Journal of Computer Applications
计算机应用,2017,37(12):3581-3585
ISSN 1001-9081
CODEN JYIIDU
2017-12-10
http://www.joca.cn

文章编号:1001-9081(2017)12-3581-05
DOI:10.11772/j.issn.1001-9081.2017.12.3581

PLC 程序控制流分析方法

张　晔*,陆余良
(合肥电子工程学院,合肥 230037)
(*通信作者电子邮箱 zhangye_only@foxmail.com)

摘　要:可编程逻辑控制器(PLC)是工业控制系统的重要组成部分,控制着各类物理设备及工艺流程。无论是攻击者的恶意篡改还是内部人员的编程错误所造成的 PLC 控制程序错误都将严重威胁设备及人身安全。为解决该问题,提出了针对 PLC 程序的控制流分析方法。首先,利用 flex 和 bison 分析了源代码的词法及语法结构;其次,通过分析抽象语法树(AST)生成并优化了不含指令副作用的中间表示;最后,在中间表示的基础上划分基本块,并以此为基本单元构建了程序的控制流图。实验结果表明,所提方法能够恢复语句表形式 PLC 程序的控制流结构,为程序理解和安全性分析提供了基础。

关键词:可编程逻辑控制器;控制流;中间表示;程序理解;安全性
中图分类号:TP393.08;TP312　　**文献标志码**:A

Control flow analysis method of PLC program

ZHANG Ye*, LU Yuliang
(*Hefei Electronic Engineering Institute, Hefei Anhui* 230037, *China*)

Abstract: Programmable Logic Controller (PLC) is one of the most important components of industrial control system, which controls varieties of physical equipments and production processes. The faults of PLC control program caused by malicious tempering of attacker and programming errors of internal personnel will seriously threaten equipment safety and personal safety in industrial field. In order to solve this problem, a control flow analysis method of PLC program was proposed. Firstly, the lexical and syntactic structure of source code were analyzed by using flex and bison. Then, the intermediate representation without instruction side effects was generated and optimized by analyzing the Abstract Syntax Tree (AST). Finally, the basic blocks were divided on the basis of intermediate representation, and the control flow graph of the program was constructed by taking basic block as the basic unit. The experimental results show that, the proposed method can restore the control flow structure of PLC program in the form of statement table, and provide the basis for program understanding and security analysis.

Key words: Programmable Logic Controller (PLC); control flow; intermediate representation; program understanding; security

www.joca.cn

0　引言

自“震网”病毒面世以来,人们对工业控制系统安全的认识达到了前所未有的高度。作为工业控制系统的核心组件,可编程逻辑控制器(Programmable Logic Controller, PLC)关系着整个工业现场的运转秩序,同时也影响着财产、人身安全,其安全性保障也具有较高需求,因此 PLC 控制逻辑程序也往往成为攻击者的恶意篡改对象。与此同时,针对 PLC 程序的形式化分析、静态分析等程序分析方法成为了人们广泛研究的课题。

控制流分析是一类用于分析程序控制流结构的静态分析技术,目的在于生成程序的控制流图,在编译器设计、程序分析、程序理解等领域都有重要应用。针对 PLC 程序的控制流分析实际上分析的是工业控制系统中离散事件的先后执行顺序和状态的转换条件,也是各类 PLC 程序分析方法的基础。

本文针对西门子 S7 系列 PLC 所使用的 STL(STatement List)语言展开研究,开发了 STL 代码的控制流分析工具。STL 也被称为语句表,衍生于 IEC 61131-3[1]标准中的指令表语言,是一种类似于汇编语言的私有结构化语言。

1　相关研究

当前针对 PLC 程序的分析方法主要有模型检测[2-4]、静态分析[5-6]、定理证明[7-8]、符号执行[9]等。

模型检测主要用于验证程序有穷状态是否满足命题性质,文献[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,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[2]还支持西门子的 SFC、SCL 等高级语言,但该工具还处于概念验证阶段。

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。

收稿日期:2017-05-09;**修回日期**:2017-07-04。
作者简介:张晔(1993—),男,江西九江人,硕士研究生,主要研究方向:信息安全;　陆余良(1964—),男,江苏宜兴人,教授,博士,主要研究方向:信息安全。

文献[7-8]采用定理证明的方法将 PLC 程序转化为数学公式,并用定理证明器进行分析与验证。

文献[9]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 程序模型检测中状态爆炸的问题。

上述的大部分工作都为 PLC 程序的验证与分析奠定了良好的基础,但仍然存在两个问题:

1)在程序建模的过程中重点阐述了程序分析后端内容,对代码解析部分介绍得不够详细。

2)分析并未采用中间语言,而是基于源码进行建模,指令副作用导致了建模规则复杂、状态空间爆炸等问题。

本文针对这两个问题,提出了一种消除语义副作用的 STL 语言中间表示(Simple STL language Intermediate Representation, SSIR),在不改变 STL 语义的前提下将 STL 代码表示成 SSIR 形式,之后在 SSIR 的基础上进行相应的控制流分析,有效实现了程序的控制流恢复,使后端工作仅需要在控制流图的基础上进行即可。

2 控制流分析框架

控制流分析框架如图 1 所示,主要分为以下几个步骤:

1)利用词法分析生成器 flex 对 STL 程序进行词法分析,生成相应的 token 序列。

2)基于 bison 对 token 序列进行语法分析,构造程序的抽象语法树(Abstract Syntax Tree, AST)。

3)遍历抽象语法树生成中间表示 SSIR。

4)在中间表示的基础上划分基本块,构建程序控制流图用于过程内分析。

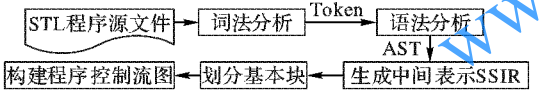

图 1 控制流分析框架

Fig. 1 Framework of control flow analysis

由于前两步的工作已公开发表不再赘述,本文主要介绍中间表示的生成与优化以及所面临的主要困难,并给出了程序的控制流图生成算法。

3 中间表示生成与优化

STL 源代码不适合直接用于控制流分析,主要因为源代码的指令副作用使得程序控制流不够清晰[9],因此,在完成语法分析后,通过遍历抽象语法树生成无指令副作用的中间表示 SSIR。本章首先介绍了 STL 程序基本单元,随即举例介绍了源代码中存在的指令副作用,之后介绍了中间表示的语法格式,最后列举了中间表示生成过程中 STL 特有元素的翻译规则。

3.1 STL 程序基本单元

3.1.1 逻辑块

逻辑块是 STL 程序的基本单元,不仅包含用户编写的程序,而且包含程序操作所需的数据。用户可编辑的部分主要有组织块(OB)、功能(FC)、功能块(FB)、数据块(DB)等,具体功能如下:

1)组织块相当于用户程序中的主程序,可以调用其他块,是操作系统与用户的接口,用于控制扫描循环、中断程序、PLC 的启动和错误处理等,决定了用户程序的结构。

2)功能和功能块类似于函数,通过其输入、输出参数实现与过程控制的传感器、执行器、用户程序中的其他块交换数据,主要区别在于功能块有独立的背景数据块。

3)数据块是用于存放用户数据的数据区域。

3.1.2 程序状态字与累加器

STL 程序主要依赖于程序状态字寄存器、累加器两类寄存器执行指令操作。

状态字是一个 16 位的寄存器,实际上仅使用了其中的 9 位,用于存储 CPU 执行指令后的状态、结果以及错误,几乎所有位逻辑运算指令在执行过程中都要与程序状态字进行交互。用户一般不直接使用状态位,但是 PLC 在解释执行代码时会对状态字进行隐式的写操作,从而影响了后续指令的执行结果。状态字的具体描述如表 1 所示。

表 1 程序状态字

Tab. 1 Program status word

位	简称	描述
0	FC	首次检测位。该位状态为 0 表示逻辑串的第一条指令。输出指令与 RLO 有关的跳转指令将该位清零
1	RLO	逻辑运算结果。用于存储位逻辑指令和比较指令的结果
2	STA	状态位。执行位逻辑指令时,STA 与位变量值一致
3	OR	或位。暂存逻辑“与”的运算结果
4	OS	溢出状态保存位。用于指明前面的指令执行过程是否发生过错误。SOS 指令、块调用指令、块结束指令能复位该位
5	OV	溢出位。算术运算、逻辑运算指令执行出错,溢出位被置 1
6	CC0	条件码 0。用于存储比较、算数、字逻辑、转移操作的状态信息
7	CC1	条件码 1。用于存储比较、算数、字逻辑、转移操作的状态信息
8	BR	二进制结果位。用于保存 RLO 位的值

累加器(ACCUx)是执行 STL 指令操作的关键部件。除位逻辑运算以外,大部分语句表操作都是在累加器中进行[10]。不同 PLC 型号的累加器数目不同,本文以两个累加器的为例介绍 STL 指令的副作用。

3.2 指令副作用

STL 语言的指令副作用主要源于状态字,对状态位的隐式读写操作由 PLC 解释执行,其增加了程序的条件分支,但并未显式地表现在代码中。

例如图 2 所示,双斜杠后的内容表示指令的隐式操作。与操作“A”将操作数的布尔值赋予状态寄存器 STA,第 3 行根据该指令是否是逻辑串的首指令来决定后续的指令流向,这就是指令副作用产生的额外控制流。为了减少副作用引起的额外控制流,需要对部分指令进行预处理及优化,最为主要的有 FC 状态位和 OR 状态位。

3.2.1 FC 状态位

FC 状态位为 0 表示该行指令是逻辑串的首指令,与 RLO 相关的操作将 STA 状态值直接赋予 RLO;否则,RLO 则需要与 STA 位进行相应的逻辑操作后再将值赋予 RLO。输出指令或与 RLO 有关的跳转指令将其置位为 0,表示逻辑串结束。

除状态字传送指令、O 先“与”后“或”指令[10]、调用背景块指令以外,FC 状态位值均取决于操作符而非操作值。因此,在源代码翻译为中间表示之前,仅需要对源代码进行一次遍历,标记执行前 FC 位为 0 的指令行号,之后行号已被标记的指令便可直接采用相应的翻译规则,从而省略多余的条件分支语句。预处理规则如下:

www.joca.cn

1)首行指令执行前 FC 位为 0。

2)赋值、置位、复位指令下一条指令若非跳转目标,则执行前 FC 位为 0。

3)与 RLO 相关的跳转指令的下一条指令若非跳转目标,则执行前 FC 位为 0。

4)除装载指令以外的计数器指令的下一条指令非跳转目标,则执行前 FC 位为 0。

```
1.  A I  0.1
2.  // STA = I 0.1
3.  // cjmp L1, FC==0
4.  // RLO=RLO&&STA
5.  // jmp L2
6.  // L1:  RLO=STA
7.  // L2:  FC = 1
8.  JC  L3
9.  // STA = 1
10. // OR = 0;
11. // FC = 0
12. // cjmp L3, RLO==1
13. // RLO = 1
```

图 2　STL 指令副作用

Fig. 2　Instruction side effects of STL

3.2.2　OR 状态位

OR 位主要功能是在 O 先“与”后“或”指令中保存之前“与”操作的运算结果,若运算结果为 1,则 OR 位值为 1,后续若无将 OR 位复位的指令,RLO 值一直为 1。

因此,需要在每条对 RLO 赋值的指令后添加“RLO = RLO or OR”指令,以此替代冗余的条件判断。实际上,OR 位在多数情况下为常量 0,也意味着该指令无效,所以可以在中间表示的优化过程中检测同一基本块中该条指令前 OR 是否为 0,若为 0 则删除该条指令。

3.3　SSIR 语法规则

中间表示 SSIR 抽象语法规则用巴科斯范式表示如图 3 所示。

```
<prog>  ::= |<prog><ins>*
<ins>   ::= jmp id  |cjmp id, exp | call id | call id,id
            |label : <ins>| BE |<var> := <exp>
<exp>   ::= <var>< binop>< var>
            |<unop>< var> |<var>
<binop>::= + | -| * | / | and | or | xor |…
<unop>  ::= minus | push | pop| fn | fp |…
<var>   ::= id:  <type>
<type>  ::= boolean | byte | short | int | long | timer
            |counter
```

图 3　SSIR 语法规则

Fig. 3　Syntax rules of SSIR

prog 表示构成程序的逻辑块结构,ins 代表指令,包括跳转指令 jmp id、条件跳转指令 cjmp id,exp、功能调用指令 call id、功能块调用指令 call id,id、块结束指令 BE、带有标签的指令 label: < ins > 、赋值指令 < var > := < exp > 。exp 为表达式,包含了变量、变量的二元运算、变量的一元运算。条件调用指令用于实现组织块对功能、功能块的条件调用。pop 和 push 指令用于嵌套指令的堆栈操作,在 3.4.5 节详细介绍。

实际上,SSIR 仅对 STL 源代码进行了局部优化,并且以四元式作为存储结构。STL 源代码的指令结构包括了标签、操作符、至多一个操作数、注释。SSIR 在 STL 的基础上消除了注释信息,为了消除指令副作用,SSIR 增加至三个操作数,包括一个目的操作数和至多两个源操作数。

3.4　特有元素的翻译规则

与一般程序相比,STL 具备一些特有的元素,在生成中间表示时需要得到相应的处理。下面介绍 STL 特有元素的翻译规则。

3.4.1　定时器

不同类型的定时器具备不同功能,西门子 S7 系列 PLC 为每个定时器都分配一个 16 位字和一个二进制位。定时器的 16 位字用于存放剩余时间值,二进制位表示定时器输出触点状态,根据定时器的类型在满足相应条件时发生跳变。SSIR 根据操作符类型判定程序访问的是逻辑值还是具体数值。

3.4.2　计数器

计数器的主要功能是在满足预置的指定数目脉冲后进行某种操作。西门子 S7 系列 PLC 为每个计数器都分配一个 16 位字和一个二进制位。计数器的 16 位字用于存放当前数值。计数器值大于 0 时,二进制位为 1;反之为 0。因此,SSIR 对计数器的处理与定时器相同。

3.4.3　边沿检测指令

某些指令在特定的内存单元值发生高低位变化时才会执行,边沿检测指令就是通过检测单个地址位信号的上升或下降来决定后续指令的执行与否。紧跟在“FN”算符或“FP”算符后的边沿存储位用于存储之前的逻辑运算结果,程序执行到该条指令,通过判断边沿存储位和指令执行前 RLO 值决定指令执行后 RLO 的取值。SSIR 将边沿检测指令转换为 RLO 位与边沿存储位的比较指令和边沿存储位的再赋值。

例如,“FN M 0.2”被转换为“RLO = RLO < M 0.2”和“M 0.2 = RLO”两条指令。

3.4.4　主控继电器指令

主控继电器(Master Control Relay, MCR)指令控制着一片区域的指令是否被正常执行,自“MCR(”指令开始,以“MCR)”指令结束。“MCR(”指令将 RLO 存放在一个 8 位深、1 位宽 MCR 堆栈中。“)MCR”删除 MCR 的栈顶项。

若进入 MCR 区域前,堆栈内所有 RLO 值为 1,则 MCR 激活,该区中与 MCR 相关的指令正常执行;否则,MCR 去活,赋值指令与传送指令均向寻址位写入“0”,复位和置位指令对寻址位不产生影响。因此,SSIR 为 MCR 区域内指令设立 MCR 标识,并且在 MCR 区域内这 4 类写操作指令执行前都要进行条件判断。

3.4.5　嵌套指令

除主控继电器指令以外,STL 语言中允许嵌套的指令只有“A(”“AN(”“O(”“ON(”“X(”“XN(”六类位逻辑运算指令。这六类嵌套指令将 RLO 和 OR 位以及指令操作数保存在嵌套堆栈中,当遇到“) ”指令时,堆栈中弹出输入项,恢复 OR 位,根据指令代码完成与 RLO 的运算。

SSIR 为位逻辑指令和 MCR 指令分别设立了位堆栈和 MCR 堆栈,根据指令操作数来判别具体使用的堆栈类型。

4　基于 SSIR 的控制流图生成算法

控制流分析分为过程内的控制流分析和过程间的控制流分析。因为组织块相当于 PLC 程序的主函数,所以在中间语言生成过程中将组织块所调用的功能块及功能代码内联至组织块程序中,针对 STL 程序的控制流分析便由此转化为针对 STL 程序组织块的过程内控制流分析。介绍控制流图生成算法前,首先介绍几个基本定义。

www.joca.cn

4.1 基本定义

定义 1 基本块。基本块是最大化的连续指令序列，控制流只能从第一条指令进入，从最后一条指令退出，基本块内部不存在分支。

基本块划分规则是：

1）第一条指令是入口语句。

2）条件跳转或无条件跳转的目标语句是入口语句。

3）条件跳转或无条件跳转指令的后一条指令是入口语句。

4）每条入口语句到下一个入后语句前的所有语句为一个基本块。

根据基本块的最后一条指令，可以分为四种类型：

1）单出口基本块：末指令是无条件跳转指令，或者该指令的下一条指令是分支转移的目标地址。

2）双出口基本块：末指令是条件跳转指令。

3）多出口基本块：末指令是多分支跳转指令，根据累加器的低字决定跳转目标。

4）返回基本块：末指令是功能、功能块返回指令或者块结束指令。

实际上，多分支跳转指令很少应用于 PLC 程序中[11]，所以在概念验证时不考虑多分支跳转。

定义 2 控制流图。控制流图 $G=(N,E)$ 是一个以基本块为节点的有向图。$N=\{B_1,B_2,\cdots,B_n\}$ 是程序的基本块集合，$E\subseteq N\times N$ 表示边的集合，如果存在 $(B_i,B,B_j)\in E$，则从 B_i 到 B_j 存在一条有向边，且 B_i 称为 B_j 的前驱节点，B_j 称为 B_i 的后继节点。

4.2 算法步骤

控制流图生成算法描述如下。

输入 SSIR 指令表；

输出 控制流图。

```
//若指令表为空,返回
if (List_Instruction == null) return;
//遍历指令表,将入口语句序号添加至入口语句序号表
for (Instruction ins: List_Instruction)
{
  //程序的第一条指令为基本块入口语句
  if (ins.Id == 0)
    List_Entry.add(ins.Id);
  //跳转指令的下一条指令及目标指令是入口语句
  if (ins.Op == "jmp" | "cjmp")
    List_Entry.add(ins.Id+1);
    List_Exit.add(ins.Id);
    List_Entry.add(LineOfLabel(ins.Label));
  //标签语句是入口语句
  if (ins.Label ! = null)
    List_Entry.add(ins.Id);
}
//将指令集大小加入入口语句表用于判定循环结束。
List_Entry.add(List_Instruction.size());
//遍历入口语句序号表,构建基本块,bc_i 表示指令序号,bb_i
//表示基本块号,n_bb 表示基本块总数
int n_bb = List_Entry.size()+1;
int[] n_pred = new int[n_bb];
ControlFlowGraph cfg = new ControlFlowGraph(n_bb);
//基本块 0 和 1 分别为程序的入口节点和出口节点。
cfg.basic_blocks[0] = new BasicBlock(0,-1);
cfg.basic_blocks[1] = new BasicBlock(1,-1);
int bb_i = 2; BasicBlock bb = null;
Iterator it = List_Entry.iterator();
for (;;)
{
  int bc_i = it.next();
  if (bc_i == bc.length) break;
  bb = cfg.basic_blocks[bb_i] = new BasicBlock(bb_i, bc_i);
  ++bb_i;
}
//为每个基本块添加后继节点信息
cfg.basic_blocks[0].successors = new BasicBlock[1];
cfg.basic_blocks[0].successors[0] = cfg.basic_blocks[2];
cfg.basic_blocks[1].successors = new BasicBlock[0];
Iterator it = List_Exit.iterator();
bb_i = 2;
while (it.hasNext())
{
  int bc_i = it.next();
  bb = cfg.basic_blocks[bb_i];
  //该基本块中没有出口语句
  while (bc_i > bb.end)
  {
    bb.successors = new BasicBlock[1];
    bb.successors[0] = cfg.basic_blocks[bb_i+1];
    BasicBlock next_bb = bb.successors[0];
    bb = next_bb;
  }
  //出口语句为非条件跳转指令
  if(ins[bc_i].op == "jmp")
  {
    BasicBlock target_bb = cfg.getBasicBlockByIndex
    (ins[bc_i].oprand1);
    bb.successors = new BasicBlock[1];
    bb.successors[0] = target_bb;
  }
  //出口语句为条件跳转指令
  if(ins[bc_i].op == "cjmp")
  {
    int bb_i = bb.id;
    BasicBlock next_bb = cfg.basic_blocks[bb_i+1];
    BasicBlock target_bb = cfg.getBasicBlockByIndex
    (ins[bc_i].oprand2);
    bb.successors = new BasicBlock[2];
    bb.successors[0] = next_bb;
    bb.successors[1] = target_bb;
  }
}
```

算法中：List_Entry 为入口节点表；List_Exit 为出口节点表；List_Instruction 为指令表；ControlFlowGraph 为控制流图类；BasicBlock 为基本块类；LogicBlock 表示逻辑块类。

4.3 实例分析

因为本文主要针对 PLC 程序的控制流进行分析，所以采用一个控制分支较多的实例程序进行实验验证。图 4 所示的的 STL 程序包含了常用的跳转语句及循环语句，将图中的指令转化为 SSIR 后分析其控制流，输出的控制流图如图 5 所示。程序分为 9 个基本块，其中，BB0 、BB1 分别是整个控制流图的入口块和出口块，二者均不含指令。

www.joca.cn

Line	Label	Operator	Operand	
1.		A	I	0.1
2.		JC	L1	
3.		A	I	0.2
4.		JCN	L2	
5.		L	5	
6.	BACK	T	LW	32
7.		L	MW	50
8.		L	30	
9.		+I		
10.		T	MW	50
11.		L	LW	32
12.		LOOP	BACK	
13.	L1	S	Q	1.1
14.		JU	L3	
15.	L2	R	Q	1.2
16.	L3	BE		

图 4　控制流分析实例代码

Fig. 4　Example code of control flow analysis

```
BB0(ENTRY)          (in:   <none>,    out:   BB2)
BB2          (in:   BB0  (ENTRY),   out:   BB3,  BB6)
1              RLO        :=   I0.1
2              RLO        :=   RLO      or    OR
3              FC         :=   0
4              cjmp       L1,  RLO==1
BB3          (in:   BB2,   out:   BB4,   BB7)
5              RLO        :=   I0.2
6              RLO        :=   RLO      or    OR
7              FC         :=   0
8              cjmp       L2,  RLO==0
BB4          (in:   BB3,   out:   Bb5)
9              ACCU2      :=   ACCU1
10             ACCU1      :=   5
BB5          (in:   BB4,   BB5,   out:   BB6,   BB5)
11             LW32       :=   ACCU1
12             ACCU2      :=   ACCU1
13             ACCU1      :=   MW50
14             ACCU2      :=   ACCU1
15             ACCU1      :=   30
16             ACCU1      :=   ACCU1    +     ACCU2
17             MW50       :=   ACCU1
18             ACCU2      :=   ACCU1
19             ACCU1      :=   LW32
20             ACCU1      :=   ACCU1    -     1
21             cjmp       BACK,     ACCU1>0
BB6          (in:   BB2,   BB5,  out:    BB8)
22   L1:       Q1.1       :=   1
23             FC         :=   0
24             jmp        L3,
BB7          (in:   BB3,   out:   BB8)
25   L2:       Q1.2       :=   1
26             FC         :=   0
BB8          (in:   BB6,   BB7,   out:   BB1  (EXIT))
27   L3:       BE
BB1 (EXIT)          (in:  BB8,  out:  <none>)
```

图 5　控制流图示例

Fig. 5　Example of control flow graph

将图 5 中字符串形式的控制流图转换为图形化表示如图 6 所示。BB5 是一个循环体,因此存在一条指向自身的回边。由于 BB6 中最后一条指令是无条件跳转指令,因此不存在指向 BB7 的边。

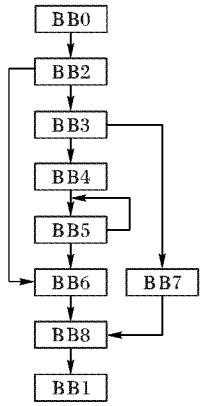

图 6　控制流图的图形化表示

Fig. 6　Graphical representation of control flow graph

5　结语

本文提出了 PLC 程序控制流分析方法,在对源程序进行语法分析的基础上,利用中间表示有效解决了指令副作用的问题,生成的控制流图为后续研究奠定了良好的基础。未来工作主要集中于以下三方面:1)浮点数指令的中间表示转换;2)SSIR 中间表示的静态单赋值形式转化;3)PLC 程序的数据流分析方法研究。

参考文献(References)

[1] International Electrotechnical Commission. IEC 61131-3: programmable controllers — Part 3: programming languages [S]. Geneva: IEC, 2003.

[2] DARVAS D, ADIEGO B F, VINUELA E B. PLCverif: a tool to verify PLC programs based on model checking techniques [C]// Proceedings of the 15th International Conference on Accelerator and Large Experimental Physics Control Systems. Melbourne, Australia: JACoW, 2015: 911 – 914.

[3] 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.

[4] BIALLAS S, BRAUER J, KOWALEWSKI S. Arcade. PLC: a verification platform for programmable logic controllers [C]// Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. New York: ACM, 2012: 338 – 341.

[5] STATTELMANN S, BIALLAS S, SCHLICH B, et al. Applying static code analysis on industrial controller code [C]// Proceedings of the 2014 IEEE Conference on Emerging Technology and Factory Automation. Piscataway, NJ: IEEE, 2014: 1 – 4.

[6] PRAHOFER H, ANGERER F, RAMLER R, et al. Opportunities and challenges of static code analysis of IEC 61131-3 programs [C]// Proceedings of the 2012 IEEE 17th Conference on Emerging Technologies & Factory Automation. Piscataway, NJ: IEEE, 2012: 1 – 8.

[7] HUUCK R. Semantics and analysis of instruction list programs [J]. Electronic Notes in Theoretical Computer Science, 2005, 115: 3 – 18.

[8] BIHA S O. A formal semantics of PLC programs in Coq [C]// Proceedings of the 2011 IEEE 35th Computer Software and Applications Conference. Washington, DC: IEEE Computer Society, 2011: 118 – 127.

[9] MCLAUGHLIN S, POHLY D, MCDANIEL P, et al. A trusted safety verifier for process controller code [EB/OL]. [2017-04-20]. http://pdfs.semanticscholar.org/3f5e/13e951b58c1725250cb60afc27f08d8bf02c.pdf.

[10] 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]. 慕尼黑: 西门子公司, 2002: 24. (Siemens LTD. STatement List (STL) for SIMATIC S7-300 and S7-400 Programming, Reference Manual [M]. Munich: Siemens, 2002: 24.)

[11] 廖常初. S7-300/400 PLC 应用技术[M]. 北京: 机械工业出版社, 2005: 120. (LIAO C C. Application Technology of S7-300/400 PLC [M] Beijing: China Machine Press, 2005: 120.)

ZHANG Ye, born in 1993, M. S. candidate. His research interests include information security.

LU Yuliang, born in 1964, Ph. D., professor. His research interests include information security.

www.joca.cn