基于敏感位置识别的状态化简技术研究

高洪博*　李清宝　王 炜　朱 瑜
(解放军信息工程大学　郑州　450002)

摘　要：模型构建是模型检验的基础，在微控制器代码模型构建过程中面临状态爆炸的问题。由于生成模型的状态数量与代码规模密切相关，通过简化代码可以有效缩减生成的状态数量。该文提出了敏感变量和敏感位置的概念，并以此为基础提出了结合子程序摘要信息的敏感位置识别算法；该算法从待验证的性质出发，提取敏感变量，识别代码中与敏感变量相关的敏感位置；模型构建过程中只对敏感位置对应代码进行建模，从而实现对模型状态的缩减。实验结果表明所提的方法能够有效缓解微控制器代码模型生成过程中的状态爆炸问题。

关键词：模型检验；状态爆炸；敏感变量；敏感位置
中图分类号：TP309　　**文献标识码**：A　　**文章编号**：1009-5896(2013)03-0742-07
DOI: 10.3724/SP.J.1146.2012.00878

The Study of State Simplification Techniques Based on Sensitive Position Identification

Gao Hong-bo　Li Qing-bao　Wang Wei　Zhu Yu
(*PLA Information Engineering University*, *Zhengzhou* 450002, *China*)

Abstract: Model construction is the basis of model checking. State explosion can not be avoided during building model for microcontroller code. Because the state number of generated model is related to code size, the number of state can be reduced through simplifying microcontroller code. An algorithm of sensitive position identification with subroutine summary information is proposed, based on concepts of sensitive variable and sensitive position. Sensitive variables are extracted from verified properties and used to identify sensitive positions. Then model is constructed from code corresponding to sensitive positions. Experimental results show that the problem of state explosion can be effectively alleviated through the proposed method.

Key words: Model checking; State explosion; Sensitive variable; Sensitive position

1　引言

作为一类广泛使用的嵌入式系统核心芯片，微控制器大量应用于工业控制、自动化、航空航天等设备中，保证其代码的安全性十分关键。模型检验作为一种重要的形式化验证方法，具有高度的自动化和良好的实用性，在系统安全性验证上日益受到重视，适合于对微控制器代码进行安全性分析[1−3]。

在程序代码验证中，模型检验主要应用于C语言，Java等高级语言源程序的验证[4,5]。不同于高级语言源程序，微控制器代码具有硬件依赖、环境相关等特性，因此对其进行模型检验不同于对高级语言源程序的验证，相关研究日益增多[6,7]。模型构建是模型检验的基础，在构建模型的过程中面临状态爆炸问题，为解决该问题研究人员提出了各种不同的技术，如对称化简、偏序化简、程序抽象和运行时分析等[8,9]。针对微控制器代码的模型检验，文献[1]提出了采用静态分析的方法来缩减模型状态。已有的状态缩减技术多数从模型检验算法或者程序代码自身出发对状态进行缩减，很少涉及待验证性质。模型检验的目的是为了验证性质与模型是否相符合，所以待验证性质与模型密切相关。待验证性质一般只与生成模型的代码中的部分位置和变量相关，为对某一性质进行验证，可以首先识别与待验证性质相关的程序代码，然后只将这些相关代码转化为模型，从而达到状态规模的缩减[10,11]。

本文从待验证性质出发，通过识别代码中与性质相关的位置实现对代码化简，进而解决模型构建过程中的状态爆炸问题。首先提出微控制器代码模型生成算法，根据代码的控制流，结合数据流分析，将二进制代码转化为对应模型；然后从待验证性质出发，提出结合子程序摘要信息的敏感位置识别算法，对二进制代码进行预处理，识别出与待验证性质相关的那些程序位置，然后只对这些位置处的代

2012-07-09收到，2012-12-04改回
国家863计划项目(2009AA01Z434)资助课题
*通信作者：高洪博　ghb912@163.com

码应用模型生成算法生成模型，最终实现状态的缩减，有效缓解状态爆炸问题。

2　微控制器二进制代码模型生成

为系统构建模型是进行模型检验的基础。针对微控制器代码，文献[1]通过构建模拟器模拟程序执行的方式获得状态和状态迁移关系，生成模型，其缺点在于一方面生成的状态空间较大，每个状态都要包括寄存器、内存、栈等所有信息；另一方面要为每种微控制器都构建对应的模拟器，工作量大，而且导致检验系统可扩展性不好。

本文以微控制器二进制代码为研究对象，通过反汇编工具得到对应的汇编程序，然后从汇编代码出发，按照代码控制流的执行顺序，结合数据流分析得到对应的模型，用迁移系统表示。为便于描述，首先给出以下几个概念。

控制流图　程序 P 的控制流图 CFG 是一个有向图，用四元组 $<N,E,n_{in},n_o>$ 表示，其中 N 是节点集，程序中每条语句对应 CFG 中一个节点；E 是边集，$E=\{e|e=<n_i,n_j>, n_i,n_j\in N\}$，$n_i$ 称为 n_j 的直接前驱节点，n_j 称为 n_i 的直接后继节点；$n_{in},n_o\in N$，n_{in} 是程序 P 的起始节点，n_o 是程序 P 的终止节点。

迁移系统　用一个三元组 $M=(S,\rightarrow,L)$ 表示，其中 S 为状态集合，$\rightarrow$ 表示迁移关系，L 为标记函数；$L: S\rightarrow P(\text{Atoms})$，其中 Atoms 是待验证系统原子公式的集合，$P(\text{Atoms})$ 表示 Atoms 的幂集，每个状态 s 都有其对应的原子命题集合 $L(s)$。

迁移系统状态　迁移系统状态是一个二元组 $<l,V>$，其中 l 表示程序执行位置，$V=\{v_1,\cdots,v_n\}$ 表示变量的集合，其中每个 $v_i(1\le i\le n)$ 代表程序中的一个变量，各对应一个有限域 D_{vi}。

在以上定义的基础上，表 1 给出采用深度遍历控制流图的方式得到代码对应的迁移系统的算法，其中 $s[i]$ 为迁移系统的状态，$l[i]$ 为程序第 i 条指令，$V[i]$ 为状态 $s[i]$ 处程序所有变量的值，m 为迁移系统中状态编号，TR 是迁移关系集合，tr∈TR, ST 为暂存状态的栈，初始为空，flag 用于标记状态是否已被访问，为 0 表示还没被访问，为 1 表示已经被访问。

算法从代码入口地址开始，将第 1 条指令记为 $l[0]$，计算该指令处变量情况，记为 $V[0]$，以 $<l[0], V[0]>$ 作为迁移系统的第 1 个状态并入栈；之后循环执行循环体中指令(对应表 1 中的 9-20 行)，将当前栈顶元素弹出，首先判断该状态是否被访问，如果被访问则取下一个栈顶元素，否则将该状态标记为已访问，找到该状态中指令的所有直接后继指令，分析对应指令处的变量情况，然后将指令和对应指

表 1　迁移系统生成算法

(1)	输入：微控制器二进制反汇编代码 P
(2)	输出：迁移系统，包括状态集合 S 和状态迁移关系集合 TR
(3)	**begin**
(4)	$s[0]=<l[0], V[0]>$;
(5)	$s[0]$.flag=0;
(6)	**push**(ST, $s[0]$);　　//将 s_0 压入栈 ST
(7)	m=0;
(8)	**While** ST !=Null **do**
(9)	**begin**
(10)	CurrentState=POP(ST);
(11)	**if** (CurrentState.flag=1) **then**
(12)	Continue( );
(13)	CurrentState.flag=1;
(14)	**foreach** l, $l\in\{l|e=<l_{\text{CurrentState}}, l>\}$**do**// $l_{\text{CurrentState}}$ 表示当前状态对应的指令
(15)	计算 l 对应的 V;
(16)	m++;
(17)	$s[m]=<l, V>$;
(18)	tr= CurrentState→$s[m]$;
(19)	**push**(TR, tr);
(20)	**push** (ST, $s[m]$);
(21)	**end**
(22)	**end**

令处的变量组合成新的状态，添加到栈中，并将当前状态和所有新生成状态组成迁移关系对，加入迁移关系集合 TR。当栈为空时，算法结束，得到代码对应的迁移系统。该算法给出了由微控制器二进制代码得到迁移系统的基本步骤，可以发现生成的状态数量与程序代码密切相关，通过对代码进行化简可以缩减生成的迁移系统规模。

3　敏感位置识别技术

通过对代码进行化简可以缩减生成模型中状态的数量，本节提出了结合子程序摘要信息的敏感位置识别算法对代码进行化简。首先简要分析了微控制器代码特点，给出敏感变量和敏感位置的定义；然后从待验证性质出发，从性质中提取敏感变量，并根据敏感变量利用结合子程序摘要信息的敏感位置识别算法识别程序中的敏感位置；构建模型过程中只需要将敏感位置处的代码转换为迁移系统，这样既可以保证对性质的正确验证，同时能够大大缩减生成的系统规模。

3.1　敏感位置的定义和识别

不同于计算机上层程序，微控制器代码具有硬件相关性，代码行为直接决定了电路的功能，而模型检验的待验证性质根据电路功能得到[12]。微控制

器代码直接作用于硬件端口上，通过输入/输出端口与外界交互信息控制电路实现功能，而端口在微控制器存储器空间中以内存映射的形式出现，在代码中将其作为变量处理[13]，可见待验证性质与代码中变量密切相关，因此相对于计算机上层程序，从微控制器代码中提取与待验证性质相关的变量更为直接[14]。

程序包含多个变量，而模型检验要验证的性质可能只涉及到其中的部分变量，如果不对代码进行简化处理，则在应用第 2 节的模型生成算法生成迁移系统的过程中，要对所有指令进行操作，生成的很多状态对于性质的验证并没有帮助。如图 1 所示，是台灯控制程序的部分代码[2] ，其中 PINB 表示按键，PORTA 的值表示灯的状态：PORTA=55H 时，表示灯处于弱光线状态，PORTA=00H 时表示处于开启状态，PORTA=0FFH 时表示处于关闭状态。台灯的正常功能是当处于关闭状态时，按下一次按钮，台灯进入弱光线状态；如果在两秒钟内连续按了两次，台灯进入开启状态；如果两次按键时间超过两秒钟，则台灯熄灭，回到关闭状态，所有操作由微控制器代码控制。根据台灯的功能描述我们可以很容易得到该系统的正常功能，如台灯处于弱光线状态时按下了按键，则台灯会进入开启或关闭状态，对应待验证性质表示为 AG((PORTA=55H & ButtonPressed=1)→A(PORTA=55H)U(PORTA=0FFH|PORTA=00H))。可以发现待验证性质只涉及到了代码中的 PORTA 和 ButtonPressed 两个变量，粗斜体部分代码对性质的验证没有影响，在生成迁移系统的过程中可以不对这些代码进行转换，从而可以大量减少生成的状态数量。

在对微控制器代码进行模型检验过程中，可以根据待验证性质中的原子公式提取代码中对应的变量，如“寄存器 $R0$ 为 0”对应于寄存器变量 $R0$，“端口 0 输入 55H”对应于变量 Port0，为表示变量 x 与原子公式 Ω 的对应关系，引入符号≙，$x \triangleq \Omega$ 表示变量 x 对应于原子公式 Ω 。为验证某一性质，在对代码进行建模时只需要关注与性质中变量相关的那些代码[10]。为了便于后续分析，引入以下定义。

首先为控制流图中每个节点引入以下集合：

(1)定义集 Def(n)：在节点 n 处出现且值被改变的变量集合；

(2)引用集 Ref(n)：在节点 n 处出现且参与运算的变量集合。

定义 1 执行路径 由若干条顺序执行的语句组成的序列称为一条执行路径，$\mathrm{Path}_{ij}=\{n_i,n_{i+1},\cdots,n_j\}$，$i<j$，表示从节点 i 到节点 j 的执行路径，令 $i<=k<j$,$<n_k,n_{k+1}>\in E$。

```
        PINB BIT P2.1
        PORTA EQU P1
        ButtonPressed EQU 30H
        ORG 00H
MAIN:   ……
        MOV R4, #0
        SETB PINB
FIRST:  JB PINB, NEXT
        ACALL DELAY
        JNB PINB, FIRST
NEXT:   MOV #ButtonPressed, #1
WEAK:   CJNE #ButtonPressed, #1, MAIN
        CJNE R4, #0, SHOW
        MOV PORTA, #55H
        MOV R4, #1
        MOV TMOD, #01
        MOV TL0, #1AH
        MOV TH0, #0CFH
        SETB TR0
        SJMP OVER
SHOW:   CJNE R4, #1, DOWN
        JNB TF0, MIDS
        MOV PORTA, #FFH
        MOV R4, #0
        SJMP MIDO
MIDS:   MOV PORTA, #0
        MOV R4, #2
MIDO:   CLR TR0
        SJMP OVER
DOWN:   CJNE R4, #2, OVER
        MOV PORTA, #0FFH
        MOV R4, #0
OVER:   MOV #BottonPressed, #0
        SJMP MAIN
DELAY:  MOV R1, #105
HERE2:  MOV R2, #60
HERE1:  DJNZ R2, HERE1
        DJNZ R1, HERE2
        RET
        END
```

图 1 台灯控制部分示例代码

定义 2 数据依赖 设 n_i, n_j 为 CFG 的两个节点，v 为一个变量，n_j 关于变量 v 数据依赖于 n_i，记为 $n_j \xrightarrow{\mathrm{DDv}} n_i$ 当且仅当：

(1)$v\in\mathrm{Def}(n_i)$，即语句 n_i 对 v 进行了定义；

(2)$v\in\mathrm{Ref}(n_j)$，即 v 在语句 n_j 处被使用了；

(3)存在执行路径 Path_{ij}，且对于任意 $i<k<j$, v 不属于 $\mathrm{Def}(n_k)$。

定义 3 直接控制依赖 设 n_i, n_j 为 CFG 的两个节点，n_i 为分支节点，n_j 直接控制依赖于 n_i，记

为 $n_j \xrightarrow{\text{CD}} n_i$，满足以下条件：

(1)存在路径 Path_{ij}；

(2)对于任意 n_k, $n_k \in \text{Path}_{ij}$ 且 $k \neq i$, n_j 是 n_k 的后必经节点；

(3) n_j 不是 n_i 的后必经节点。

定义 4　敏感变量　程序中与待验证性质相关的变量称为敏感变量。敏感变量构成的集合称为敏感变量集。设待验证性质为 φ, φ 对应原子公式集为 $\text{Atom}(\varphi)$，则敏感变量集 $S_v = \{x \mid x \triangleq \Omega,\ \Omega \in \text{Atom}(\varphi)\}$。

定义 5　敏感位置　代码中能够直接或者间接作用于敏感变量的语句称为敏感位置。敏感位置构成的集合称为敏感位置集。设敏感变量为 sv，则敏感位置集 $S_l = \{n \mid sv \in \text{Def}(n)\} \cup \{n \mid n \xrightarrow{\text{DDv}} sn,\ sn \in S_l\} \cup \{n \mid n \xrightarrow{\text{CD}} sn,\ sn \in S_l\}$。

敏感位置集可以通过程序切片技术来得到。程序切片是一种重要的程序分析技术，每个程序切片对应一个切片准则 $<n, V>$，其中 n 是程序的某条指令，V 是所关注的变量集合。程序 P 的切片 $S(n,V)$ 由对语句 n 中变量 V 产生影响的所有语句组成。

为得到需要的敏感位置，首先根据敏感变量得到切片准则，然后依照切片准则采用切片算法得到每个切片准则对应的敏感位置集，最终关注的敏感位置是这些集合的并集。设 v 为某个敏感变量，遍历程序控制流找到 v 被定义的所有位置集 N'，$N' = \{n \mid v \in \text{Def}(n)\}$，进而得到切片准则集 $\text{SC}=\{<n, v> \mid n \in N'\}$，根据切片准则集中的每条准则对程序进行切片分别得到对应的敏感位置集，进而求得这些位置集的并集。程序切片有很多算法，主流的有基于数据流的算法，基于程序依赖图的图形可达性算法等。本文采用 Weiser 的基于数据流的方法对目标二进制程序进行切片，通过对以下两个步骤进行迭代来求得程序的最少切片：(1)求解直接相关变量和语句；(2)计算间接相关变量和语句[15,16]。

3.2 结合子程序摘要信息的敏感位置识别算法

在微控制器汇编代码中，大量使用子程序，主程序通过 call 语句来调用子程序。单个子程序一般实现某个特定的会被多次执行的任务，子程序和主程序使用共同的变量，主程序会在调用子程序前，对子程序中需要使用的变量进行初始化，在子程序执行完成后，子程序对变量的作用结果也可以被主程序直接使用。在用程序切片技术求解敏感位置集的过程中，对于 call 语句，需要用对应子程序代码替换 call 语句继续进行切片。如果某个子程序不包含敏感变量或敏感变量的相关变量，则无需对其进行切片分析。针对这一问题可以预先对子程序进行分析，考察其是否与敏感变量相关，为此我们引入子程序摘要的概念。

定义 6　子程序摘要　用一个三元组 Summary $=<l_{\text{call}}, V_{\text{refc}}, V_{\text{defc}}>$ 表示，l_{call} 为 call 语句的位置，V_{refc} 为在子程序中被使用的变量集，V_{defc} 为经过子程序操作后被定义的变量集。与 call 指令 i 对应的子程序摘要记为 Summary(i)。

子程序的摘要信息可以通过过程内分析得到，在识别微控制器二进制代码的敏感位置集的过程中考虑子程序的摘要信息，可以提高识别敏感位置集的处理效率，缩短识别时间。根据二进制文件生成控制流图的过程中对 call 指令进行标记，同时以它的下一条指令作为其直接后继节点。在敏感位置集识别过程中，如果发现某条 call 语句调用的子程序摘要信息的定义变量包含其直接后继节点中关于切片的相关变量或者敏感变量本身，则将子程序替换 call 指令继续进行切片处理，否则不对子程序进行处理。

设敏感变量集为 V，切片准则集合 $\text{SC}=\{<l, v> \mid v \in V\}$，结合子程序摘要信息的敏感位置识别算法如表 2 所示。

表 2　结合子程序摘要信息的敏感位置识别算法

(1)	输入：对微控制器目标二进制代码反汇编得到的汇编代码，切片准则集合 SC
(2)	每个子程序的摘要信息 Summary
(3)	输出：敏感位置集 SP
(4)	**foreach** $<l, v> \in \text{SC}$ **do**
(5)	$R_C^0(l) = v$
(6)	**foreach** $<i, j> \in E$ **do**
(7)	$R_C^0(i) = R_C^0(i) \cup \{v \mid v \in R_C^0(j), v \notin \text{Def}(i)\} \cup \{v \mid v \in \text{Ref}(i), \text{Def}(i) \cap R_C^0(j) \neq \varnothing\}$
(8)	**if** (i是call指令 $\wedge$ (summary(i) $\cap (R_C^0(j) \cup V) \neq \varnothing$))
(9)	删除 i 到 j 的边；
(10)	将 i 的直接后继设为子程序第 1 条语句；
(11)	将子程序中 Reti 语句的直接后继设为 j；
(12)	**While** $S_C^k \neq S_C^{k+1}$ **do**
(13)	$R_C^{k+1}(i) = R_C^k(i) \cup \bigcup_{b \in B_C^k} R^k_{(b,\text{Re}f(b))}(i)$
(14)	**if** (i是call指令 $\wedge$ (summary(i) $\cap (R_C^k(j) \cup V) \neq \varnothing$))
(15)	删除 i 到 j 的边；
(16)	将 i 的直接后继设为子程序第 1 条语句；
(17)	将子程序中 Reti 语句的直接后继设为 j；
(18)	$S_C^{k+1} = B_C^k \cup \{i \mid \text{Def}(i) \cap R_C^{k+1}(j) \neq \varnothing, <i, j> \in E\}$
(19)	得到对于准则 $<l, v>$ 的敏感位置集 TS $<l, v>$
(20)	$\text{SP} = \bigcup_{<l,v> \in \text{SC}} \text{TS} <l, v>$

算法中(5)-(11)行用于求解与切片准则直接相关变量和语句，(12)-(18)行递归计算间接相关的变量和语句，直至语句集合不再变化。其中(8)-(11)行和(14)-(17)行都是对子程序的处理，第(8)行和第(14)行用于判断 call 指令对应的子程序摘要信息中是否包含后继节点中关于切片的相关变量或敏感变量本身。

应用模型检验构建模型的过程中面临状态爆炸的问题，而生成的模型状态数量与代码规模密切相关，首先应用本节提出的结合子程序摘要信息的敏感位置识别算法对代码进行化简，之后采用第 2 节的模型生成算法对化简后的代码生成模型，可以有效缩减生成的模型中的状态数量。

4 实验与分析

为了测试本文提出方法的有效性和性能，我们选取了 8 款实际运行的不同硬件应用环境下、不同类型微控制器的二进制代码进行了测试，表 3 给出了这些测试用例的简单描述[1,13]。对于每个代码都根据其具体的应用电路要求提取一条验证性质，用 CTL 公式表示，然后从中提取各个程序中对应的敏感变量，如台灯控制程序验证性质的公式为 AG((PORTA=55H&ButtonPressed=1)→A(PORTA=55H)U(PORTA=0FFH|PORTA=00H))，对应两个敏感变量 PORTA 和 ButtonPressed。实验环境为 Dell Optiplex 360 计算机，内核为 Intel(R) Core(TM) i5 2.67 GHz，主存 2 G，操作系统为 Windows XP SP3。

表 4 显示了对 8 组二进制代码文件进行测试的结果，分别测试了不对代码进行优化和采用第 3 节优化算法后生成的状态数量和处理时间。其中第 2 列显示了各个二进制代码文件经反汇编的得到的汇编代码条数，第 3 列“直接转换状态数”是指不对代码进行优化直接采用模型生成算法得到的迁移系统状态数，第 4 列“处理时间”是不对代码进行优化应用模型生成算法所用的时间，第 5 列“优化操作后状态数”是指采用第 3 节的结合子程序摘要信息的敏感位置识别算法后得到的敏感位置程序转换为迁移系统生成的状态数，最后一列是采用第 3 节优化算法优化及状态生成时间。

通过对比可以发现，采用本文提出的基于敏感位置识别的状态缩减技术可以大量缩减生成的状态数，优化率最小为 45.8%，最大的达到 91.5%，转化时间也有很大程序的减少。

表 3 测试代码描述

序号	代码名称	微控制器名称	微控制器类型	功能描述
1	Desk_lamp.bin	8051	Intel MCS-51 系列	台灯工作控制代码
2	Traffic_light.bin	8051	Intel MCS-51 系列	交通灯工作控制代码
3	Air controller.bin	8051	Intel MCS-51 系列	遥控器控制代码
4	Mix.bin	8051	Intel MCS-51 系列	自己设计的混合控制代码
5	Clock.hex	ATmega 16	Atmel AVR 系列	电子表控制代码
6	Motor.hex	ATmega 16	Atmel AVR 系列	步进电机控制代码
7	Voter.hex	PIC16F876	Microchip PIC 系列	表决器控制代码
8	Elevator.hex	PIC16F876	Microchip PIC 系列	简易电梯控制代码

表 4 基于敏感位置识别技术生成状态结果

文件名	反汇编后代码条数	直接转换状态数	处理时间(s)	优化操作后状态数	优化及状态生成时间(s)
Desk_lamp.bin	191	7412	43	2982	18.1
Traffic_light.bin	379	98396	534	37539	155.2
Air_controller.bin	147	3567	19	1493	9.4
Mix.bin	577	1054796	4562	89487	435.7
Clock.hex	294	385303	1421	87462	341.8
Motor.hex	185	13725	57	7438	29.3
Vetor.hex	238	197639	687	71309	249.8
Elevator.hex	427	1876432	7143	287549	978.3

在敏感位置集识别过程中，本文算法通过提取子程序摘要信息对子程序进行化简处理，如果子程序中不含有与敏感变量相关的变量，则对应子程序段不进行处理，可以减少敏感位置集识别时间。选用 Mix.bin 二进制文件对算法利用子程序摘要信息在敏感位置集识别时间上的优化效果进行了测试，通过分析该文件的反汇编代码发现其中有 21 个与待验证性质中敏感变量无关的子程序，对汇编代码进行修改，分别将子程序代码代入主程序对应的 call 指令处，然后采用结合子程序摘要信息的敏感位置识别算法对这些修改后的汇编代码进行处理，以识别敏感位置集，测试其识别时间，结果如图 2 所示，call_i 代表针对 21 个与敏感变量无关的子程序，在主程序中有 i 个 call 指令没有被对应的子程序代码替换。

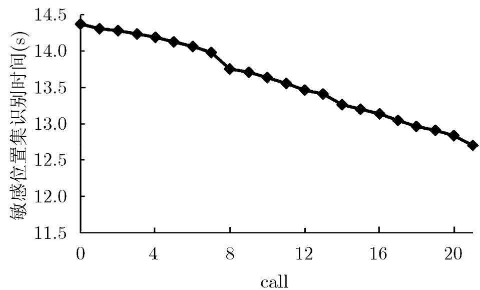

图2 结合子程序摘要信息的敏感位置识别算法对子程序处理的效果

通过分析图 2 可以发现，结合子程序摘要信息对敏感位置进行识别，可以减少转化时间，对于与敏感变量无关的子程序，当将所有 call 指令用对应子程序代码替换后，相当于程序中不含与敏感变量无关的子程序，这种情况下，子程序摘要信息对敏感位置识别没有优化作用，此时识别时间为 14.37 s；当程序中与敏感变量无关的子程序逐渐增多时，子程序摘要信息对敏感位置的识别的优化作用得到体现，与敏感变量无关的子程序越多，敏感位置识别时间越短，当程序包含全部的 21 个与敏感变量无关的子程序时，敏感位置识别时间最短，为 12.72 s。实验结果充分说明了利用子程序摘要信息对敏感位置集识别的优化效果。

5 总结

微控制器作为嵌入式系统中一类广泛使用的核心处理器，其安全性至关重要，随着信息安全研究逐渐深入系统底层，微控制器代码的安全受到更多关注。本文以微控制器二进制代码为研究对象，重点解决微控制器二进制代码模型检验过程中的状态爆炸问题。根据二进制代码控制流，结合数据流分析提出了微控制器二进制代码模型生成算法，生成模型的状态数量与代码的规模密切相关。模型检验待验证的性质，只与部分代码相关，很多代码与性质的验证没有关系，因此可以首先对代码进行化简，只对简化后的代码进行建模，实现状态的缩减。分析微控制器代码的特点，发现待验证性质与代码中变量密切相关，提出了敏感变量和敏感位置的概念，以此为基础提出了结合子程序摘要信息的敏感位置识别算法，从模型检验待验证性质出发提取敏感变量，并根据敏感变量对微控制器代码进行分析，识别敏感位置，只对敏感位置对应代码进行建模，从而缩减生成的状态空间。最后通过实验对本文提出的方法进行了验证，结果表明本文提出的方法能够有效缓解模型生成过程中的状态爆炸问题。

参 考 文 献

[1] Bastian S, Jorg B, and Stefan K. Application of static analyses for state-space reduction to the microcontroller binary code[J]. *Science of Computer Programming*, 2011, 76(2): 100–118.

[2] Bastian S and Stefan K. Model checking C source code for embedded systems[J]. *International Journal on Software Tools for Technology Transfer*, 2009, 11(3): 187–202.

[3] Thomas R, Jorg B, Martin H, *et al*.. Past time LTL runtime verification for microcontroller binary code[C]. The 16th International Conference on Formal Methods for Industrial Critical Systems(FMICS'11), Trento, Italy, 2011: 37–51.

[4] 贾仰理, 李舟军, 邢建英, 等. 基于模型检验的构件验证技术研究进展[J]. 计算机研究与发展, 2011, 48(6): 913–922.
Jia Yang-li, Li Zhou-jun, Xing Jian-ying, *et al*.. Advances in the component verification technology based on model checking[J]. *Journal of Computer Research and Development*, 2011, 48(6): 913–922.

[5] Moonzoo K, Yunho K, and Hotae K. A comparative study of software model checkers as unit testing tools: an industrial case study[J]. *IEEE Transactions on Software Engineering*, 2011, 37(2): 146–160.

[6] Bastian S. Model checking of software for microcontrollers [D]. [Ph.D. dissertation], RWTH Aachen University, Aachen, Germany, 2008.

[7] Thomas W R, Junghee L, Aditya V T, *et al*.. There's plenty of room at the bottom: analyzing and verifying machine code[C]. The 22nd International Conference on Computer Aided Verification (CAV10), Edinburgh, UK, 2010: 41–56.

[8] 李兴锋, 张新常, 杨美红, 等. 基于 SPIN 的模块化模型检测方法研究[J]. 电子与信息学报, 2011, 33(4): 902–907.
Li Xing-feng, Zhang Xin-chang, Yang Mei-hong, *et al*.. Study on modularized model checking method based on SPIN[J]. *Journal of Electronics & Information Technology*, 2011, 33(4):

902–907.

[9] Alessandro C and Alberto G. Software model checking via IC3[C]. The 24th International Conference on Computer Aided Verification (CAV2012), Berkeley, CA, USA, 2012: 277–293.

[10] John H, Matthew B D, and Hongjun Z. Slicing software for model construction[J]. *Higher-Order and Symbolic Computation*, 2000, 13(4): 315–353.

[11] Karen Y and Orna G. Static analysis for state-space reductions preserving temporal logics[J]. *Formal Methods in System Design*, 2004, 25(1): 67–96.

[12] Gao Hong-bo, Li Qing-bao, Zhu Yu, *et al*.. Code-controlled hardware trojan horse[C]. The 3th International Conference on Information Computing and Applications, Chengde, China, 2012: 171–178.

[13] MacKenzie I S. The 8051 Microcontroller[M]. Third Edition, Englewood Cliffs: Prentice Hall, 1998: 19–21.

[14] Thomas N and Bastian S. Delayed nondeterminism in model checking embedded systems assembly code[C]. The 3rd International Haifa Verification Conference on Hardware and software: Verification and Testing (HVC2007), Haifa, Israel, 2008: 185–201.

[15] 李必信. 程序切片技术及其应用[M]. 北京: 科学出版社, 2006: 20–35.
Li Bi-xin. Program Slicing Technique and Its Applications[M]. Beijing: Science Press, 2006: 20–35.

[16] Jose B B, Daniela C, Pedro R H, *et al*.. Assertion-based slicing and slice graphs[J]. *Formal Aspects of Computing*, 2012, 24(2): 217–248.

高洪博： 男，1984年生，博士生，研究方向为信息安全与可信计算.

李清宝： 男，1967年生，博士，教授，研究方向为信息安全与可信计算.

王　炜： 男，1975年生，博士，研究方向为信息安全.

朱　瑜： 女，1986年生，硕士，研究方向为信息安全.