基于符号约束的PLC程序正确性验证

张 晔 陆余良
(电子工程学院 合肥 230037)

摘 要 符号约束描述了程序中的变量关系,被广泛运用于模型检测、符号执行等程序的静态分析方法中。将符号约束应用于可编程逻辑控制器(PLC)程序的正确性验证,能够发现程序中的逻辑错误。人工计算符号约束不仅冗杂枯燥,而且错误率高。针对语句表形式的PLC程序,提出一种基于符号约束的正确性验证方法,通过分析PLC源代码的控制流及数据流,构造程序的控制流图并将其转换为静态单赋值形式的三地址码,最后使用迭代计算的方法求出每个变量的符号约束。

关键词 可编程逻辑控制器,符号约束,正确性验证,三地址码,静态单赋值
中图法分类号 TP312 **文献标识码** A

Correctness Verification of PLC Program Based on Symbolic Constriant

ZHANG Ye LU Yu-liang
(Electronic Engineering Institute of PLA, Hefei 230037, China)

Abstract Symbolic constraints describe the relations among variables of the program, and they are widely used in static program analysis methods such as model checking and symbol execution. The logic error in the program could be found while applying symbolic constraints to the correctness verification of the programmable logic controller (PLC) program. However, the process of manually computing symbolic constraints is not only cumbersome and boring, but also with high error rates. Aiming at the PLC program in the form of statement list (STL), a verification method based on generating symbol constraint automatic was proposed. By analyzing the control flow and data flow of the PLC source code, the control flow graph of the program is constructed and the source code is converted into three address code in the form of static single assignment. Finally, the iterative method is used to find the symbol constraint of each variable.

Keywords Programmable logic controller, Symbolic constraints, Correctness verification, Three address code, Static single assignment

1 引言

可编程逻辑控制器(Programmable Logic Controllers, PLC)广泛应用于工业控制领域,无论是软件还是硬件,任何细微的错误都有可能造成巨大的人身、财产损失,因此对其安全性要求较高。正因如此,针对PLC程序的安全性分析及测试引起了研究人员的重视。

目前,国内对PLC程序分析的研究还属于发展的起步阶段,主要集中于形式化验证方法。陈钢等[1]利用定理证明器COQ验证了一个PLC抢答器程序,证明了定理证明器应用于PLC程序形式化验证的可行性。肖力田等[2]定义了PLC程序语言的指称语义,为模型检测和定理证明提供了基础。陈雪琨[3]利用Petri网对PLC程序进行形式化建模并提出了基于网模型的竞争状态检测方法。

近年来,国外已有的研究成果将模型检测、符号执行、定理证明等诸多的形式化分析方法以及静态分析方法应用于PLC程序的安全分析中。

模型检测主要用于验证程序有穷状态是否满足命题性质,Biallas等[4]基于抽象解释理论、反例引导的抽象求精算法设计了具备模型检测和静态分析功能的形式化验证平台Arcade.PLC,它能够兼容IEC 61131-3标准中的所有语言以及西门子的STL及ST语言。CERN基于NuSMV和nuXmv模型检测引擎开发的模型检测工具PLCverif[5]还支持了西门子的SFC和SCL等高级语言,但该工具还处于概念验证阶段。Adiego等[6]提出如何将模型检测的方法应用于工业级大小的PLC程序中。

符号执行是一种用符号值程序中的变量具体值执行程序的技术。McLaughlin等[7]利用符号执行的方法分析程序的路径约束,通过约束求解计算满足目标约束的具体值,消除了不可达路径的访问,有效解决了PLC程序模型检测中的状态爆炸问题。

定理证明首先将程序和规范均转换为逻辑形式的模型,然后利用数学定理推导证明程序的模型是否等价于或蕴含规范的模型。Biha[8]定义了PLC指令表程序指令集的部分形式化操作语义,并利用COQ定理证明器证明其是否满足某些安全属性。Blech等[9]在其基础上将该方法拓展到LD,SFC,FBD等PLC程序编程语言中。

当前针对PLC程序的静态分析利用了控制流、数据流分析等技术,主要应用于程序语法规范性的验证。

以上研究成果大多都是先为程序的安全缺陷属性建模,

张 晔(1993-),男,硕士生,主要研究方向为信息安全,E-mail:zhangye_only@foxmail.com;陆余良(1964-),男,博士,教授,主要研究方向为信息安全。

再利用相应的工具证明程序模型是否满足该属性。例如,模型检测通常将程序中需要检测的安全缺陷属性转换为线性时序逻辑、分支时序逻辑等;定理证明则将程序的安全缺陷属性转换为命题逻辑、布尔公式等定理证明器可接受的规范形式。这些方法中的核心问题即为程序的安全属性对应的形式化模型该如何生成。形式化模型生成的正确性、覆盖面都将极大地影响安全性验证的效果。不仅如此,安全属性的建模还耗费大量人力,过程冗杂枯燥。

针对上述不足,文中提出一种用符号约束检验 PLC 程序安全性的方法。利用词法分析、语法分析将语句表形式的 PLC 程序转化为三地址码,进一步生成程序的控制流图(CFG)。在遍历控制流图的基础上将三地址码转换为静态单赋值(SSA)形式,通过迭代计算静态单赋值形式的三地址码自动生成程序变量的符号约束,最后人工判断符号约束是否违背安全属性。与形式化方法相比,该方法虽然需要人为判断符号约束是否违背安全规范,但是省略了安全属性人工建模的步骤,能够遍历所有程序路径,以达到验证程序安全性的目的。

2　预备知识

2.1　语句表

IEC 61131-3[10]标准定义了 5 类 PLC 程序语言,分别是指令表(IL)、结构文本(ST)、梯形图(LD)、功能块图(FBD)和顺序功能图(SFC)。其中,指令表是一种类汇编的低级语言。本文的研究对象语句表(Statement List,STL)演化于指令表,专门应用于西门子 PLC,它在语义上等价于指令表,但在语法上却存在一定的区别。

语句表程序结构分为 3 部分:1)程序声明;2)变量声明;3)由指令序列组成的程序体。程序声明部分包括块类型、块名称、标题、作者、版本等配置信息;变量声明部分声明了所有引用的变量,不仅包括输入变量、输出变量、输入输出变量等接口变量,还包括临时变量和静态变量;程序体由多条指令组成,每条指令包含一个操作符和至多一个操作数。程序以确定的顺序先后执行这些指令,其控制流由逻辑控制指令和标签决定。

2.2　符号约束

符号约束长期被用于验证程序的正确性,它通过符号值描述变量之间的关系而不需要考虑变量的具体值,与形式化方法中的安全属性功能相同,区别在于二者使用的语法不同,例如,$z=y+z$。因为接口变量才与 PLC 的输出模块相关联,所以在生成语句表程序的符号约束后,左值(LHS)为输出变量、输入输出变量两类接口变量的约束可以作为程序正确性的重要判定依据。在某些情况下,也可以对左值为静态变量或临时变量的约束进行分析来达到辅助验证程序正确性的目的。

通常情况下,采用符号执行技术生成变量的符号约束。本文基于语句表语言的特性提出了另一种方法,即通过迭代计算为所有变量生成由符号和常量组成的表达式。

3　方法实现

利用符号约束验证程序的正确性主要解决的问题是:在给定程序源代码的条件下,如何遍历所有的路径生成程序左值为输出变量的符号约束,通过比较符号约束与程序所需表征的语义判定程序的正确性。

符号约束的自动生成主要分为 4 个部分,具体如图 1 所示。首先,需要将语句表程序转换为三地址码形式的中间语言;其次,基于中间语言构造程序的控制流图;之后,遍历控制流图进行数据流分析,生成静态单赋值形式的三地址码;最后,迭代计算各输出变量的符号约束。

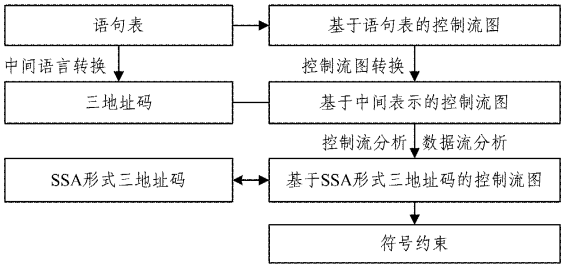

图 1　方法实现原理图

3.1　中间语言转换

低级语言通常都存在指令副作用,例如汇编语言,大部分指令都会对程序状态字产生影响。语句表程序亦是如此,指令副作用的存在使控制流和数据流不够直观,不便于程序的自动化分析,需要转换为中间语言,且中间语言的操作语义需要与源代码保持一致。本文使用三地址码作为中间语言,三地址码有一个操作符,至多 4 个操作数,比较适合语句表的转换。语句表指令的中间语言转换示例如表 1 所列。

表 1　中间语言转换示例

指令类型	实例	三地址码
赋值指令	= Q 0.1	RLO=Q 0.1
装入指令	L IW 0	ACCU2=ACCU1 ACCU1=IW 0
传送指令	T QW 0	QW 0=ACCU1
位逻辑指令	A I 0.1	STA=I 0.1 CJMP FC == 0,L1 RLO=RLO &&. STA JMP L2 L1:RLO=STA L2:FC=1
数学运算指令	+I	ACCU1=ACCU1+ACCU2
比较指令	>I	RLO=ACCU2>ACCU1
无条件跳转指令	JU L1	JMP L1
条件跳转指令	JC L1	CJMP RLO == 1,L1

表 1 中,ACCU1 和 ACCU2 是两个 32 位的累加器,RLO 是逻辑运算结果位,大部分指令通过两个累加器和 RLO 完成执行。因此,在中间语言中累加器和 RLO 位的出现频率十分高,使定值引用、活跃性分析等数据流分析变得更加复杂,静态单赋值形式可以有效解决这个问题。

具体实现中,三地址码采用五元式的数据结构保存在内存中。除跳转指令以外,其他语句表指令均被转换为赋值指令。三地址码的存储结构如表 2 所列。

表 2　三地址码指令结构

指令类型	数据结构
一元运算	(opr,op_dst,op_src1)
二元运算	(opr,op_dst,op_src1,op_src2)
无条件跳转	(opr,lineNo)
条件跳转	(opr,op_src1,op_src2,cmp,lineNo)

其中,opr 表征操作符,op 表征操作数,op_dst 表征目的操作数,op_src 表征源操作数,lineNo 表征行号,cmp 表征比较运算符。在中间语言中,每条指令设立了一个专有的 id 标

识行号，这个 id 用于替换语句表中的标签实现程序的指令跳转。

除此之外，中间语言中的转换还存在几个难题：1）如何处理循环结构。由于语句表的循环次数需要显式地进行定义，因此针对循环结构采用了展开的方式，尽管增加了代码长度，但是能够无损地恢复程序的语义信息。2）定时器、计数器等特殊指令的翻译规则。因为符号约束不考虑时间因素，所以定时器和计数器指令不在考虑范围之内。3）嵌套指令的翻译规则。除了主控继电器指令（MCR）外，语句表有“A(”，“O(”，“X(”，“AN(”，“ON(”，“XN(” 6 类嵌套指令。下面以表 3 中的一个例子阐述嵌套指令的翻译规则。表 3 所列的指令表征的是一个语义为 RLO=I0.1&(I0.2|I0.3)的表达式，鉴于论文篇幅的大小，翻译后的三地址码省略了除 RLO 以外的其余状态位指令。显然，在遇到嵌套指令时，将 RLO 位的值分配给一个新的临时变量，并将操作符“A”和这个临时变量压入栈中，在遇到右括号时，从栈中弹出操作符及临时变量，令 RLO 值与该临时变量根据操作符进行运算，并将运算结果保存至 RLO 中。实际上，压栈时还包含 OR 状态位，在此不详细描述。

表 3 嵌套指令翻译规则

嵌套指令	三地址码
A I 0.1	RLO=I 0.1
A(	T1=RLO;——S=T1
O I 0.2	RLO=I 0.2
O I 0.3	RLO=I 0.2 | I 0.3
)	RLO=S++ & RLO

3.2 静态单赋值形式转换

在一个程序中，同一个变量可能被多次定值。如图 2 所示，RLO 状态位分别在两个基本块中被定值，在两个基本块交汇时被引用。于是在交汇处为 Q0.1 变量生成符号约束时就需要使用定值引用的数据流分析方法来判断具体引用的是之前定义的哪个 RLO 值。由于 RLO 状态位和累加器的定值频率较高，导致 DU 链数目过多，分析效率非常低，因此提出了静态单赋值形式转换来解决该问题。

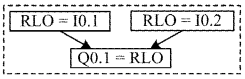

图 2 三地址码

静态单赋值转换为每个变量添加了不同的标号加以区分，并在程序的控制流交汇处插入了 Φ 函数将变量的多个定值合并起来。在 SSA 转换后，每个变量仅被定值一次，大幅减少了 DU 链的数目。将图 2 转换为 SSA 形式后如图 3 所示。

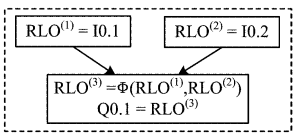

图 3 SSA 形式的三地址码

本文采用了 Cytron 等[11]提出的静态单赋值转换算法，转换过程分为 3 个步骤：1）遍历控制流图分析各节点的支配边界；2）利用支配边界确定各变量 Φ 函数的插入位置；3）在变量 V 的引用处用 V_i 进行替换。具体细节不再赘述。

3.3 迭代计算生成符号约束

符号约束自动生成方法的核心在于遍历控制流图的所有路径，并在每条路径上迭代计算每个变量的符号表达式。

为每个变量引入静态单赋值形式后，程序中有且仅有一个符号表达式左值为该变量。符号表达式的右值通常是由其他变量组成的运算表达式或者常量。例如，$RLO^{(2)}=RLO^{(1)}+1$。在代码的实现过程中，用〈VAR，EXP〉这样的二元组保存符号表达式信息，VAR 代表左值变量，EXP 表示右值表达式。

当遍历到某条指令时：1）若指令的右值表达式中存在某个变量已经有相应的符号表达式，则将用该表达式替换该变量；2）若指令的右值表达式存在常量与变量的运算，则用该常量的符号值与变量进行符号计算；3）遇到 Φ 函数时用“‖”连接变量的多种定值；4）保存元素为左值变量和新的右值表达式的二元组。

图 4 给出了如何迭代计算 SSA 形式的三地址码的符号约束。为了简化运算符优先级的处理，该方法在每次进行迭代替换时为被替换变量的符号表达式添加双括号。

三地址码	符号约束迭代计算
$RLO^{(0)}=I0.1$	$RLO^{(0)}=I0.1$
$RLO^{(1)}=I0.2\ \&\ RLO^{(0)}$	$RLO^{(1)}=I0.2\ \&(I0.1)$
$Q0.1^{(0)}=RLO^{(1)}$	$Q0.1^{(0)}=I0.2\ \&(I0.1)$

图 4 SSA 形式的三地址码

4 实验分析

4.1 实验环境

本文的实验环境是运行在 Intel Core7 4790 3.60GHz 和 8G 内存等硬件平台上的 Windows7 操作系统。代码实现采用 x86 下版本为 jdk 1.8.0_101 和 jre 1.8.0_101 的 Java 环境。

4.2 验证程序

实验采用了两个验证程序。程序一是西门子 S7-300 和 S7-400 编程语句表（STL）参考手册[12]中的控制传送带语句表程序，它的程序代码如表 4 所列，代码中绝对地址所对应的系统部件如表 5 所列。

表 4 控制传送带语句表程序

程序代码	解释
O I 1.1	//按动一个启动开关，可以接通电机
O I 1.3	
S Q 4.0	//按动一个停止开关或打开传送带端部的
O I 1.2	常闭接点，可以切断电机
O I 1.4	
ON I 1.5	
R Q 4.0	

表 5 绝对地址对应的系统部件

系统部件	绝对地址
启动按钮开关	I 1.1
停止按钮开关	I 1.2
启动按钮开关	I 1.3
停止按钮开关	I 1.4
传感器	I 1.5
电机	Q 4.0

由于程序一不含跳转指令，所有的指令都在一个基本块中，因此无法验证多路径下符号约束生成的有效性。本文通过程序二加以补充验证，程序二是带有跳转指令的数学运算，其程序代码如表 6 所列。代码所表述的语义是：如果 IW0＞10，则 QW0=(IW0−5)＊2；否则 QW0=(IW0+5)＊2。

表 6　含有跳转指令的语句表程序

程序代码	解释
L IW 0	//ACCU2=ACCU1;ACCU1=IW0
L 10	//ACCU2=ACCU1=IW0;ACCU1=10
>I	//RLO=IW0>10
JC L1	//若 RLO=1,跳转到 L1 处
L IW 0	
L 5	
−I	// ACCU1=ACCU2−ACCU1=IW0−5
JU L2	//无条件跳转到 L2
L1:L IW 0	
L 5	//ACCU2=ACCU1;ACCU1=5
+I	//ACCU1=ACCU2+ACCU1=IW0+5
L2:L 2	//ACCU2=ACCU;ACCU1=2
*I	//ACCU1=ACCU2*ACCU1
T QW 0	//QW0=ACCU1
BE	//代码结束

4.3　实验结果

程序自动完成了语句表到三地址码以及静态单赋值形式的三地址码的转换。程序一的三地址码如图 5 所示。其中,BB0 和 BB1 分别是入口基本块和出口基本块。

```
Infinite loop discovered in FB, linking BB2 to exit.
Control flow graph for FB:
BB0 (ENTRY)(in: <none>, out: BB2)
BB2(in: BB0 (ENTRY), out: BB1 (EXIT))
1 MOVE_X R0 Q4.0, BOOLConst: false,
2 MOVE_X R1 RLO, R2 I1.1,
3 OR_X   R1 RLO, R1 RLO,        R3 I1.3,
4 OR_X   R0 Q4.0, R0 Q4.0,      R1 RLO,
5 MOVE_X R1 RLO, R4 I1.2,
6 OR_X   R1 RLO, R1 RLO,
7 NEG_I  R6 I1.5, R6 I1.5,
8 OR_X   R1 RLO, R1 RLO,        R6 I1.5,
9 NEG_I  R6 I1.5, R6 I1.5,
10 NEG_I R1 RLO, R1 RLO,
11 AND_X R0 Q4.0, R0 Q4.0,      R1 RLO,
12 NEG_I R1 RLO, R1 RLO,
BB1 (EXIT)(in: BB2, out: <none>)
```

图 5　程序一对应的三地址码

变量前的 Rn 标识变量的编号,如图 6 所示,静态单赋值转换后的三地址码将相同变量赋予了不同的编号加以区分。

```
Register factory: Registers: 7
Infinite loop discovered in FB, linking BB2 to exit.
Control flow graph for FB:
BB0 (ENTRY)(in: <none>, out: BB2)
BB2(in: BB0 (ENTRY), out: BB1 (EXIT))
1 MOVE_X R7 Q4.0,  BOOLConst: false,
2 MOVE_X R8 RLO,       R2 I1.1,
3 OR_X   R9 RLO,       R8 RLO,       R3 I1.3,
4 OR_X   R10 Q4.0,     R7 Q4.0,      R9 RLO,
5 MOVE_X R11 RLO,      R4 I1.2,
6 OR_X   R12 RLO,      R11 RLO,      R5 I1.4,
7 NEG_I  R13 I1.5,     R6 I1.5,
8 OR_X   R14 RLO,      R12 RLO,      R13 I1.5,
9 NEG_I  R15 I1.5,     R13 I1.5,
10 NEG_I R16 RLO,      R14 RLO,
11 AND_X R17 Q4.0,     R10 Q4.0,     R16 RLO,
12 NEG_I R18 RLO,      R16 RLO,
BB1 (EXIT)(in: BB2, out: <none>)
Register factory: Registers: 19
Q4.0=((false|(I1.1|I1.3))&!((I1.2|I1.4)|!1.5))
```

图 6　程序一 SSA 形式的三地址码与符号约束

最后,程序为输出变量 Q4.0 所生成的符号约束为:Q4.0=((false|I1.1|I1.3))&!((I1.2|I1.4|!I1.5)),通过人工代码审查可以看出,该符号约束正确表达了程序的语义信息。

由于程序二中包含控制流分支,程序被划分为 6 个基本块,如图 7 所示。每个基本块括号内都标注了该基本块的前驱节点与后继节点。例如,BB4(in:BB2,out:BB5)表示基本块 4 的前驱节点为基本块 2,后继节点为基本块 5。该程序的基本块 3 和 4 都对 ACCU1 变量进行了定值,在基本块进行了引用,因此 SSA 转换后在基本块 5 的入口处插入了 Φ 函数,Φ 函数中包含了该变量在所有前驱节点中的定值及基本块信息。最后生成的符号约束 QW0=(((IW0−5)‖(IW0+5))*2)中“‖”连接了两种可能的取值,表示 QW0 既可能等于(IW0−5)*2,亦可能等于(IW0+5)*2,同样正确表达了程序的语义信息。实验结果与预期完全一致。

```
Control flow graph for ORGANIZATION_BLOCK:
BB0 (ENTRY)(in: <none>, out: BB2)
BB2(in: BB0 (ENTRY), out: BB3, BB4)
1  MOVE_I    R5 ACCU1,  INTConst: 0,
2  MOVE_I    R6 ACCU2,  INTConst: 0,
3  MOVE_I    R7 ACCU2,  R5 ACCU1,
4  MOVE_I    R8 ACCU1,  R2 IW0,
5  MOVE_I    R9 ACCU2,  R8 ACCU1,
6  MOVE_I    R10 ACCU1, INTConst: 10,
7  CMPGT_I   R3 RLO,    R9 ACCU2,     R10 ACCU1,
8  CJMP_RLO  R3 RLO,    BOOLConst: true, ADD,      BB4
BB4 (in: BB2, out: BB5)
9  MOVE_I    R16 ACCU2, R10 ACCU1,
10 MOVE_I    R17 ACCU1, R2 IW0,
11 MOVE_I    R18 ACCU2, R17 ACCU1,
12 MOVE_I    R19 ACCU1, INTConst: 5,
13 ADD_I     R20 ACCU1, R18 ACCU2,     R19 ACCU1,
BB3 (in: BB2, out: BB5)
14 MOVE_I    R11 ACCU2, R10 ACCU1,
15 MOVE_I    R12 ACCU1, R2 IW0,
16 MOVE_I    R13 ACCU2, R12 ACCU1,
17 MOVE_I    R14 ACCU1, INTConst: 5,
18 SUB_I     R15 ACCU1, R13 ACCU2,     R14 ACCU1,
19 JMP       BB5,
BB5 (in: BB3, BB4, out: BB1 (EXIT))
25 PHI       R21 ACCU1, (R15 ACCU1, R20 ACCU1)    {BB3, BB4}
20 MOVE_I    R22 ACCU2, R21 ACCU1,
21 MOVE_I    R23 ACCU1, INTConst: 2,
22 MUL_I     R24 ACCU1, R22 ACCU2, R23 ACCU1,
23 MOVE_I    R4 QW0,    R24 ACCU1,
24 RETURN
BB1 (EXIT)(in: BB5, out: <none>)
Register factory: Registers: 25
QW0=(((IW0−5)||(IW0+5))*2)
```

图 7　程序二 SSA 形式的三地址码与符号约束

结束语　本文提出了一种有效的语句表程序符号约束自动生成方法,它通过将语句表程序转换为静态单赋值形式的三地址码,然后通过迭代计算生成输出变量的符号约束,用于程序的正确性验证。基于 Java 开发环境,本文构建了一个符号约束生成的概念验证程序,实验结果表明该方法能够正确表达程序的语义信息。该方法还有较多不足,未来将进一步完善:

1)扩展定时器、计数器的语义表述;

2)区分变量在不同路径约束下的符号约束;

3)对生成的符号约束进行优化,消除冗余的括号及运算;
4)将该方法应用于实际的工业级 PLC 程序。

参 考 文 献

[1] 陈钢,宋晓宇,顾明. COQ 定理证明器辅助 PLC 程序验证和分析[J]. 北京大学学报(自然科学版),2010,46(1):30-34.

[2] 肖力田,顾明,孙家广. 一种 PLC 程序语言指称语义及函数的形式化定义方法[C]// 中国智能自动化会议. 2011.

[3] 陈雪琨. PLC 程序的 Petri 网建模与分析方法研究[D]. 泉州:华侨大学,2013.

[4] BIALLAS S,BRAUER J,ARCADE K S. PLC:A verification platform for programmable logic controllers[C]// Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. ACM,2012:338-341.

[5] DARVAS D,APIEGO B F,VINUELA E B. PLCverif:a tool to verify PLC programs based on model checking techniques[C]// International Conference on Accelerator and Large Experimental Physics Control Systems. 2015:911-914.

[6] ADIEGO B F,DARVAS D,VINUELA E B,et al. Applying model checking to industrial-sized PLC programs[J]. IEEE Transactions on Industrial Informatics,2015,11(6):1400-1410.

[7] MCLAUGHLIN S E,ZONOUZ S A,POHLY D J,et al. A Trusted Safety Verifier for Process Controller Code[C]// NDSS. 2014:14.

[8] OULD BIHA S. A Formal Semantics of PLC Programs in Coq[C]// IEEE International Computer Software and Applications Conference,COMPSAC 2011. Munich,Germany,DBLP,2011:118-127.

[9] BLECH J O,BIHA S O. On Formal Reasoning on the Semantics of PLC using Coq[J]. axXiv:2013. 1301:3047.

[10] IEC (International Electrotechnical Commission). IEC Standard 61131-3:Programmable controllers-Part 3[S]. 1993.

[11] CYTRON R,FERRANTE J,ROSEN B K,et al. Efficiently Computing Static Single Assignment Form and the Control Dependence Graph[J]. ACM Transactions on Programming Languages & Systems,1991,13(4):451-490.

[12] 西门子股份有限公司. SIMATIC S7-300 和 S7-400 语句表(STL)编程:参考手册[M]. 西门子股份有限公司,2002.

(上接第 333 页)

[18] HARN L,XU Y. Design of generalized ElGamal type digital signature schemes based on discrete logarithm[J]. Electronics letters,1994,30(24):2025-2026.

[19] HARN L. New digital signature scheme based on discrete logarithm[J]. Electronics Letters,1994,30(5):396-398.

[20] 韩小西,王贵林,鲍丰,等. 针对基于离散对数多重签名方案的一种攻击[J]. 计算机学报,2004,27(8):1147-1152.

[21] 杜海涛,张青坡,杨义先. 新的 ElGamal 型广播多重数字签名方案[J]. 计算机工程,2007,33(12):10-11.

[22] 王晓峰,张璟,王尚平. 多重数字签名方案及其安全性证明[J]. 计算机学报,2008,31(1):176-183.

[23] WROBLEWSKI G. General method of program code obfuscation[C]// International Conference on Software Engineering Research and Practice. 2002.

[24] LINN C,DEBRAY S. Obfuscation of executable code to improve resistance to static disassembly[C]// Proceedings of the 10th ACM Conference on Computer and Communications Security. ACM,2003:290-299.

[25] BARAK B,GOLDREICH O,IMPAGLIAZZO R,et al. On the (im) possibility of obfuscating programs[C]// Advances in cryptology—CRYPTO 2001. Springer Berlin Heidelberg,2001:1-18.

[26] GARG S,GENTRY C,HALEVI S,et al. Candidate indistinguishability obfuscation and functional encryption for all circuits[C]// 2013 IEEE 54th Annual Symposium on Foundations of Computer Science (FOCS). IEEE,2013:40-49.

[27] HOHENBERGER S,SAHAI A,WATERS B. Replacing a random oracle:Full domain hash from indistinguishability obfuscation[C]// Advances in Cryptology-EUROCRYPT 2014. Springer Berlin Heidelberg,2014:201-220.

[28] BONEH D,ZHANDRY M. Multiparty key exchange,efficient traitor tracing,and more from indistinguishability obfuscation[C]// Advances in Cryptology-CRYPTO 2014. Springer Berlin Heidelberg,2014:480-499.

[29] SHAAI A,WATERS B. How to use indistinguishability obfuscation:deniable encryption,and more[C]// Proceedings of the 46th Annual ACM Symposium on Theory of Computing. ACM,2014:475-484.

[30] HARN L,KRESLER T. New scheme for digital multisignatures[J]. Electronics Letters,1989,25(15):1002-1003.

[31] BONEH D,WATER B. Constrained pseudorandom functions and their applications[C]// International Conference on the Theory and Application of Cryptology and Information Security. Springer Berlin Heidelberg,2013:280-300.

[32] BOYLE E,GOLDWASSER S,IVAN I. Functional signatures and pseudorandom functions[M]// International Workshop on Public Key Cryptography. Springer Berlin Heidelberg,2014:501-519.

[33] KIAYIAS A,PAPADOPOULOS S,TRIANDOPOULOS N,et al. Delegatable pseudorandom functions and applications[C]// Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. ACM,2013:669-684.

[34] 杨亚涛,薛霆,李子臣. 广播多重量子数字签名方案的设计与分析[J]. 中国科学技术大学学报,2011,41(10):924-927.

[35] HU Y,JIA H. Cryptanalysis of GGH map[C]// Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer Berlin Heidelberg,2016:537-565.

[36] MILES E,SAHAI A,ZHANDRY M. Annihilation attacks for multilinear maps:Cryptanalysis of indistinguishability obfuscation over GGH13[C]// Annual Cryptology Conference. Springer Berlin Heidelberg,2016:629-658.

[37] LIN H. Indistinguishability obfuscation from constant-degree graded encoding schemes[C]// Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer Berlin Heidelberg,2016:28-57.