基于控制流信息的克里普克结构生成方法

牛小鹏 李清宝 谢晓东
(解放军信息工程大学信息工程学院 郑州 450002)

摘 要 恶意程序检测是信息安全技术研究的重要内容,基于程序行为特征的检测可以弥补二进制特征码检测方法的很多不足。使用模型检验技术可以对程序的操作行为做属性验证,它需要对目标程序进行建模,得到一个符合克里普克结构的迁移系统。通过对模型检验技术和克里普克结构的研究分析,提出了一种以完整控制流信息为基础、采用贪婪归一策略的克里普克迁移系统生成方法。测试分析表明,利用该方法生成的迁移系统可以完整地描述控制流信息,也可以精确地刻画系统状态的改变。

关键词 模型检测,克里普克结构,控制流,系统状态,标记函数
中图法分类号 TP309 **文献标识码** A

Kripke Structure Generating with Control Flow Information

NIU Xiao-peng LI Qing-bao XIE Xiao-dong
(Institute of Network Engineering, PLA Information Engineering University, Zhengzhou 450002, China)

Abstract Malware detection is an important part of information security technology. The detection based on program behavior characteristics can remedy the limits of binary signature detection method. Model checking technology can verify a program's specific behavior property, which requires a model for the target program, in order to obtain a transition system which is coincident with Kripke structure. Current model checking technology and Kripke structure were thoroughly analyzed, and then the method of generating Kripke structure was proposed, which is based on the full control flow information and greed strategy. The generated transition system can fully represent the control flow information and describe the changes of target system status.

Keywords Model checking, Kripke structure, Control flow, System status, Lable function

1 引言

个人计算机互联成网变得越来越普遍,病毒和木马的感染方式和杀伤力都发生了深刻的变化。在前互联网时代,经典病毒的传播依靠感染可执行文件,杀伤范围也仅仅局限于几台或者几十台个人计算机。而在互联网时代,邮件木马、网页病毒等恶意程序都具有更多的感染方式,被感染后也能迅速传播给成千上万的主机。现在主流的反病毒技术仍然采用特征码匹配技术来识别特定的病毒木马,因此终端用户计算机上的病毒特征库就需要经常更新。而且,各种病毒木马很容易就能做到在保持功能语义不变的前提下,修改其语法结构,达到改变自身特征码的目的。基于特征码检测技术的安全防护软件如果不能及时地更新病毒库,就会检测失败。因此需要研究新的、更有效的方法检测代码的恶意行为,而不是单纯依靠二进制特征,使得终端用户不用为功能相同而仅仅是二进制特征不同的恶意代码更新病毒库。

模型检验技术可以验证计算机系统是否具有某方面的属性[1],当然也可以对目标代码是否具有恶意性进行检测验证。Johannes Kinder[2],Prbhat K. Singh[3]等人均有这方面的研究。它需要将系统抽象成规范的、符合克里普克结构(Kripke)的迁移系统 M(以下均简称为克里普克结构),再将需要验证的程序恶意属性用时态逻辑语言形式化描述出来,形成属性公式 Φ。最后把 M 和 Φ 送入模型检验器,进行推理分析演算,就可以判断目标程序是否满足该恶意属性。如果满足,则为恶意代码;不满足,则为普通良性代码。模型检验器还能够给出满足恶意属性的状态路径,即程序在满足什么样的输入条件后可以执行到该恶意分支。

综上所述,模型检验技术应用于恶意代码检测的关键在于两点:其一是如何从待检测的目标程序中生成规范的克里普克结构;其二是如何用时态逻辑语言形式化地描述种类繁多的病毒木马行为。本文主要讨论第一点,利用控制流信息来构建适合于模型检验器分析推理的克里普克结构。

Johannes Kinder[2]等研究者利用 IDA Pro 反汇编工具从二进制程序提取控制流图,基于控制流图构建克里普克结构,对恶意程序做 Windows API 调用时的参数计算、堆栈结构、各个 API 调用之间的时序关系等行为特征形成了 CTPL 逻

到稿日期:2011-07-08 返修日期:2011-09-29 本文受国家863项目(2009AA01Z434)资助。
牛小鹏(1982—),男,博士生,主要研究方向为网络信息安全,E-mail:xiaoyuer8082@163.com;李清宝 男,博士,教授,主要研究方向为网络信息安全、计算机体系结构;谢晓东 男,硕士生,主要研究方向为网络信息安全、应用数学。

辑公式，验证了模型检验技术在基于行为特征的恶意程序检测方面的有效性。但是其工作有两点需要改进：(1)程序模型中每个节点只包含一条指令语句，这就可能包含无效指令或者有效指令，导致状态集庞大，甚至爆炸的问题；(2)不支持对间接跳转指令的处理，遇到间接跳转指令则使其跳转指向自身。胡刚[4]等人研究并研制了二进制分析平台 AmPro，它可以从二进制程序提取控制流图，但其目的是帮助分析人员理解程序，所提取的控制流图并不适合于自动推理分析，更不利于模型检验技术的应用。另外，AmPro 也没有对间接跳转指令进行处理。

本文根据模型检验技术对目标系统的建模需求，提出了简明控制流信息表示方法，依据贪婪归一策略从控制流信息中生成适合于模型检测器自动推理分析的克里普克结构。本方法所生成的克里普克结构可以在不改变语义表达能力的情况下减少状态数目，减小状态爆炸的风险。

2 克里普克结构及生成方法

2.1 模型检验

模型检验[5]是用算法验证某个有限状态系统是否满足某个确定的属性。在最近若干年中，模型检验已经成为硬件设计正确性验证的标准化工具，其已经逐渐应用于验证软件的正确性，特别是应用于检测某软件程序是否具有某种恶意功能。在模型检验过程中，用一个带标记的迁移系统代表目标系统，这个迁移系统需要符合克里普克结构。例如模型检验器 NuSMV 采用 Kripke 结构为被检测系统建模，可以检验用计算树逻辑 CTL(Computation Tree Logic)和线性时态逻辑(Linear-time Temporal Logic)描述的系统性质[6]。迁移系统中的状态节点被一组原子命题标记，这些原子命题表示目标系统在该状态节点上满足为真的属性。模型检验要验证的属性多用时态逻辑公式或者 μ-演算形式化描述，比如用 CTL*，CTL，LTL 等。

文献[5]指出，为了验证一个系统是否满足一个性质，必须做 3 件事情：

(1)使用模型检测器的描述性语言对系统建模，得到一个模型 M；

(2)用模型检测器的规范逻辑语言对性质编码，产生一个时态逻辑公式 Φ；

(3)以 M 和 Φ 作输入，运行模型检测器。

如果 $M \models \Phi$，则模型检测器输出回答“yes”，否则输出“no”，大多数模型检测器会产生导致失败的系统行为轨迹。本文主要研究上述 3 件事情中的第一件事情，即对目标待验证的程序代码建模，生成符合克里普克结构的迁移系统。

2.2 克里普克结构

克里普克结构可以抽象为一个三元组 $M=(S,T,L)$。其中 S 是状态集合；T 是集合 S 上的二元关系 $T\in S\times S$，使得每个 $s\in S$ 都有某个 $s'\in S$，满足 sTs'；L 是标记函数，$L:S\rightarrow 2^P$，其中 P 表示某个系统中原子命题(公式)的集合，这些原子命题代表系统可能成立的原子事实，诸如“进程 3259 被挂起”、“寄存器 R1 的内容是整值”，或者“将内存地址为 L1 的内容搬移到内存地址为 L2 的地方”。

可以把 L 看成是对所有原子命题的一个真值赋值，如命题逻辑一样。其差别之处是迁移系统不只含有一个状态，因此这种赋值依赖于系统具体所处的某个特殊状态 s，$L(s)$ 包含了在状态 s 下赋值为真的所有原子命题。

从降低系统分析复杂性方面考虑，按照调用关系将整个程序的控制流图划分成若干个子控制流图。每一个子控制流图被抽象为一个克里普克结构。因此，目标程序模型就是一组克里普克结构的集合，每一个结构代表一个控制流子图。故为得到目标程序的克里普克结构，首先需要从二进制代码逆向得到其控制流图，然后对控制流图进行抽象提升，定义克里普克结构中的状态以及与状态相关的标记函数。

2.3 克里普克结构生成方法分析

从目标待分析代码生成克里普克结构，目前常用的方法是利用反汇编工具 IDA Pro 提取目标代码的控制流信息，生成控制流图，然后将控制流图用形式化语言来描述，比如模型检测器 Spin 的标准建模语言 Prolema。这种方法存在如下几点不足需要改进：(1)该方法仅对验证上层应用程序的行为属性有效，对较底层的固件程序效果不是很理想，比如 BIOS。其原因是固件程序通常无结构信息提示。(2)克里普克结构中的状态信息未优化，存在冗余现象，容易使状态集膨胀。

二进制程序代码可以分为结构化的代码和无结构化的代码。在结构化二进制程序中，指令部分和数据部分的区分依赖于规范的文件格式来完成，比如 Windows 程序都需要符合 PE 结构，而固件程序中只包含数据和指令，并没有任何关于程序入口地址、函数开始地址、代码和数据空间位置大小及分布情况的信息。IDA Pro 对无结构的二进制代码反汇编效果不是很理想，其不能提取完整的控制流信息，基于该控制流信息也就不能生成准确的克里普克结构。IDA Pro 也无法正确处理 X86 实模式和虚拟模式相混合编码的固件程序[7]。

所以，使用模型检测验证系统底层的固件程序行为特征不能套用已有的克里普克结构生成方法，需要考虑到固件代码的特殊性。

3 克里普克结构生成过程

3.1 相关定义

控制流图由若干个节点组成，每个节点可以由一条或若干条指令组成。控制流图的定义方法直接关系到对系统行为的建模和推理分析，因此选择合适的方法构建控制流图很重要。文献[2]对控制流图及控制流图中节点的非形式化定义如下，控制流图由若干子图组成，每个子图代表一个子过程，其中每个节点包含一条指令。这种定义的优点在于每个节点代表克里普克结构中的一个状态，状态的变迁可以清晰地表征系统中寄存器、内存、堆栈、IO 等关键资源中数值的变化，便于对程序行为进行推理分析。不足之处是每个状态强制规定为包含一条指令，这就造成某些状态的变迁对系统状态变化的描述是无效的，比如 nop 指令、jmp 指令等，容易造成状态数目众多、状态集膨胀的问题。某些恶意代码为了躲避二进制特征匹配，就采用了插入垃圾指令(主要是 nop)、花式跳转等编码方式。文献[4]对控制流图及控制流图中节点的形式化定义如下。

定义 1(程序块)　程序块是一个三元组 $B=(start,end,mode)$,$start$ 是块的开始地址,end 是块中最后一条指令地址+1,$mode$ 是编码模式。

程序块通常用一个小写字母 b 表示,但有时为了区分不同的块,记为 b_s,b 是块标识,下标 s 为块的开始地址。如果 b 是 P 中的一个程序块,则称 b 属于 P,记为 $b<P$。

基本块是特殊的程序块,用 $Base(b)$ 表示。基本块中只包含一组连贯的指令,控制流在块开始地址进入,在块结束地址离开,而没有在块结束地址之外的地方停止或出现分支的可能性。即除块中最后一条指令可以为跳转指令外,其它指令都必须是顺序指令。如图 1 所示,语句 11—13 构成一个基本块,每个基本块只有一个入口和一个出口。

L0000000E:20 48 04	JB	#48H,#0015H
L00000011:E4	CLR	A
L00000012:93	MOVC	A,@A+DPTR
L00000013:80 01	SJMP	#0016H
L00000015:E0	MOVX	A,@DPTR
L00000016:60 06	JZ	#001EH

图 1　代码片段和基本块

定义 2(控制流图)　控制流图 cfg 可以定义为一个三元组 $cfg=(B,E,b_0)$:B 是程序基本块的集合,$B=\{b\mid b<P, Base(b)\}$;E 是边的集合,$E=\{\langle b,d\rangle\mid b\in B,d\in B\}$ 表示 b,d 两个节点之间存在控制转移,b_0 是根节点 $b_0\in B$,且 $b_0.start=P.start$。

文献[4]对控制流图及程序基本块的定义能够很好地描述程序的控制流信息,与文献[2]的区别在于文献[4]允许控制流图中节点内包含一组连贯的指令,这样就可以用最简洁的流图来表示控制流信息,使图中的节点个数尽可能地少,但是这不利于描述系统状态的实时变化。为方便后面的论述,事先做如下几个定义。

由于程序的所有操作都可以看作是对内存单元的操作(堆栈、寄存器也可以视作具有固定地址的内存单元),而且同一个操作的效果依赖于处理器的操作模式,因此可以通过内存分布结构和处理器操纵模式来定义系统状态。

定义 3(系统状态)　系统状态表示为二元组 $S=\langle mem_i, mode\rangle$,其中 mem_i 表示与指令 i 相关的内存值,$mode$ 表示处理器模式。用 ${}^iS$ 表示执行指令 i 前的系统状态,用 S^i 表示执行指令 i 后的系统状态。

定义 4(状态等价性)　令 $S_1=\langle mem_1, mode_1\rangle$,$S_2=\langle mem_2, mode_2\rangle$,iff $mem_1=mem_2$ and $mode_1=mode_2$, then $S_1\equiv S_2$,称系统状态 S_1 等价于系统状态 S_2;用 $S_1\neq S_2$ 表示系统状态 S_1 不等价于系统状态 S_2。

定义 5(有效指令)　令 I 表示目标系统的指令系统,$i_{valid}=\{i\in I\mid {}^iS\neq S^i\}$。

定义 6(无效指令)　$i_{invalid}=\{i\in I\mid {}^iS\equiv S^i\}$。

定义 7(系统状态刻画粒度 S-$granularity$)　控制流图中从一个节点迁移到另外一个节点时系统状态 $S=\langle mem_i, mode\rangle$ 被改变的次数。

根据前面对无效指令的定义,nop 指令和 jmp 类型的指令都属于无效指令,它们的执行不改变系统状态,而其它类型的指令都属于有效指令。我们的目的是使控制流图既能反映目标程序的控制流信息,又能够通过状态迁移来标识系统状态的变化。因此,综合考虑文献[2,4]对控制流图定义的优点和不足,提出一种简明控制流表示方法,其核心思想是控制流图中每个节点只包含一条能够改变系统状态的有效指令,如某条有效指令后面跟着若干条对系统状态改变无显式影响的无效指令,就将后面的无效指令与该有效指令紧缩在一个状态中。具体构建方法在后文阐述。

令 cfg_1 是采用文献[2]中的控制流图构造方法构造的控制流图,cfg_2 是采用文献[4]中的控制流图构造方法构造的控制流图,cfg_3 是采用简明控制流信息表示法构造的控制流图。令 $\pi_1=s_{10}\rightarrow s_{11}\rightarrow s_{12}\rightarrow\cdots\rightarrow s_{1k}$ 是 cfg_1 中的任意一条状态路径,$\pi_2=s_{20}\rightarrow s_{21}\rightarrow s_{22}\rightarrow\cdots\rightarrow s_{2m}$ 是 cfg_2 中与 π_1 相对应的状态路径,$\pi_3=s_{30}\rightarrow s_{31}\rightarrow s_{32}\rightarrow\cdots\rightarrow s_{3n}$ 是 cfg_3 中与 π_1 相对应的状态路径,它们实现的功能是等价的,且一般情况下有 $m<n<k$。在 cfg_1 中 S-$granularity\in\{0,1\}$,在 cfg_2 中 S-$granularity\geqslant 0$,在 cfg_3 中 S-$granularity\in\{0,1\}$。由以上分析可知,根据简明控制流信息表示法构造出来的控制流图具有状态数目少、对系统状态变化刻画的粒度小、更精确的优点。

简明控制流信息表示法已经能够为克里普克结构的三元组提供前二元的信息,其中控制流图中的每个节点表示状态,节点之间的连线表示状态之间的迁移关系。为了构建克里普克结构,还需要为每个状态指定相应的标记函数,同时为了便于模型检验器的推理分析,所以为每个状态定义了两个原子命题:第一个命题代表了与某个状态相关联的有效指令(包括操作码和操作数);另一个命题是该状态的唯一表示符,选定特殊命题标记 $\#loc(L)$,其中 L 为该状态对应的有效指令的地址。如图 6 所示第二个节点所代表状态 s_1 对应的有效指令为 CLR P3.7,地址为 $0x0089$;第三个节点所代表状态 s_2 对应的有效指令为 $mov\,Adr\_l,\#10H$,地址为 $0x008B$。状态 s_1 所对应的标记函数为 $CLR(P3.7),\#loc(0x0089)$,即 $L(s_1)=\{CLR(P3.7),\#loc(0x0089)\}$;状态 s_2 所对应的标记函数为 $mov(Adr\_l,\#10H),\#loc(0x008B)$,即 $L(s_2)=\{mov(Adr\_l,\#10H),\#loc(0x008B)\}$。也就是说,在状态 s_1 下,命题 $CLR(P3.7)$ 和命题 $\#loc(0x0089)$ 为真,其它命题为假;在状态 s_2 下,命题 $mov(Adr\_l,\#10H),\#loc(0x008B)$ 为真,其它命题为假。

3.2　具体过程

3.2.1　状态节点的生成

如前分析了两种控制流图中节点的组织方法,并提出了适合模型检验推理分析的简明控制流信息表示法,这里给出其具体方法的描述。与状态节点紧密相关的是反汇编后的指令序列,构造状态节点的过程实质上是对反汇编后的指令分类的过程。构造需要遵循如下两条原则:

(1)每个节点最多只包含一条有效指令;

(2)跳转指令只能出现在节点的最后。

依据上述原则,提出了基于贪婪归一策略的状态节点构造算法,其描述如表 1 所列。贪婪归一策略是指在构造状态

节点时，如果遇到无效指令，则继续扫描，直至扫描到有效指令为止。

表 1　状态节点构造算法

顺序扫描指令序列，可能会遇到 3 种情况：
(1)遇到跳转类型的指令(包括 call 指令)，那么该条指令独立成为一个节点。
(2)遇到无效指令(不包括跳转类)，继续扫描，直到遇到跳转类型的指令或者遇到一个有效指令，最终扫描到的指令与前面的指令归结在一个节点内部。
(3)遇到有效指令，继续扫描，直到遇到跳转类型的指令或者遇到另一个有效指令。如果最终扫描到的指令是跳转类型的指令，那么该跳转类型的指令与它前面的指令归结在一个节点内部；如果最终扫描到的指令是另一个有效指令，该指令被排除在该节点外，属于下一节点。

本状态节点构造算法的流程图如图 2 所示。

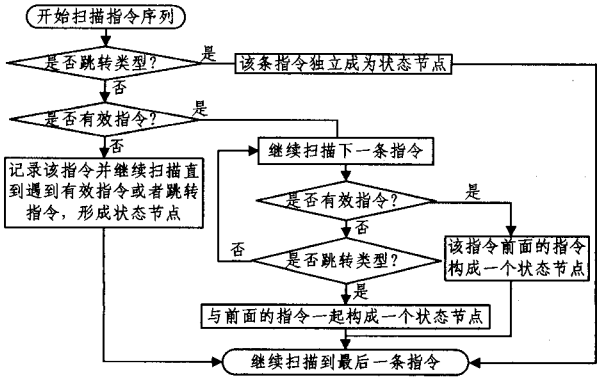

图 2　状态节点构造算法流程图

3.2.2　转移关系的生成

确定状态节点后，就需要确定状态之间的迁移关系。根据各个状态节点类型的不同，其迁移关系也不同，具体方法如表 2 所列。

表 2　迁移关系构造方法

(1)针对每一个无条件跳转类型的状态节点，只连接到它的目标跳转节点。
(2)针对每一个条件跳转类型的状态节点，则存在一个非确定性的连接关系，一方面是连接到它的直接后继节点，另一方面则是连接到它的目标跳转节点。
(3)针对间接跳转类型的状态节点，采用基于程序静态后向切片的控制流恢复算法，使间接跳转类型的状态节点连接到它的动态目标节点。具体方法请参考作者的另一篇文章。
(4)针对子函数调用类型节点，一方面连接到它的目标调用节点，另一方面要连接到它的后继节点。
(5)针对子函数调用返回节点，则不连接到任何节点，因为它的返回目标地址只有在函数动态调用的过程中才能真正确定。
(6)针对其它普通类型的节点，则直接连接它的后继节点，不需要做任何特殊处理。

3.2.3　标记函数的生成

对克里普克结构中的每个状态节点定义两个原子命题：第一个命题以该状态包含的有效指令为名称，包括操作码和操作数；第二个命题的作用是唯一标识该状态，以该状态所包含的首条指令地址为名称，记作 $\#Loc(Adrr)$。针对状态 s 的标记函数是 $L(s)=\{opcode(parameter1,parameter2,parameter3),\#Loc(Addr)\}$。

其中第二个命题唯一标识某个状态，它可以在属性描述公式里规定各个状态的时序关系。比如在下面的属性描述公式中，$\exists L\exists r_1 EF(mov(r_1,0)\wedge EF\#loc(L))\wedge\exists r2EF(push(r_2)\wedge EF(push(r_1)\wedge\#loc(L)\wedge EF(call(fn)))))$，地址谓词 $\#loc(L)$ 规定了 $mov(r_1,0)$ 和 $push(r_1)$ 所操作的是同一个寄存器。

4　比较分析

使用模型检验技术验证某个程序的行为属性，需要对目标程序建模，得到符合克里普克结构的迁移系统。为构建目标程序的迁移系统提出了简明控制流信息表示方法。

对图 3 的样本程序采用不同的控制流图定义方法，将会得到不同的控制流图，当然也是不同的克里普克结构。

依据文献[2]的方法抽象出来的控制流图如图 4 所示。由图 4 可知，状态数目越多，在模型检验过程中越容易产生状态爆炸问题。

```
Lable1:
JNB P3.7,Lable1
NOP
NOP
CLR P3.7
MOV Adr_L,#10H
MOV Adr_H,#00H
LCALL Read_EP
SETB P3.7
NOP
NOP
MOV A,EP_D
CJNE A,#01H,lable2
LCALL LCD_SHOW_WRONG
LCALL GET_KEY_INPUT
LJMP Lable1
Lable2:
RET
```

图 3　示例代码片段

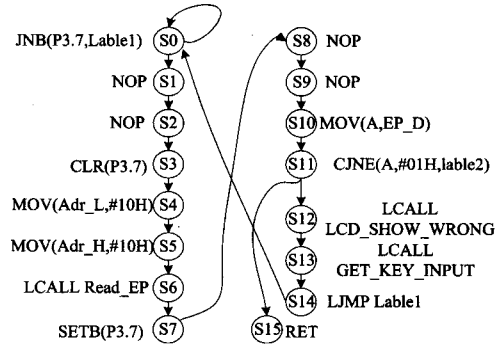

图 4　控制流图 1

依据文献[4]的方法抽象出来的控制流图如图 5 所示。其虽然状态数目比较少，控制流信息也很清晰，但是每个状态中很有可能包含多条能够改变系统状态的指令，不利于对系统的属性做进一步的推理分析。

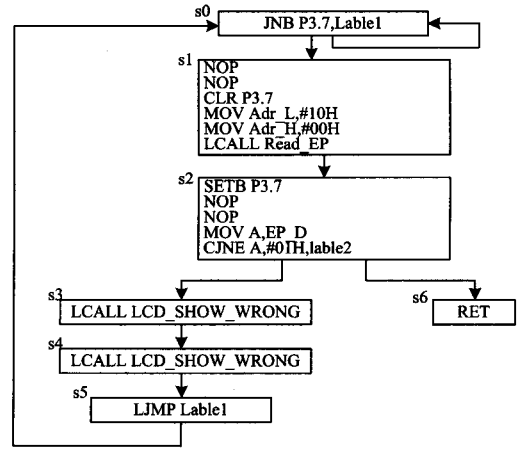

图 5　控制流图 2

如图 5，状态 s_1 中系统状态被改变了 4 次，对于各个状

态，系统状态改变次数分别是 1,4,2,1,1,0,1。

图 6 是在综合分析了文献[2,4]的思想后提出来的简明控制流信息表示图，其状态数目比图 4 中的状态数目少，比图 5 中的状态数目多，在每个状态，系统状态被改变的粒度为 0 或 1。

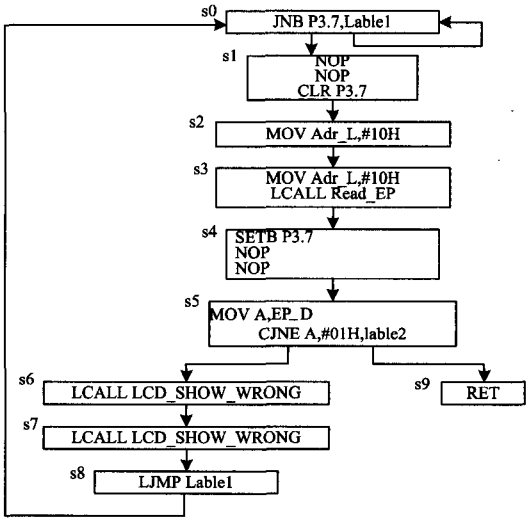

图 6　控制流图 3

表 3　分析结果比较

方法	描述能力	刻画粒度	状态数目
简明控制流信息表示法	强	{0,1}	较少
文献[2]方法	强	{0,1}	多
文献[4]方法	强	≥0	少

将简明控制流信息表示方法与文献[2,4]所提出的方法做了比较分析，主要在控制流信息描述能力、系统状态刻画粒度、生成克里普克结构后的状态数目等 3 个方面做比较。分析结果如表 3 所列。分析结论表明，采用简明控制流信息表示法生成的克里普克结构具有控制流信息描述能力强、对系统变化情况的刻画粒度精确、状态数目较少的特点，其适合用于模型检验技术。

结束语　本文针对如何从二进制程序自动生成克里普克结构以便于模型检验器进行推理验证的问题，提出了简明控制流信息表示法，其可以有效地表示控制流信息，也可以实时地表征系统状态的变化。下一步的工作重点是用时态逻辑语言形式化描述各种病毒、木马的恶意行为，然后与克里普克结构一起送入模型检验器进行检验分析。

参 考 文 献

[1] Huth M, Ryan M. Logic in Computer Scienc[M]. 何伟，樊磊，译. 北京：机械工业出版社，2007：115-168

[2] Kinder J, Katzenbeisser S. Proactive Detection of Computer Worms Using Model Checking[J]. IEEE Transactions on Dependable and Secure Computing, 2009

[3] Singh P K, Lakhotia A. Static Verification of Worm and Virus Behavior in Binary Executables using Model Checking [C]// Workshop on Information Assurance. United States Military Academy West Point, NY, June 2003

[4] 胡刚. 固件程序代码逆向分析关键技术研究[D]. 郑州：郑州信息工程学院，2010

[5] Clarke E, Emerson E. Design and synthesis of synchronization skeletons using branching time temporal logic[J]. Logics of Programs, ser. LNCS, Springer, 1981, 131: 52-71

[6] 曾红卫，繆淮扣. 模型检验在构件数据流测试中的应用[J]. 计算机科学与探索，2110, 4(12)：1122-1130

[7] Ilfak Guifanov. IDA Pro disassemble[EB/OL]. http://www.datarescue.com/index.htm

(上接第 83 页)

[2] Pinagapany S, Kulkarni A V. Solving channel allocation problem in cellular radio networks using genetic algorithm[C]// International Conference on Communication Systems Software and Middleware and Workshops. Jan. 2008：239-244

[3] Jie H L, Chiu C T, Tzung P H. A maximum channel reuse scheme with Hopfield Neural Network based static cellular radio channel allocation systems[C]// IEEE International Joint Conference on Neural Networks. June 2008：3660-3667

[4] Jiayuan C, Olafsson S, Xu Gu. Observations on Using Simulated Annealing for Dynamic Channel Allocation in 802.11 WLANs [C]// International Conference on Vehicular Technology. May 2008：1801-1805

[5] Gozupek D, Genc G, Ersoy C. Channel assignment problem in cellular networks: A reactive tabu search approach[C]// International Symposium on Computer and Information Sciences(ISCIS 2009). Sept 2009：298-303

[6] San R C, Gigi E, Lyandres V. Channel assignment for cellular mobile networks with nonuniform cells: an improved heuristic algorithm[J]. IEEE Proceedings of Communications, 2006, 153 (1/2)：61-68

[7] Kim V, Liu Wei, Cheng Wen-qing. Channel Assignment Problem in Cellular Mobile Network: A Distributed Constraint Satisfaction Approach [C]// International Conference on Communications and Mobile Computing(CMC). 2010：132-137

[8] Khanbary L M O, Vidyarthi D P. A GA-based Effective Fault-tolerant Model for Channel Allocation in Mobile Computing[J]. IEEE Transactions on Vehicular Technolgoy, 2008, 57 (3)：1823-1833

[9] Yokoo M, Hirayama K. Algorithms for Distributed Constraint Satisfaction: A Review[J]. International Conference on Autonomous Agents and Multi-Agent Systems, 2000, 2(2)：185-207

[10] Yan J, Jian Z. Backtracking Algorithms and Search Heuristics to Generate Test Suites for Combinatorial Testing[C]// International Conference on Computer Software and Applications. 2006, 1：385-394