基于 NuSMV 的 LD 和 ST 语言形式化验证研究与实现*

郭肖旺[1,2],赵德政[1,2]
(1.中国电子信息产业集团有限公司第六研究所,北京 100083;2.中电智能科技有限公司,北京 102209)

摘 要:依据工控系统的特点,在分析现有工控系统编程标准 IEC61131-3 规定的工业语言基础上,研究基于工业语言的形式化验证方法,通过对 ST 和 LD 语言进行分析得到有限状态机组态模型,实现对控制目标进行准确描述;通过 NuSMV 验证有限状态机模型,获得形式化验证的结果,从而实现对 IEC61131-3 编程语言实现的 PLC 逻辑代码进行分析,建立形式化验证模型,发现用户编写的 PLC 逻辑代码可能存在的逻辑缺陷,并提供对这些缺陷分析验证的报告。

关键词:工控系统;编译;形式化验证;有限状态机;建模
中图分类号:TP314 **文献标识码**:A DOI:10.16157/j.issn.0258-7998.212293

中文引用格式:郭肖旺,赵德政. 基于 NuSMV 的 LD 和 ST 语言形式化验证研究与实现[J].电子技术应用,2022,48(12):94-99.
英文引用格式:Guo Xiaowang,Zhao Dezheng. Research and implementation of formal verification of LD and ST language based on NuSMV[J]. Application of Electronic Technique,2022,48(12):94-99.

Research and implementation of formal verification of LD and ST language based on NuSMV

Guo Xiaowang[1,2],Zhao Dezheng[1,2]
(1.The Sixth Research Institute of China Electronics Corporation,Beijing 100083,China;
2.Intelligence Technology of CEC Co.,Ltd.,Beijing 102209,China)

Abstract:According to the characteristics of industrial control system, based on the analysis of the existing industrial control system programming standard IEC61131-3 stipulated industrial language, this paper studies the formal verification method based on industrial language, analyzes the ST and LD languages, and gets the finite state model of the machine to achieve accurate description of the control objectives. The finite state machine model is verified by NuSMV, and the result of formal verification is gotten. In this way, the PLC logic code of IEC61131-3 programming language is analyzed, the formal verification model is established, and the possible logic defects of PLC logic code written by users are found, and the report of analysis and verification of these defects is provided.

Key words:industrial control system;compile;formal verification;finite state machine;modeling

0 引言

工控系统具有一次启动长期运行的特点,在现场调试完成之后,不会再频繁修改下装逻辑,控制目标具有持续性。根据 IEC 的最佳实践标准,形式化验证技术是开发安全攸关系统应用时被强烈推荐使用的技术之一[1]。文献[2]对利用形式化验证对智能合约设计和代码实现的过程中存在缺陷进行了分析;文献[3]提出一种基于 SysML 状态机图子集的机载软件分层精化建模与验证方法;文献[4]面向 PLC 提出支持精化关系的形式化语言,提出工业控制领域相关的建模及验证方法;文献[5]将控制系统的运行过程描述为基于状态转移图的自动机中间模型;文献[6]设计了一种基于 FBD 语言的形式化验证方法,采用比 PLC 测试或仿真更好的形式化方法,利用它的遍历性可以全面地描述到系统的状态,而且能生成不满足性质的反例路径;文献[7]设计 ST 的形式化验证方法,定义了一个形式化模型来描述其运行时的行为,并给出了该模型上的 LTL 验证方法,借助 ST 程序的形式化操作语义和加权下推系统,实现了形式化建模过程的自动化。依据工控系统的特点,本文在分析现有工控系统编程标准 IEC61131-3 规定的工业语言基础上,研究基于工业语言的图形化建模方法,实现对控制目标的形式化准确描述。

*基金项目:国防基础科研计划资助项目(JCKY2018211C001)

通过分析 PLC 的 5 种编程语言，给出对 ST 语言和 LD 语言解析生成建立状态机图的方法分析，通过分析 ST 和 LD 语言编写的控制程序，生成对应的状态机图，并基于生成的状态机图设计生成形式化验证模型的方法，生成验证模型，再通过 NuSMV[8](新符号模型检查器)分析验证模型。

1 形式化验证技术分析

NuSMV 是经典的模型检测工具，它高效地实现了符号模型检测技术，是实现形式化验证的重要工具。文献[9]设计并实现了一套基于 NuSMV 的安全苛求软件需求 Simulink 模型形式化验证方法；文献[10]提出基于 NuSMV 的 AADL 模型形式化验证方法；文献[11]给出一套从 SysML 设计模型到 NuSMV 模型转换的语义规则。本项目采用 NuSMV 模型检测工具实现形式化验证方案。

NuSMV 是 SMV 的重新实现和扩展，SMV 是第一个基于 BDD 的模型检查器。NuSMV 被设计成一个开放的模型检查体系结构，可以可靠地用于工业设计的验证，作为自定义验证工具的核心，作为形式化验证技术的测试平台，并应用于其他研究领域。

本文设计 ST 语言和 LD 语言解析生成建立状态机图的方法分析，如图 1 所示，通过分析 ST 和 LD 语言编写的控制程序，生成对应的状态机图，并基于生成的状态机图设计生成形式化验证模型的方法，生成验证模型，再通过 NuSMV 分析验证模型。

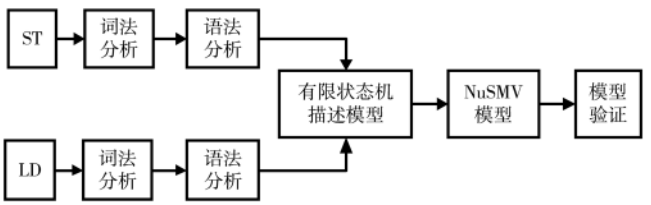

图 1 支持的语言和模型验证过程方案

对 ST 语言进行词法分析、语法分析，对抽象语法树进行分析，产生有限状态机模型，有限状态机组态的逻辑经过转化后生成 NuSMV 建模语言描述模型，调用 NuSMV 接口进行模型验证，输出模型验证结果。同时，ST 编译后生成目标运行指令，可下载到控制器运行。

对 LD 语言进行词法分析、语法分析，对抽象语法树进行分析，产生有限状态机模型，有限状态机组态的逻辑经过转化后生成 NuSMV 建模语言描述模型，调用 NuSMV 接口进行模型验证，输出模型验证结果。同时，LD 编译后生成目标运行指令，可下载到控制器运行。

2 基于 ST 的有限状态机图生成模型

对 ST 语言进行词法分析、语法分析，生成抽象语法树，根据抽象语法树遍历所有语句，生成状态机。ST 语句的状态主要考虑简单运算语句、调用语句、IF、WHILE、CASE、JMP、RETURN 等会影响执行控制流的语句，本文对 ST 语言中的简单运算语句、调用语句、IF、WHILE、CASE、JMP、RETURN 关键字的语句建立状态，并建立状态机转换图。

ST 的语句集合定义如下：

$$\text{Statements}=\{S_{\text{Simple}},S_{\text{Call}},S_{\text{IF}},S_{\text{WHILE}},S_{\text{CASE}},S_{\text{JMP}},S_{\text{RETURN}}\} \tag{1}$$

变量集合定义：

$$\text{Variables}=\{p_1,p_2,p_3,\cdots,p_n\} \tag{2}$$

对于任意一个变量，可以根据变量得到不同的状态集合，则 StateV 集合下有不同变量的状态子集：

$$\text{StateV}=\{\text{StateV}_{p1},\text{StateV}_{p2},\cdots,\text{StateV}_{pn}\}$$

$$\forall p_i:\text{Variables}\Rightarrow\text{StateV}p_i\in\text{StateV} \tag{3}$$

$$\text{StateV}_{p_i}=\{\text{StateV}_{p_{i0}},\text{StateV}_{p_{i1}},\cdots,\text{StateV}_{p_{im}}\} \tag{4}$$

由于 IF、WHILE、CASE、JMP、RETURN 等语句会引起流程的变化，因此根据语句不同会有不同的执行条件，条件定义集合如下：

$$\text{Conditions}=\{C_0,C_1,\cdots,C_n\} \tag{5}$$

下面对 ST 的几种语句进行推导：

(1)顺序执行语句

顺序执行的语句定义为 S_{Simple}，不影响流程的顺序执行，每执行完一句顺序执行下一句。

则对于任意 S_{Simple} 语句，每一个语句执行前为一个状态 StateV_{pim}，执行后为下一个状态 StateV_{pim+1}。

$$\forall S\in S_{\text{Simple}}\Rightarrow\exists p_i\in\text{Variables}\ \wedge\ \text{StateV}_{pim}\in\text{StateV}_{pi}\ \wedge\ \text{StateV}_{pim+1}\in\text{StateV}_{pi} \tag{6}$$

(2)调用语句

调用语句的语句定义为 S_{call}，调用语句包含调用函数和子程序，在函数和子程序中可能会改变变量的值，因此，在调用之前状态为 StateV_{pim}，为不影响流程的顺序执行，每执行完一句顺序执行下一句。

则对于任意 S_{Simple} 语句，每一个语句执行前为一个状态 StateV_{pim}，执行后为下一个状态 StateV_{pim+1}。

$$\forall S\in S_{\text{call}}\Rightarrow\exists p_i\in\text{Variables}\ \wedge\ \text{StateV}_{pim}\in\text{StateV}_{pi}\ \wedge\ \text{StateV}_{pim+1}\in\text{StateV}_{pi} \tag{7}$$

(3)IF 语句

IF 语句的语句定义为 S_{IF}，假设进入 IF 之前设置一个状态 StateV_{pim}，进入 IF 语句，根据 IF 的条件设置运行状态，假设条件为 C_m，则条件满足后状态为 StateV_{pim+1}，条件不满足，则状态为 StateV_{pim+2}，则：

$$\forall S\in S_{\text{IF}}\Rightarrow(C_m\wedge\text{StateV}_{pim}\rightarrow\text{StateV}_{pim+1})\vee(\neg C_m\wedge\text{StateV}_{pim}\rightarrow\text{StateV}_{pim+2}) \tag{8}$$

$$\begin{cases}C_m\in\text{Conditions}\\ \text{StateV}_{pim}\in\text{StateV}_{pi}\\ \text{StateV}_{pim+1}\in\text{StateV}_{pi}\\ \text{StateV}_{pim+2}\in\text{StateV}_{pi}\end{cases} \tag{9}$$

(4)WHILE 语句

WHILE 语句的语句定义为 S_{WHILE}，假设进入 WHILE 语句之前是状态 StateV_{pim}，WHILE 语句的条件是 C_m，则条件

满足后状态为 $StateV_{pim+1}$，条件不满足，则状态为 $StateV_{pim+2}$，且由于 WHIE 的条件 C_m 满足，会一直在状态 $StateV_{pim+1}$ 循环执行该循环体中的语句，则：

$$\forall S \in S_{WHILE} \Rightarrow (C_m \wedge StateV_{pim} \rightarrow StateV_{pim+1}) \vee (\neg C_m \wedge StateV_{pim} \rightarrow StateV_{pim+2}) \tag{10}$$

$$\begin{cases} C_m \in Conditions \\ StateV_{pim} \in StateV_{pi} \\ StateV_{pim+1} \in StateV_{pi} \\ StateV_{pim+2} \in StateV_{pi} \end{cases} \tag{11}$$

如下实例的 ST 语句，生成的状态机图如图 2 所示。

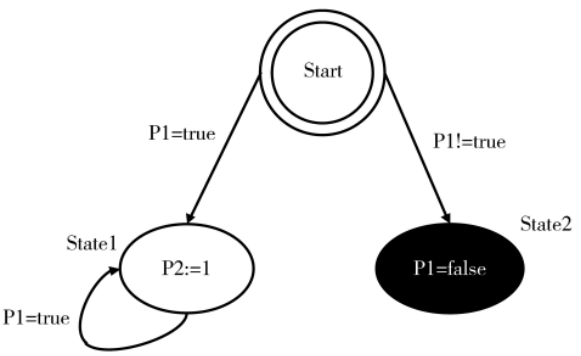

图 2 ST 语言 WHILE 语句的状态机

```
WHILE p1 = true DO
    p2:=1;
END_WHILE
```

(5)CASE 语句

CASE 语句的语句定义为 S_{CASE}，对于 CASE 语句，可存在多个条件，每一个条件下需要执行不同的 ST 语句。假设进入 CASE 语句之前是状态 $StateV_{pim}$，CASE 语句的条件定义为 $C_m, C_{m+1}, \cdots, C_{m+i}$，每个条件下执行相关的语句后为状态 $StateV_{pim+1}, StateV_{pim+2}, \cdots, StateV_{pim+i+1}$，则：

$$\forall S \in S_{CASE} \Rightarrow (C_m \wedge StateV_{pim} \rightarrow StateV_{pim+1}) \vee (C_{m+1} \wedge StateV_{pim} \rightarrow StateV_{pim+2}) \cdots \vee (C_{m+i} \wedge StateV_{pim} \rightarrow StateV_{pim+i+1}) \tag{12}$$

$$\begin{cases} C_m \in Conditions \\ C_{m+1} \in Conditions \\ \cdots \\ C_{m+i} \in Conditions \\ StateV_{pim} \in StateV_{pi} \\ StateV_{pim+1} \in StateV_{pi} \\ StateV_{pim+2} \in StateV_{pi} \end{cases} \tag{13}$$

(6)JMP 语句

对于 JMP 语句，在 ST 语句，JMP 语句的作用与 C 语言的 Goto 语句一样，是跳转到指定位置执行的语句，会改变顺序执行的控制流。JMP 语句的定义为 SJMP，假设进入 JMP 语句之前是状态 $StateV_{pim}$，根据 JMP 跳转执行跳转到的状态为 $StateV_{pin}$，则有如下公式：

$$\forall S \in S_{JMP} \wedge StateV_{pim} \rightarrow StateV_{pin} \tag{14}$$

(7)RETURN 语句

对于 RETURN 语句，认为 RETURN 之前为一个状态，之后的语句为一个状态，RETURN 语句的定义为 S_{RETURN}，假设进入 RETURN 语句之前是状态 $StateV_{pim}$，该语句之后的语句不会被执行，则状态机运行到改部为结束，则有如下公式：

$$\forall S \in S_{RETURN} \wedge StateV_{pim} \rightarrow StateV_{pim} \tag{15}$$

3 基于 LD 的有限状态机图生成模型

LD 语言是 PLC 程序设计中最常用的编程语言，具有直观性、形象性及实用性，与电气操作原理图相对应。LD 语言的主要元件是：电源导轨、线圈、触点、块。按照 IEC61131-3 标准，元件可以进行串联和并联。本文针对 LD 语言执行过程分析，总结出状态转换的方法模型，LD 语言为从左到右、从上到下的执行过程，则从左电源导轨开始，每一个元件前的连接线为一个状态，左电源导轨为起始状态，右电源导轨为结束状态，每个触点、线圈或块的 ENENO 引脚是一个转换条件。

对 LD 语言进行解析，生成状态机。LD 的语句集合定义如下：

$$Statements = \{S_{LeftRail}, S_{contact}, S_{Coil}, S_{Block}\} \tag{16}$$

每运行能流线上的一个元件为一个状态，则有如下状态定义：

$$States = \{State_{Start}, State_1, \cdots, State_n, State_{End}\} \tag{17}$$

LD 语言为从左到右、从上到下执行，能流线导通根据元件的状态来判断，因此能流线上元件值为条件，有如下条件定义：

$$Conditions = \{C_0, C_1, \cdots, C_n\} \tag{18}$$

则根据 LD 语言的元件类型和运行过程有如下公式：

(1)左电源导轨为起始状态：

$$\forall S : S_{LeftRail} \rightarrow State_{Start} \tag{19}$$

(2)触点分为常开触点、常闭触点、上升沿触发触点、下降沿触发触点，常开触点关联变量为 TRUE 时，能流线导通：

$$\forall S : S_{Contact} \Rightarrow (C_m = (Variable_{Contact} = true)) \tag{20}$$

$$\begin{cases} C_m \rightarrow StateV_{pim} \\ \exists C_m \in Conditons \end{cases} \tag{21}$$

常闭触点关联变量为 TRUE 时，能流线导通：

$$\forall S : S_{Contact} \Rightarrow (C_m = (Variable_{Contact} = false)) \tag{22}$$

$$\begin{cases} C_m \rightarrow StateV_{pim} \\ \exists C_m \in Conditons \end{cases} \tag{23}$$

上升沿触发触点关联变量上一周期为 FALSE，本周期为 TRUE 时，能流线导通：

$$\forall S : S_{Contact} \Rightarrow (C_m = (Variable_{Contact-1} = false \wedge Variable_{Contact} = true)) \tag{24}$$

$$\begin{cases} C_m \rightarrow StateV_{pim} \\ \exists C_m \in Conditons \end{cases} \tag{25}$$

下降沿触发触点关联变量上一周期为 TRUE，本周期为 FALSE 时，能流线导通：

$$\forall S: S_{\text{Contact}} \Rightarrow$$

$$(C_m=(\text{Variable}_{\text{Contact}-1}=\text{true} \wedge \text{Variable}_{\text{Contact}}=\text{false}) \tag{26}$$

$$\begin{cases} C_m \rightarrow \text{StateV}_{pim} \\ \exists\, C_m \in \text{Conditons} \end{cases} \tag{27}$$

(3)线圈，总是将线圈左侧的状态赋值给右侧的状态，因此不管线圈是哪种类型，能流线都导通：

$$\forall S: S_{\text{Coil}} \Rightarrow \text{StateV}_{pim} \rightarrow \text{StateV}_{pim+1} \tag{28}$$

(4)块，块连接在能流线上的引脚必须是 BOOL 类型或使用 EN、ENO 引脚，ENO 引脚作为块的右侧输出引脚，因此块需判断右侧输出引脚的值是否为 true，则有：

$$\forall S: S_{\text{Block}} \Rightarrow C_m = S_{\text{Block}} \cdot \text{out1} \tag{29}$$

$$\begin{cases} C_m \rightarrow \text{StateV}_{pim+1} \\ \neg C_m \in \text{StateV}_{pim} \end{cases} \tag{30}$$

4 有限状态机图生成 NuSMV 模型

有限状态机图图形向 NuSMV 转换步骤如图 3 所示，具体如下：

(1)获取状态机：读取 ST 或 LD 转成的状态机图；

(2)判断起始状态是否存在：如果起始状态不存在则报错；

(3)从起始状态后获取所有状态和转换条件；

(4)将遍历到的第一个状态设置为当前状态；

(5)查找该状态起始的转移条件；

(6)生成状态判断语句；

(7)生成转移语句；

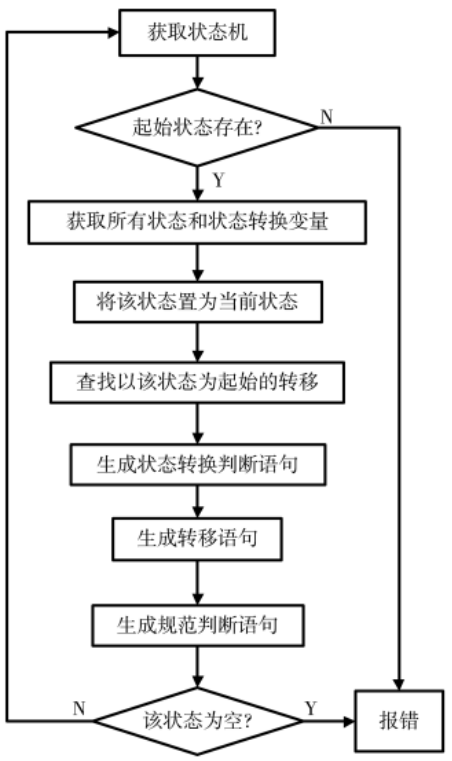

图 3 基于有限状态机图向 NuSMV 模型转换流程

(8)生成规范判定语句；

(9)遍历后继状态，如果后继状态为空则结束，否则继续转向步骤(4)。

按照以上步骤遍历状态机图，获得状态、转移条件、判定规范等语句，根据 NuSMV 有限状态机模型生成规则，生成 NuSMV 验证模型。

本文给出基于 LD 实例生成的状态机图，LD 实例程序如图 4 所示，根据 LD 语言的有限状态机生成模型，得到状态机图如图 5 所示。如图 5 所示 LD 语言的图形，在左母线处生成状态 0，在右母线结束处生成结束状态。

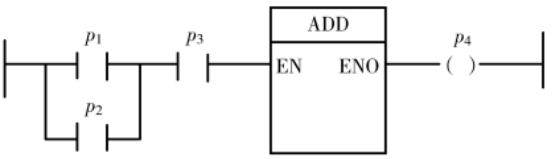

图 4 LD 语言程序示例

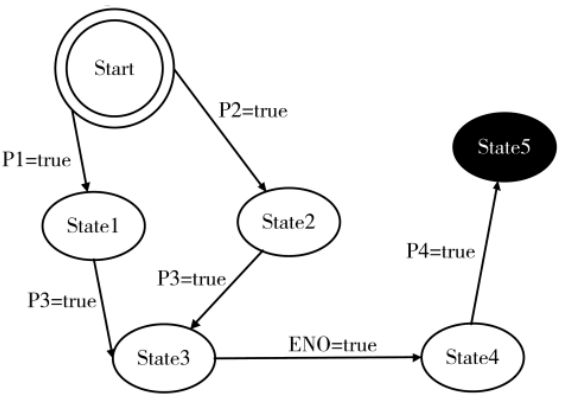

图 5 LD 语言程序转换状态机图实例

其中 p_1、p_2、p_3、p_4 为变量，用 Tmp 当作 ADD 块 EN ENO 引脚的临时变量。根据 NuSMV 模型定义如下变量：

```
VAR
  p1 : boolean ;
  p2 : boolean ;
  p3 : boolean ;
  p4 : boolean ;
  Tmp : boolean ;
```

状态包括 Start、State_1、State_2、State_3、State_4、State_5，其中初始状态为 Start。

根据 NuSMV 模型定义如下状态：

```
State : {Start , State_1 , State_2 , State_3 , State_4 , State_5} ;
```

状态转换条件为 p_1、p_2、p_3、p_4 和 Tmp 变量的值。根据变量的值和状态的转换得到状态转换的处理语句。

```
ASSIGN
  init(State) := Start;
  next(State) :=
        case
    State = Start & p1 : {State_1};
    State = Start & p2 : {State_2};
```

```
    State = State_1 & p3 : {State_3};
    State = State_2 & p3 : {State_3};
    State = State_3 & Tmp: {State_4};
    State = State_4 & p4 : {State_5};
    TRUE : State;
    esac;
next(p1) :=
        case
    State = Start : TRUE;
    State = State_1 : FALSE;
    State = State_2 : FALSE;
    State = State_3 : FALSE;
    State = State_4 : FALSE;
    State = State_5 : FALSE;
    TRUE: p1;
    esac;
    next(p2) :=
        case
    State = Start  : TRUE;
    State = State_1 : FALSE;
    State = State_2 : FALSE;
    State = State_3 : FALSE;
    State = State_4 : FALSE;
    State = State_5 : FALSE;
    TRUE: p2;
    esac;
next(p3) :=
        case
    State = Start : FALSE;
    State = State_1 : TRUE;
    State = State_2 : TRUE;
    State = State_3 : FALSE;
    State = State_4 : FALSE;
    State = State_5 : FALSE;
    TRUE: p3;
    esac;
next(Tmp) :=
        case
    State = Start : FALSE;
    State = State_1 : FALSE;
    State = State_2 : FALSE;
    State = State_3 : TRUE;
    State = State_4 : FALSE;
    State = State_5 : FALSE;
    TRUE: Tmp;
    esac;
next(p4) :=
        case
    State = Start : FALSE;
    State = State_1 : FALSE;
    State = State_2 : FALSE;
    State = State_3 : FALSE;
    State = State_4 : TRUE;
    State = State_5 : FALSE;
    TRUE: p4;
    esac;
```

验证规格可根据状态的变化设置，如设置如下规则：

```
SPEC
    AG(State = Start -> AF State = State_5)
```

调用 NuSMV 验证模型，得到结果如图 6 所示，说明经过变量赋值和状态转换，从 Start 到状态 State_5 是可达的。

```
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:37:51 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification AG (State = Start -> AF State = State_5)  is true
```

图 6　AG(State = Start -> AF State = State_5)结果

如果验证规则改为 SPEC AG(State=Start->AF State=State_1)，调用 NuSMV 验证，则结果为 FALSE，如图 7 所示。从图上可以看到，从状态 Start 开始，先给 p2 赋值 TRUE，则状态 Start 的下一个状态为 State_2，再给 p3 赋值 TRUE，则状态转向 State_3，在这个转换路径上即使再

```
-- specification AG (State = Start -> AF State = State_1)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    p1 = FALSE
    p2 = TRUE
    p3 = FALSE
    p4 = FALSE
    Tmp = FALSE
    State = Start
  -> State: 1.2 <-
    p1 = TRUE
    State = State_2
  -> State: 1.3 <-
    p1 = FALSE
    p2 = FALSE
    p3 = TRUE
  -> State: 1.4 <-
    State = State_3
  -> State: 1.5 <-
    p3 = FALSE
    Tmp = TRUE
  -> State: 1.6 <-
    State = State_4
  -> State: 1.7 <-
    p4 = TRUE
    Tmp = FALSE
  -> State: 1.8 <-
    State = State_5
  -- Loop starts here
  -> State: 1.9 <-
    p4 = FALSE
  -> State: 1.10 <-
```

图 7　AG(State=Start -> AF State=State_1)结果

给 p1 赋值为 TRUE，也不能再达到状态 State_1 了。

5 结论

本文完成对 ST 语言和 LD 语言生成有限状态机图的分析，在语言分析的基础上基于有限状态机实现了模型描述语言和模型检测的研究。在完成图形化建模编程后，可以在编译阶段对建立的模型进行检查，对模型中的不一致、类型不匹配、数据回路等所有的模型错误进行定位，以便实现对工程逻辑的纠错。

参考文献

[1] ISO/IEC.ISO/IEC 27000 family-information security management systems[S].2018.

[2] 张文博，陈思敏，魏立斐，等.基于形式化方法的智能合约验证研究综述[J].网络与信息安全学报，2022，8(4)：12-28.

[3] 肖思慧，刘琦，黄滟鸿，等.基于 SysML 的机载软件分层精化建模与验证方法[J].软件学报，2022，33(8)：2851-2874.

[4] 毛侠.面向可编程逻辑控制器的功能安全形式化建模与验证方法研究[D].上海：华东师范大学，2022.

[5] 范宝文，包健.嵌入式控制系统时序安全性建模及验证[J].杭州电子科技大学学报(自然科学版)，2021，41(6)：21-27.

[6] 李建龙.PLC 系统及其 FBD 编程语言的形式化建模与实时性验证[D].泉州：华侨大学，2015.

[7] 卜祥兴.符合工控 IEC 61131-3 国际标准的结构化文本程序的验证方法研究[D].上海：华东师范大学，2018.

[8] CIMATTI A，CLARKE E，GIUNCHIGLIA E，et al.NuSMV 2：an opensource tool for symbolic model checking[C]//International Conference on Computer Aided Verification，2002：359-364.

[9] 李乾旭.基于 NuSMV 的 XX 软件需求模型转换与形式化验证[D].北京：北京交通大学，2021.

[10] 刘畅，蒋永平，马春燕，等.基于 NuSMV 的 AADL 模型形式化验证技术[J].航空学报，2022，43(3)：451-466.

[11] 邓刘梦，葛晓瑜，宛伟健.基于 NuSMV 的 SysML 模型形式化验证[J].计算机技术与发展，2019，29(10)：153-156.

(收稿日期：2021-11-04)

作者简介：

郭肖旺(1986-)，女，硕士，高级工程师，主要研究方向：工业软件、编译技术。

赵德政(1985-)，男，博士，研究员，主要研究方向：工业自动化控制、工控安全。

扫码下载电子文档

(上接第 93 页)

注入攻击检测[J].中国安全生产科学技术，2021，17(3)：124-129.

[8] 阮兆文，孟干，周冬青，等.智能电网中的虚假数据注入攻击检测方法研究[J].自动化与仪器仪表，2019(3)：49-52.

[9] 彭华晔，彭晨，孙洪涛，等.微电网在虚假数据注入攻击下的增量检测机制[J].信息与控制，2019，48(5)：522-527.

[10] HAO J，KANG E，SUN J，et al.An adaptive Markov strategy for defending smart grid false data injection from malicious attackers[J].IEEE Transactions on Smart Grid，2016，4(6)：1-3.

[11] YANG X，ZHAO P，ZHANG X，et al.A Gaussian-mixture model based detection scheme against data integrity attacks in the smart grid[J].IEEE Internet of Things Journal，2016，82(5)：1-6.

[12] WANG H，RUAN J，WANG G，et al.Deep learning-based interval state estimation of AC smart grids against sparse cyber attacks[J].IEEE Transactions on Industrial Informatics，2018，56(2)：82-86.

[13] 张逸为，许德智，杨玮林，等.含混合储能的互联电力系统传感器容错负荷频率控制[J].控制与决策，2021，36(5)：9-10.

[14] 安娜.微电网模式切换控制策略研究[D].秦皇岛：燕山大学，2014.

[15] 游国栋，李继生，侯勇，等.光伏 LCL 型并网逆变器的积分滑模容错控制策略[J].太阳能学报，2018，39(4)：1008-1017.

(收稿日期：2021-11-19)

作者简介：

王君(1973-)，女，博士，教授，主要研究方向：动态系统故障诊断与容错控制。

冯甜(1993-)，通信作者，女，硕士，主要研究方向：动态系统故障诊断与容错控制，E-mail：2691978497@qq.com。

扫码下载电子文档

版权声明