x:\mathbb{N}\cdot add(x)=x+1$

$\forall x:\mathbb{N}\cdot dec(x)=x-1$

cal: $Flag\times\mathbb{P}A\times\mathbb{P}A\rightarrow Flag$

$\forall T:Flag;\ s1,s2:\mathbb{P}A\cdot$

$cal(T,s1,s2)=up\Leftrightarrow(T=up\wedge\#s1>0)\vee(T=0\wedge\#s1\geq\#s2)$

$cal(T,s1,s2)=down\Leftrightarrow(T=down\wedge\#s2>0)\vee(T=0\wedge\#s1<\#s2)$

$cal(T,s1,s2)=0\Leftrightarrow(\#s1=0\wedge\#s2=0)$

4.1.2 状态模式

状态模式(State schema)中定义了状态变量,它含有向上申请集合(upset)、向下申请集合(downset)、运行方向标志(tag)、电梯目前所在楼层(story)、电梯里人员数(number)及电梯门的状态(door),其状态不变式要求向上集合与向下集合不相交。

upset: $\mathbb{F}A$
downset: $\mathbb{F}A$
tag: Flag
story: A
number: B
door: State

$upset\cap downset=\varnothing$

4.1.3 初始化模式

初始化模式是对状态变量进行初始化,对相应的状态变量赋初值。

Init
$upset=downset=\varnothing$
$\wedge tag=0\wedge story=1$
$\wedge number=0\wedge door=close$

4.1.4 申请操作

它是需要乘电梯人员在电梯里面或电梯外面的按键操作,希望到某一楼层。如果人员在电梯外面,则是外申请;如果人员在电梯里面,是内申请。内外申请按键后,及所需达到的楼层进入相应的集合向上申请集合(upset)和向下申请集合(downset)中。外申请与内申请不同:外申请只是在外面按向上或向下标志符,故输入是一个数偶($A\times\{-1,1\}$),由所在的楼层与向上或向上的标志符构成(-1 表示向下,1 表示向上)。所在的楼层是默认的(属于集合 A),只需按上下标志符即可,它又分向上外申请与向下外申请。进入相应的向上申请或向下申请集合中需要函数 dom 计算,申请还与电梯目前所在的楼层有关。内申请只是在电梯内部按所需要的数字按键,没有方向。

• 外申请操作:外面申请向上(Outuprequest)与外面申请向下(Outdownrequest)操作

Outuprequest
Δ(upset, downset, door)
s?: $A\times\{-1,1\}$

$((tag=0\vee tag=up)\wedge ran(s?)=1\wedge dom(s?)=story$
$\wedge door=close\wedge door'=open$
$\wedge upset'=upset\wedge downset'=downset)$
$\vee(tag=up\wedge dom(s?)-story>0\wedge ran(s?)=1$
$\wedge downset'=downset$
$\wedge upset'=upset\cup\{dom(s?)\}\wedge door'=door)$

Outdownrequest
Δ(upset, downset, door)
s?: $A\times\{-1,1\}$

$((tag=0\vee tag=down)\wedge ran(s?)=-1\wedge dom(s?)=story$
$\wedge door=close\wedge door'=open$
$\wedge upset'=upset\wedge downset'=downset)$
$\vee(tag=down\wedge story-dom(s?)>0\wedge ran(s?)=-1$
$\wedge downset'=downset$
$\wedge upset'=upset\cup\{dom(s?)\}\wedge door'=door)$

人员在电梯外面申请操作是内向下和内向上申请的复合,即是它们两个的选择,只能其中之一,不能同时执行,因此有:Outrequest=Outuprequest □ Outdownrequest。符号"□"是 Object-Z 的操作连接符,表示两者可选一。

• 内申请操作:内申请操作即在电梯里面按键,到希望的楼层:

Inrequest
Δ(upset, downset, tag)
s?: A

$s?\notin upset\wedge s?\notin downset\wedge$
$((s?-story>0\wedge downset'=downset$
$\wedge upset'=upset\cup\{s?\})$
$\vee(s?-story<0\wedge downset'=downset\cup\{s\}$
$\wedge upset'=upset))\wedge tag'=cal(tag,upset',downset')$

• 申请操作:申请操作是内申请和外申请的复合,只能是其中之一,因此有:Request=Inrequest □ Outrequest。

4.1.5 运行操作

电梯有向上运行(Uprun)与向下运行(Downrun),其复合便是电梯运行操作(Run)。

Uprun
Δ(upset, story, tag, door)

$(door=close\wedge tag=up\wedge min(upset)-story>1$
$\wedge story'=add(story)\wedge upset'=upset$
$\wedge downset'=downset\wedge tag'=tag\wedge door'=door)$
$\vee(door=close\wedge tag=up\wedge min(upset)-story=1$
$\wedge tag'=cal(tag,upset',downset')$
$\wedge downset'=downset\wedge upset'=upset\backslash min(upset)$
$\wedge story'=add(story)\wedge door'=open)$

Downrun
Δ(upset, story, tag, door)

$(door=close\wedge tag=down\wedge story-max(downset))>1$
$\wedge story'=dec(story)\wedge upset'=upset$
$\wedge downset'=downset\wedge tag'=tag\wedge door'=door)$
$\vee(door=close\wedge tag=down\wedge story-max(downset)=1$
$\wedge tag'=cal(tag,upset',downset')\wedge upset'=upset$
$\wedge downset'=downset\backslash max(downset)$
$\wedge story'=dec(story)\wedge door'=open)$

• 运行操作:两个运行操作复合便是电梯运行操作:Run=Uprun □ Downrun。

4.1.6 电梯里增加和减少人员操作

进人数为 n1?,出人数为 n2!,它们是局部变量,要求进出后的人数要少于等于电梯最大容纳的人数。

Add_dec_man
Δ(number, door)
n1?, n2!: B

$number+n1?-n2!\leq maxsize$
$\wedge door=open\wedge door'=close$
$\wedge number'=number+n1?-n2!$

4.2 形式验证

在4.1节中，描述了电梯操作系统。在本节中，我们需要验证所书写的形式规格说明是否满足需求规格，可以按第3节所描述的验证方法来验证。

4.2.1 *初始状态存在验证*

为了验证初始状态的存在性，我们只需验证下面的式子成立：

$$Context \vdash \exists x : X \cdot Invariant(x) \wedge Init(x)$$

其中，此处上下文环境有：$Context = [A = 1..maxfloor; B = 0..maxsize; Flag ::= up \mid 0 \mid down; State ::= open \mid close]$，因此只须验证：

$$Context \vdash \exists upset, downset: PA; tag: Flag; story: A \cdot number: B; door: State \cdot upset \cap downset = \emptyset \wedge upset \cap downset = \emptyset \wedge upset = downset = \emptyset \wedge tag = 0 \wedge story = 1 \wedge number = 0 \wedge door = close$$

利用点规则(one-point rule)化简有

$$Context \vdash \emptyset \in \mathbb{P}A \wedge 0 \in Flag \wedge 1 \in A \wedge 0 \in B \wedge close \in State \wedge \emptyset \cap \emptyset = \emptyset$$

由状态变量和给定类型有

$\emptyset \in \mathbb{P}A$，$0 \in Flag$，$1 \in A$，$0 \in B$，$close \in State$，$\emptyset \cap \emptyset = \emptyset$

即有 $Context \vdash true$，故初始状态存在。

4.2.2 *前后置条件验证*

对于前后置条件验证，我们必须先求出相应的操作前后置条件，再与需求规格相比较。如果相符，则此操作形式规格说明合理，否则不符合相应的需求。

· 对于外申请操作，按照前置条件的求法，有

$$PreOutrequest = \exists door'; State; upset', downset': PA \cdot ((tag = 0 \vee tag = up) \wedge ran(s?) = 1 \wedge dom(s?) = story \wedge door = close \wedge (door' = open \wedge upset' = upset \wedge downset' = downset) \vee (tag = up \wedge dom(s?) - story > 0 \wedge ran(s?) = 1 \wedge downset' = downset \wedge upset' = upset \cup \{dom(s?)\} \wedge door' = door)$$

再用点规则(one-point rule)把其中的谓词部分化简，即可得

$$PreOutrequest = ((tag = 0 \vee tag = up) \wedge ran(s?) = 1 \wedge dom(s?) = story \wedge door = close) \vee (tag = up \wedge dom(s?) - story > 0 \wedge ran(s?) = 1)$$

对于后置条件，把它相应的后状态变量及输出变量所在的表达式提出来：

$$PostOutrequest = (door' = open \wedge upset' = upset \wedge downset' = downset)) \vee (downset' = downset \wedge upset' = upset \cup \{dom(s?) \wedge door' = door\})$$

对于前置条件 PreOutrequest，有两种情况，即有 $(tag = 0 \vee tag = up) \wedge ran(s?) = 1 \wedge dom(s?) = story \wedge door = close$ 和有 $tag = up \wedge dom(s?) - story > 0 \wedge ran(s?) = 1$ 这两种情况。对于后置条件 PostOutrequest，同样也有相应的两种情况，即有 $door' = open \wedge upset' = upset \wedge downset' = downset$ 和有 $downset' = downset \wedge upset' = upset \cup \{dom(s?) \wedge door' = door\}$。前置条件和后置条件的第一种情况相互对应，它们的第二种情况相互对应。

对于第一种情况，前置条件表明，当电梯运行方向未定或向上 $(tag = 0 \vee tag = up)$，外面按键是向上 $(ran(s?) = 1)$ 和电梯目前位于按键的楼层 $(dom(s?) = story)$，且电梯门已关闭状态 $(door = close)$ 时，申请成功。其相应的第一种后状态表明两个申请集合不变 $(upset' = upset \wedge downset' = downset)$，门被打开 $(door' = open)$。这符合一般电梯要求。

对于第二种情况，前置条件表明电梯运行方向向上 $(tag = up)$，外面按键所在的楼层与电梯目前所在的楼层至少要隔一层 $(dom(s?) - story > 0$，因为电梯运行有惯性，如在当前层按键，不会马上停止)，且按键向上时 $(ran(s?) = 1)$，申请成功。其相应的第二种后状态表明向下申请集合不变 $(downset' = downset)$，门状态不变 $(door' = door$，因为电梯还没到按键所在的层)，向上申请集合中加入按键所在的楼层 $(upset' = upset \cup \{dom(s?)\})$。这符合设计要求。

对于其它的操作，可以类似有：

· 对于向下外申请操作

$$PreOutdownrequest = ((tag = 0 \wedge tag = down) \wedge ran(s?) = -1 \wedge dom(s?) = story \wedge door = close) \vee (tag = down \wedge story - dom(s?) > 0 \wedge ran(s?) = -1)$$

$$PostOutdownrequest = (door' = open \wedge upset' = upset \wedge downset' = downset) \vee (downset' = downset \wedge upset' = upset \cup \{dom(s?)\} \wedge door' = door)$$

· 对于内申请操作

$$PreInrequest = (s? \notin upset \wedge s? \notin downset \wedge s? - story \neq 0)$$

$$PostInrequest = downset' = downset \wedge upset' = upset \cup \{s?\}) \vee (downset' = downset \cup \{s\} \wedge upset' = upset)) \wedge tag' = cal(tag, upset', downset')$$

· 对于电梯向上运行操作

$$PreUprun = (door = close \wedge tag = up \wedge min(upset) - story \geqslant 1)$$

$$PostUprun = (story' = add(story) \wedge upset' = upset \wedge downset' = downset \wedge tag' = tag \wedge door' = door) \vee (tag' = cal(tag, upset', downset') \wedge downset' = downset \wedge upset' = upset \backslash min(upset) \wedge story' = add(story) \wedge door' = open)$$

· 对于电梯向下运行操作

$$PreDownrun = (door = close \wedge tag = down \wedge max(downset) - story \leqslant 1)$$

$$PostDownrun = (story' = dec(story) \wedge upset' = upset \wedge downset' = downset \wedge tag' = tag \wedge door' = door) \vee (tag' = cal(tag, upset', downset') \wedge upset' = upset \wedge downset' = downset \backslash max(downset) \wedge story' = dec(story) \wedge door' = open)$$

· 对于电梯进出人员操作

$$PreAdd\_dec\_man = (number + n1? - n2! \leqslant maxsize \wedge door = open)$$

$$PostAdd\_dec\_man = door' = close \wedge number' = number + n1? - n2!$$

上面这些解释类似，它们满足设计需求。

4.2.3 *推理类中的性质*

可以根据扩展的 W 逻辑[2]推理出所书写的形式规格说明的一些性质，这些推理出的性质可以和相应的需求相比较，检验是否满足相应的条件。此节只验证无名状态模式和电梯进出人操作属性，其余类似。

· 对于无名状态模式，有

$$Elevator::State\ schema \vdash upset \cap downset = \emptyset$$

表明向上和向下申请集合不相交，这符合我们的要求，即不可能一个申请按键既属于向上申请集合，又属于向下申请集合。

· 对于操作 Add_dec_man，有

$$Elevator::Add\_dec\_man \vdash number + n1? - n2! \leqslant maxsize \wedge door = open \wedge door' = close \wedge number' = number + n1? - n2!$$

表明操作 Add_dec_man，如果此时电梯门为打开状态 $(door = open)$，电梯中原先的人数为进入人数和出人数线性之和，它们要求小于或等于电梯人数的最大容量 maxsize $(number + n1? - n2! \leqslant maxsize)$，此操作可以进行。实施此

操作后，电梯里的总人数为原先的人数与进入人数和出人数线性之和(number′=number+n1? −n2!)，且此时电梯门关闭(door′=close)。

结论 形式化方法能帮助发现其它方法不容易发现的系统描述的不一致、不明确或不完整，并且可以通过推理验证来保证最终的软件产品是否满足这些需求规格。形式化方法可以用于程序的验证，以保证程序的正确性。因此，在开发过程中可以采用形式化验证及时检查出错误。开发过程中尽早地消除错误是改善软件开发过程质量的关键之一。

定理证明技术就是形式化验证的一种方法，它从形式规格说明中推理出所需的性质，检验是否符合需求规格说明。若能推出应有的性质，则说明形式规格说明符合需求；否则，说明形式规格说明不完善。定理证明也是形式化方法的一个重要组成部分，没有证明就没有核心了。

本文给出了基于 Object-Z 定理证明的形式化验证方法，可以从初始状态的存在性、前后置条件和类中属性进行检验，并用 Object-Z 描述了一个电梯操作系统的实例，在此基础上给出了它定理证明的形式化验证。

参 考 文 献

1 Smith G. The Object-Z Specification Language. In: Advan- ces in Formal Methods, Kluwer Academic Publishers, 2000
2 Smith G. Reasoning about Object-Z specification. APSEC, 1995. 489～497
3 Smith G. A fully abstract semantics of classes for Object-Z. Formal Aspects of Computing, 1995, 7(3): 289～313
4 Smith G, Derrick J. Specification, Refinement and Verification of Concurrent Systems-An Integration of Object-Z and CSP. Formal Methods in System Design, 2001, 18(3): 249～284
5 Olderog E -R, Wehrheim H. Specification and (property) inheritance in CSP-OZ. Science of Computer Programming, 2005, 55(1-3): 227～257
6 Hoenicke J, Olderog E -R. CSP-OZ-DC: a combination of specification techniques for processes, data and time. Nordic Journal of Computing, 2002, 9(4): 301～334
7 Mahony B, Dong Jin Song. Blending Object-Z and Timed CSP: An introduction to TCOZ. In: Proceedings of the 20th International Conference on Software Engineering, IEEE, 1998. 95～104
8 Sun Jing, Dong Jin Song. Specifying and Reasoning About Generic Architecture in TCOZ. In: APSEC'02, IEEE, 2002. 405～414
9 Kim Il-Gon, Choi Jin-Young. Formal Verification of PAP and EAP-MD5 Protocols in Wireless Networks: FDR Model Checking. In: Proceedings of the 18th International Conference on Advanced Information Networking and Applications, 2004
10 Lowe G, Roscoe B. Using CSP to Detect Errors in the TMN Protocol. IEEE Transactions on Software Engineering, 1997, 23(10)

(上接第 214 页)

某足球机器人出现极佳的进攻态势，需要立刻发起进攻；己方队形出现致命的防守漏洞，需要尽快调整防守队形；出现违背比赛规则的情况，如本方机器人进入本方球门、或禁区的机器人个数多余规定，为避免被判罚点球需要立刻离开禁区等等。

形象模块的输出信息包含两类信息：一是当进攻优先度 T_6 或防守优先度 T_7 等于“阻止”或“高”值时，将启动图 2 中的模块③，直接进入行为选择模块，利用矩阵 I、T 等信息，计算相关足球机器人的左右轮速；二是启动图 2 中的模块④，利用矩阵 I、T 等信息，并计算剩余足球机器人(如果还存在的话)的左右轮速。需要注意，模块③、④是并行的，并且对任何一个足球机器人的左右轮速值，只能在也只能在其中一个模块中被产生。

2.3.3 行为选择模块

动作是机器人足球比赛的基本单元，任何精彩激烈的比赛都是由足球机器人一系列动作构成的，足球机器人的动作可以划分为四个层次：基本动作、技术动作、战术动作和组合动作[7]。其中，基本动作负责机器人运动指令与轮速指令之间的转换，实现了机器人的基本功能，技术动作是以基本动作为基础的，战术动作又是以技术动为基础的，而组合动作是以技术动作和战术动作为基础的。四类动作的层次分明，按照上述顺序如堆台阶一样，一步一步向上堆，底层的动作是实现上层动作的保证。

定义 5 假设基本动作集合为 B_1={点球、定位球、原地正旋转、原地反旋转、移动、转角、转身、……}，技术动作集合为 B_2={射门、拦截、阻挡、传球、顶球、刨球、扫球、跑位、避障……}，则集合 $A=B_1UB_2$ 所包含的动作就构成了足球机器人的基本行为。

显然，足球机器人的基本行为是原始的、不能再划分的动作。

如何实现足球机器人的行为选择呢？文[3]认为，生命行为选择得适当与否应由行为选择的目标来评价，并且认为行为选择的分级模型是一种比较好的行为选择模型，但它不是严格的分级行为，而是一种基于优先度的行为选择分级模型，即遵循“感知-行为”机制，按照行为优先度来确定行为的实际选择，应付突发事件，从而实现生命体行为的“突变”。

假设集合 $A=\{B_1, B_2, \cdots, B_r\}$，其中 $B_j(j=1,\cdots,r)\in B_1 \cup B_2$，即足球机器人的基本行为，正数 w_i 是集合 B_i 的优先度。则最终选择的行为是建立在 T_6、T_7 总体综合刺激程度(即决定攻或防)基础之上的优先度高的行为，即基本动作，这样就保证了极端情况下，实现了生命体行为的“突变”。

2.3.4 分层递阶控制模块

按照图 2 中模块④，根据矩阵 I、T 信息，并排除模块③中已经选择计算了左右轮速的足球机器人，输出阵形 D_j 以及对应足球机器人的动作。

2.3.5 轮速计算模块

分别根据综合模块③和④对每个足球机器人确定的动作，计算左右轮速值，从而构成了状态矩阵 S。这里需要注意，分别从模块③和④得到的足球机器人轮速值，可以分别发送，以真正达到并行目的。

结论 以文章介绍的智能体混合决策模型，实现了重庆工学院红岩机器人足球队的 FIRA 3×3、5×5 和 RoboCUP 项目的比赛系统中，参加了 2004、2005 年武汉、广州、成都、常州的全国机器人足球比赛，取得了 2 次二等奖和 1 次三等奖，证明该模型是可行和有效的。

由于机器人足球的高实时性，系统执行过程是单线程，不能设置实时评价函数以修正可能出现的错误，这是该模型今后需要解决的在线学习问题。

参 考 文 献

1 吴丽娟，张春晖，徐心和．足球机器人决策系统推理模型．东北大学学报(自然科学版)，2001，22(6)：597～599
2 班晓娟，王昭顺，刘宏伟，涂序彦．人工智能与人工生命．计算机工程与应用，2002，15：1～3
3 李祖枢，涂亚庆著．仿人智能控制．国防工业出版社，2003．27～53
4 李祖枢，等．人工生命体行为选择及其进化研究．模式识别与人工智能，2005，18(3)：303～309
5 章苏书，等．一种基于行为的多智能体协作策略设计．机器人技术与应用，2003，5：42～44
6 贾建强，等．基于有限状态机的足球机器人行为设计与综合．高技术通讯，2004，4：61～65
7 Zhang Xiao Chuan. APPLY on Action Selection Model of Soccer Robot based on the Precedence Degree. ICMIT 2005 (Chong Qing), 2005