https://doi.org/10.1051/jnwpu/20224051180

基于 PROMELA 模型的安全关键软件形式化验证技术

邢亮[1], 丁成钧[2], 杜虎鹏[2], 马春燕[2]

(1.航空工业西安航空计算技术研究所, 陕西 西安 710076; 2.西北工业大学 软件学院, 陕西 西安 710072)

摘 要:聚焦安全关键软件,研究基于 PROMELA 形式模型验证 C 程序中违反断言、数组越界、空指针解引用、死锁及饥饿等 5 类故障技术。建立 C 程序抽象语法树节点到 PROMELA 模型,验证属性相关函数到 PROMELA 模型的 2 类映射规则;根据映射规则提出由 C 程序自动生成 PROMELA 形式模型的算法,并对算法进行理论分析;针对 C 程序中 5 种故障类型,分别给出基于 PROMELA 模型的形式化验证方法,并分析验证的范围;覆盖各类故障的验证范围,为每类故障类型选取 12 个 C 程序案例进行实证研究,实验结果证明了方法的有效性。

关 键 词:C 程序;PROMELA 模型;软件故障;形式化验证
中图分类号:TP31 **文献标志码**:A **文章编号**:1000-2758(2022)05-1180-08

当前安全关键软件在航空、航天、医疗、交通等领域应用广泛,该类软件存在资源竞争、空指针和数组越界等问题,而检测相应故障主要依赖开发人员的经验和技术能力,保障其可靠性存在很大困难。形式化验证[1]是故障检测技术之一,其通过严格的数学语义进行推理以保障代码的可靠性。

形式化建模语言 PROMELA 支持对任务关键 C 程序逻辑进行建模以及 C 程序的嵌入,具有验证 C 程序的优势。目前,用 PROMELA 建模 C 程序时,主要通过人工方式实现,需要较高技术门槛和代价,限制了安全关键软件进行形式化验证的广泛应用。本文提出了从 C 程序到 PROMELA 模型的自动生成方法,实现对安全关键 C 程序中 5 类故障模式进行一键式形式化验证。本文贡献如下:

1) 制定了 C 程序到 PROMELA 模型 2 类映射规则,提出了 PROMELA 模型自动生成的方法,并对方法进行理论分析;

2) 针对 C 程序中违反断言、数组越界、空指针解引用、死锁以及饥饿 5 类故障,给出形式化验证方法和验证范围;

3) 研制了辅助工具,覆盖每类故障的验证范围,分别选取 12 个来源于开源代码托管平台 GitHub 的 C 程序案例,进行实证研究。

1 相关工作

文献[1]提出了针对并发故障的形式化验证方法,实现了 PROMELA 模型生成工具 C2Spin。Jiang[2]使用语法导向技术,采用 PROMELA 建模 C 程序,但缺少并发 C 程序库函数的建模。Wagner 等[3]提出检测缓冲区溢出的方法,但其检测精度较低。文献[4]提出基于插桩的方法验证 C 程序数组越界,但存在重复和等价插桩点,验证效率较低。Klieber 等[5]提出针对缓冲区溢出的自动化修复技术。文献[6]将随机内存分配算法和虚拟内存相结合,改进内存分配策略,避免程序执行时的内存错误。文献[7]综合程序切片和谓词抽象技术,提出 C 程序断言的静态验证方法。Yong 等[8]设计了用于

收稿日期:2021-11-15 **基金项目**:航空科学基金(20185853038,201907053004)资助
作者简介:邢亮(1983—),航空工业西安航空计算技术研究所高级工程师,主要从事机载软件技术研究。
通信作者:马春燕(1978—),西北工业大学副教授,主要从事软件自动化测试与故障定位研究。
e-mail:machunyan@ nwpu.edu.cn

检查空指针解引用的 C 程序安全工具。Zhe 等[9]通过代码插桩技术,设计针对内存漏洞的分析工具,但仅能处理较简单的 C 程序。

目前,大部分方法存在过多人工干预,并且验证的故障类型较少,也未给出存在故障的反例路径。PROMELA 对有限状态分布式系统进行建模[10]。SPIN 工具支持 PROMELA 模型中以断言形式和线性时间逻辑(linear-time logic,LTL)公式形式撰写的验证属性,生成并仿真执行验证程序[11]。

本文基于 PROMELA 模型,结合 SPIN,提出 C 程序"一键式"形式化验证技术,实现 5 类故障模式的验证,并给出 C 程序存在故障的反例路径。

2　基于 PROMELA 模型的安全关键软件形式化验证方法

2.1　C 程序到 PROMELA 模型的映射机制

2.1.1　C 程序到 PROMELA 模型的映射规则

本节定义了 C 程序抽象语法树节点(C 程序语法结构)到 PROMELA 模型的 19 个映射规则,部分规则如表 1 所示。

表 1　抽象语法树节点到 PROMELA 模型的映射规则

编号	抽象语法树节点	映射规则
R1	函数定义 FuncDef	函数名(函数参数){…}映射为 inline 函数名(函数参数){…}

R3	循环 While	while(循环条件){
		循环语句
		if(终止条件){break;}
		}
		映射为
		do
		::循环条件->循环语句
		::终止条件->break
		od

R4	Break	

R6	判断 If	if(判断条件){
		执行语句
		}
		映射为
		if
		::判断条件->执行语句
		fi

续表 1

编号	抽象语法树节点	映射规则
R10	循环 For	for(表达式 expr1;表达式 expr2;表达式 expr3){…} 映射为 表达式 expr1; do ::表达式 expr2->循环体内语句 expr;表达式 expr3; ::! 表达式 expr2->break od

由于篇幅所限,本文仅阐述循环节点 For 到 PROMELA 模型的映射规则:

1) 识别 For 循环体,生成 do...od 结构;

2) 递归遍历 For 第 1 个子节点,生成 PROMELA 模型,将返回值作为语句插入 do...od 结构前;

3) 递归遍历 For 第 2 个子节点,生成 PROMELA 模型,将返回值作为 do...od 中条件语句;

4) 对 For 第 2 个子节点的返回值取反,生成 PROMELA 模型,作为 do...od 结构中条件语句;

5) 递归遍历 For 第 4 个子节点,生成 PROMELA 模型,将返回值作为第 1 个条件语句后的语句;

6) 递归遍历 For 的第 3 个子节点,生成 PROMELA 模型,将返回值按顺序作为第 1 个条件语句后的语句;

7) 为第 2 个条件语句添加 break。

2.1.2　与验证属性相关的函数到 PROMELA 模型的映射规则

基于 2.1.1 生成的 PROMELA 模型,可以实现对断言违反、数组越界、空指针解引用的验证。为验证死锁、饥饿并发故障,本文进一步提出 10 条映射规则,部分规则如表 2 所示。

表 2　部分函数节点到 PROMELA 模型的映射规则

编号	函数节点	映射到 PROMELA 的结果
R20	main	active proctype p_main()
R21	线程	bool lck_p_线程名_ret; bool lck_p_线程名; chan ret_p_线程名=[1] of {pid}; chan req_cll_p_线程名=[1] of {pid}; chan exc_cll_p_线程名=[0] of {pid}; active proctype p_线程名()

续表 2

R22	pthread_create thr_create	bool lck_p_函数名_ret; bool lck_p_函数名; chan ret_p_函数名=[1] of {pid}; chan req_cll_p_函数名=[1] of {pid}; chan exc_cll_p_函数名=[0] of {pid}; active proctype p_函数名()

R23	thr_suspend
	thr_continue
		bool suspended;
		suspended = 1; //挂起
		suspended = 0; //继续
		在进程声明后添加 provided
		(! suspended)

R24	pthread_mutex_t	int
R25	mutex_lock	atomic {互斥变量名==0->互斥变量名=1}
	mutex_unlock	互斥变量名=0
	pthread_mutex_lock	atomic{互斥变量名==0->互斥变量名=1}
	pthread_mutex_unlock	互斥变量名=0

2.2 PROMELA 模型自动生成方法

算法 1 实现基于映射规则的 PROMELA 模型自动生成。1 至 15 行对抽象语法树进行递归遍历,获取当前节点及子节点信息,并根据节点名称,应用映射规则,生成 PROMELA 模型。16 至 26 行以 For 为例,给出其 PROMELA 模型生成算法。

算法 1 PROMELA model instance generator

input: t is the abstract tree root node of a C program; map is the mapping relationships between the abstract tree nodes and mapping rules

output: pr is an instance of PROMELA model

```
1.  while ( t ! = null) do
2.    if(map.containsValue(t.name)) then
3.      rule←map.get(t.name)
4.      instanceGenerator(rule);
5.    end if
6.    if t.char is "thr_suspend"
7.      return 1
8.    end if
9.    if t.char is "rand"
10.     return t.char
11.   end if
12.     return t.char
13.     otherwise do error
14.   end if
15. end while
16. Procedure instanceGenerator(rule)
17. if rule is R10 then
18.   PROMELAGeneration(node.firstchild)
19.   Emit("do")
20.   Emit("::")
21.   bool var = PROMELAGeneration(node.secondchild)
22.   Emit("! var->break")
23.   PROMELAGeneration(node.forthchild)
24.   PROMELAGeneration(node.thirdchild)
25.   Emit("od")
26.   end if
27.   else if rule is one of other rules then
28.   Generator the corresponding instance fragments
29. end procedure
```

2.3 PROMELA 模型生成方法的理论分析

从构建方法的语义保留、确定性和有边界 3 个方面,证明 C 程序到 PROMELA 模型生成方法的合理性。

首先借鉴文献[12]中提出的图同构思想,结合图 1 对构建方法的语义保留性进行证明。图 1 中 $Gram_c$ 指 C 语言的语法规则,$Gram_p$ 指 PROMELA 建模语言的语法规则,pro($Gram_c$)指 C 程序,pro($Gram_p$)指 PROMELA 模型实例。符号 h 表示 C 语言语法规则到 PROMELA 语法规则的映射;符号 r 表示 C 程序到 PROMELA 模型实例的转换函数;符号 t 和 t'分别表示 C 语言和 PROMELA 语言语法规则到程序实例之间的映射函数,它们为类型保留映射(type prserve mapping,TPM),符号 F 表示 C 语言语法规则到 PROMELA 模型实例之间的映射关系。

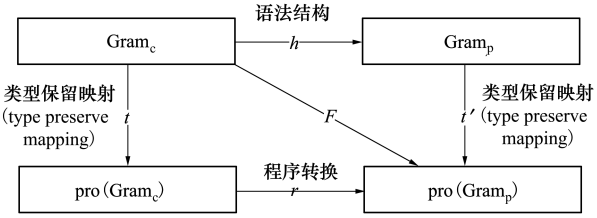

图 1 语法规则和程序实例之间的映射关系

2.1 节中 29 种映射规则是建立在形式化模型与抽象语法树节点语义一致的基础上,所以 C 程序抽象语法树节点到 PROMELA 模型的映射规则函数 h

是同构的。

定义 1　定义图同构函数 f 表示从图 G 到图 H 的转换:$f = G \rightarrow H$。$\boldsymbol{N}(G)$ 为图 G 的节点集合,$\boldsymbol{N}(H)$ 为图 H 的节点集合,$\boldsymbol{E}(G)$ 为图 G 的边的点集合,$\boldsymbol{E}(H)$ 为图 H 的边的集合。f 满足公式(1)中的约束。

$$\forall x,y \in \boldsymbol{N}(G) \wedge f(x),f(y) \in \boldsymbol{N}(H) \wedge xy \in \boldsymbol{E}(G) \Rightarrow f(x)f(y) \in \boldsymbol{E}(H) \quad (1)$$

结合定义 1 和本文的映射规则可知,图 G 类似 C 语言的语法规则,图 H 类似 PROMELA 语言的语法规则。而 $\boldsymbol{N}(G)$ 表示 C 语言抽象语法树节点,$\boldsymbol{E}(G)$ 表示映射规则中的 29 种抽象语法树节点,$\boldsymbol{N}(H)$ 表示 PROMELA 的语法节点集合,$\boldsymbol{E}(H)$ 表示 PROMELA 语法节点。

命题 1　$\mathrm{Gram_l}$ 为一种程序语言的语法规则,$\mathrm{pro}(\mathrm{Gram_l})$ 为该语言对应的程序实例,那么在语法规则及其程序实例之间的 TPM 函数是图同构的。

由命题 1,C 语言语法规则到 C 程序实例之间的映射函数 t,以及 PROMELA 语言到 PROMELA 模型实例之间的映射函数 t' 是图同构的。

命题 2　如果映射函数 f 是图同构的,并且映射函数 g 是图同构的,那么 f 和 g 的复合运算 $f \circ g$ 也是图同构的。

根据命题 2,因为函数 h,t' 是图同构的,所以 $F(h\ t')$ 是图同构的。

定理 1　已知 $\mathrm{Gram_c}$ 和 $\mathrm{Gram_p}$ 之间存在同构映射 h,$\mathrm{Gram_c}$ 和它的程序实例 $\mathrm{pro}(\mathrm{Gram_c})$ 之间存在类型保留映射函数 t,$\mathrm{Gram_p}$ 和它的模型实例之间 $\mathrm{pro}(\mathrm{Gram_p})$ 存在类型保留映射函数 t',则 $\mathrm{pro}(\mathrm{Gram_c})$ 和 $\mathrm{pro}(\mathrm{Gram_p})$ 之间的映射函数 r 是图同构的。

证明　上文已经证明函数 h,t,t',F 是图同构的,分别用公式(2)至(5)表示。

$$h(\mathrm{Gram_c}) = \mathrm{Gram_p} \quad (2)$$

$$t(\mathrm{Gram_c}) = \mathrm{pro}(\mathrm{Gram_c}) \quad (3)$$

$$t'(\mathrm{Gram_p}) = \mathrm{pro}(\mathrm{Gram_p}) \quad (4)$$

$$F(\mathrm{Gram_c}) = t'(h(\mathrm{Gram_c})) = t'(\mathrm{Gram_p}) = \mathrm{pro}(\mathrm{Gram_p}) \quad (5)$$

根据公式(1)和(2)可得公式(6):

$$\forall x,y \in \boldsymbol{N}(\mathrm{Gram_c}) \wedge h(x),h(y) \in \boldsymbol{N}(\mathrm{Gram_p}) \wedge xy \in \boldsymbol{E}(\mathrm{Gram_c}) \Rightarrow h(x)h(y) \in \boldsymbol{E}(\mathrm{Gram_p}) \quad (6)$$

根据公式(1)和(5)可得公式(7):

$$\forall x,y \in \boldsymbol{N}(\mathrm{Gram_c}) \wedge F(x),F(y) \in \boldsymbol{N}(\mathrm{pro}(\mathrm{Gram_p})) \wedge xy \in \boldsymbol{E}(\mathrm{Gram_c}) \Rightarrow F(x)F(y) \in \boldsymbol{E}(\mathrm{pro}(\mathrm{Gram_p})) \quad (7)$$

将公式(7)中的 F 替换成 $r \circ t$,化简得到公式(8):

$$\forall x,y \in \boldsymbol{N}(\mathrm{Gram_c}) \wedge r(x),r(y) \in \boldsymbol{N}(\mathrm{pro}(\mathrm{Gram_p})) \wedge xy \in \boldsymbol{E}(\mathrm{Gram_c}) \Rightarrow r(x)r(y) \in \boldsymbol{E}(\mathrm{pro}(\mathrm{Gram_p})) \quad (8)$$

根据公式(8),因为 TPM 函数 t 是图同构的,结合公式(1)表示的同构函数的约束,因此推断映射函数 r 也是图同构的,即定理 1 得证。

C 程序到 PROMELA 模型生成方法是基于 C 语言和 PROMELA 建模语言的语法规则提出的,而 C 语言和 PROMELA 语言的语法结构是精确无二义性的,且抽象语法树节点的映射规则是一一映射的,所以生成方法是确定性的。由于 C 程序的抽象语法树节点数量是有限的,所以生成方法是有限的。综上,本文提出的 C 程序到 PROMELA 模型生成方法是正确的。

2.4　基于 PROMELA 模型的 C 程序验证方法

1) 违反断言的验证方法

在 C 程序中添加的断言称为基本断言,以检查表达式的合法性。如果表达式中变量值确定,则断言表达式可以直接验证;如果变量值依赖函数参数、函数返回值或全局变量时,例如变量 var 的地址以实参形式传入函数 fun 中,则导致 C 程序转换成 PROMELA 模型后,断言语句 assert(var 小于 1 000)的逻辑值无法确定,针对变量无法确定取值的情况,本文提出将激励函数插入 PROMELA 模型的算法 2。

算法 2　Excitation function generation algorithm genStubs()

```
input: t is the abstract tree root node of a C program;
output: void
1.  while (t is not null)
2.    if (t is TN_FUNC_CALL) then
3.      line←t.cont
4.      fun_name←t.lnode.value
5.      params = t.rnode
6.      for(params hasNext())
7.        param = params.next()
8.        if param is ASSERT_VAR then
9.          param_range←param
10.           fun_name=fun_name+"_stubs"
11.           for num in param_range
```

```
12.                fun_body←num
13.                new_funcall←line,fun_name
14.              end for
15.            end if
16.          end for
17.        end if
18.    end while
```

表 3 是基于 C 程序断言的验证范围,对包含规约运算的情况作等价处理,例如,断言表达式 assert(*i*++>0)等价变换为 *i*++和 assert(*i*>0)即可。

表 3 C 程序断言验证方法的验证范围

编号	断言表达式类型	表达式值	验证措施
1	基本断言	静态确定	直接验证
2	基本断言	动态依赖全局变量	插入确定全局变量的激励函数
3	基本断言	动态依赖函数参数	插入确定参数的激励函数
4	基本断言	动态依赖函数返回值	插入确定返回值的激励函数
5	包含规约运算的断言	–	对规约运算作等价处理

2) 数组越界的验证方法

表 4 是 C 程序数组越界的验证范围。对于数组长度和索引均确定的情况,可以采用 1)中断言的形式进行验证。而当数组长度或索引依赖函数参数或全局变量时,则根据算法 2 插入激励函数;如果多维数组中每一维度长度和索引范围均静态确定,则插入断言 assert(index1<Len1 && index2<Len2),如果多维数组存在某个数组索引或长度不确定的情况,则调用算法 2 确定有关变量的值。

表 4 数组越界验证方法的验证范围

编号	数组维度	数组长度	数组索引	验证措施
1	1 维	静态确定	静态确定	直接验证
2	1 维	依赖全局变量	静态确定	插入确定全局变量的激励函数
3	1 维	依赖函数参数	静态确定	插入确定函数参数的激励函数
4	1 维	静态确定	依赖全局变量	插入确定全局变量的激励函数
5	1 维	静态确定	依赖函数参数	插入确定函数参数的激励函数
6	多维	–	–	同 1 维数组的验证方法

3) 空指针解引用的验证方法

本方法对空指针解引用类型的验证范围如表 5 所示。对于一级静态确定指针,可以采用断言表达式验证。指针变量动态依赖函数参数或全局变量需调用算法 2 生成静态确定指针变量的激励函数后再进行验证;多级指针与 1 级指针验证时插入的断言不同,假设二级指针 int * *p; 则插入断言 assert(p && *p),即需要对每一级指针进行验证。

表 5 空指针解引用验证方法的验证范围

编号	指针类型	指针变量	验证措施
1	1 级指针	静态确定	直接验证
2	1 级指针	依赖函数参数	插入确定参数值的激励函数
3	1 级指针	依赖全局变量	插入确定全局变量的激励函数
4	多级指针	静态确定	直接验证
5	多级指针	依赖函数参数	插入确定参数值的激励函数
6	多级指针	依赖全局变量	插入确定全局变量的激励函数

4) 死锁和饥饿的验证方法

死锁及饥饿问题验证是基于 C 程序 Pthread 库中线程创建函数 pthread_create、互斥锁以及条件变量等。死锁及饥饿问题验证范围如表 6 所示。

表 6 死锁及饥饿验证方法的验证范围

编号	函数	可以验证的特征范围
1	pthread_create	指定线程 id
2	pthread_create	指定线程运行地址
3	pthread_mutex_init	指定互斥量 mutex 初始化
4	pthread_mutex_lock	指定互斥量 mutex
5	pthread_mutex_unlock	
6	pthread_mutex_destroy	
7	pthread_cond_init	指定条件变量 cond
8	pthread_cond_wait	指定互斥量 mutex
9	pthread_cond_wait	指定条件变量 cond
10	pthread_cond_signal	指定条件变量 mutex
11	pthread_equal	指定线程 id
12	pthread_join	
13	pthread_cancel	
14	pthread_exit	实现线程终止

在并发 C 程序转换为 PROMELA 模型后,使用“end”标签来标识程序结束状态,如果“end”标签未

被执行,则存在死锁问题;使用“progress”标签标记重复被执行的语句,如果存在没有被执行的“progress”,即存在饥饿问题。

3　实验验证

3.1　某实验案例的验证过程

本文以存在数组越界故障的 Driver_receiv1553 程序为例,阐释验证原理。该程序从 1553b 总线接收数据,对数据有效性进行判断,读取数据长度、数据块等信息,对无效数据通过调用 trans16 函数进行处理。该 C 程序的流程图如图 2 所示。

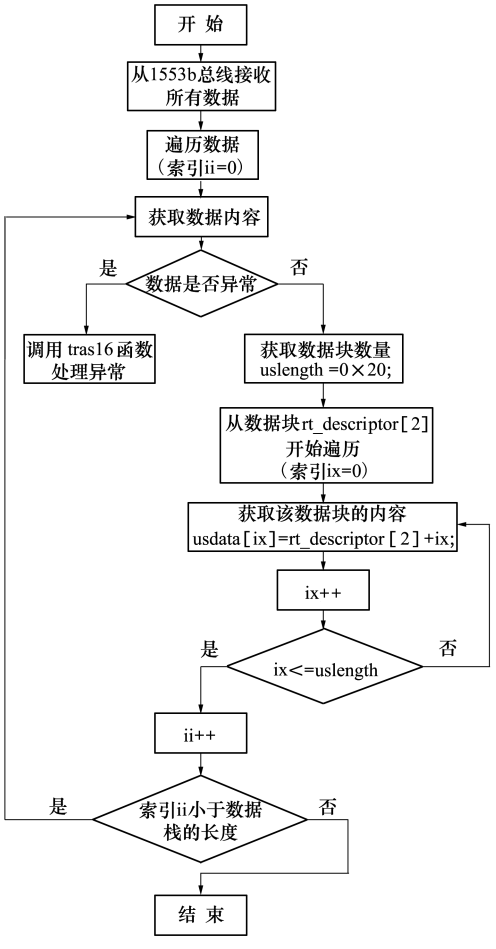

图 2　Driver_receiv1553 程序流程图

本文研制辅助工具完成 C 程序到 PROMELA 模型的转换,并与 SPIN 集成,实现对 C 程序的验证。辅助工具生成的 PROMELA 模型原理如图 3 所示。

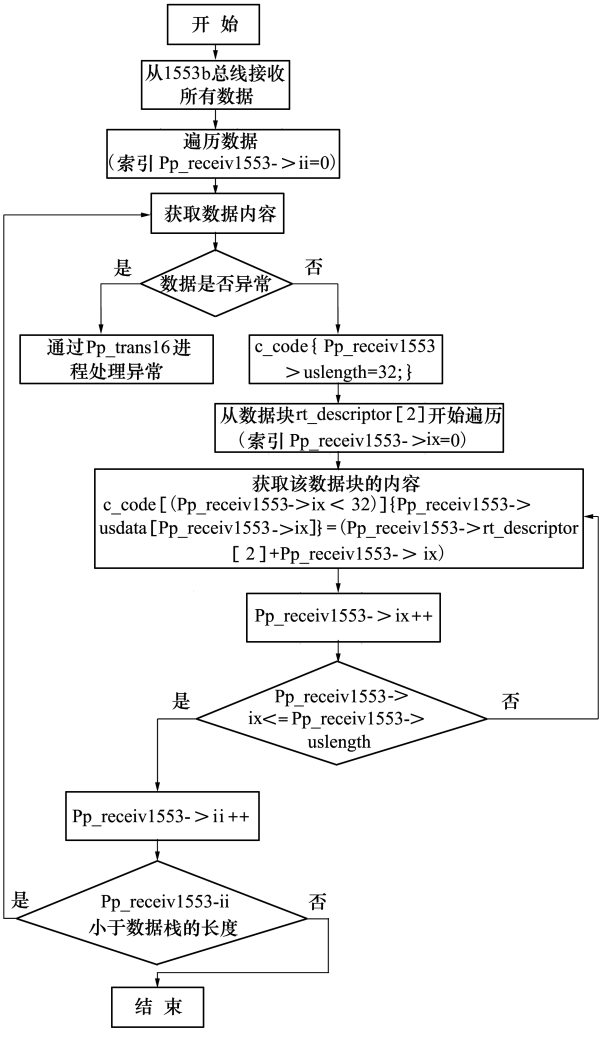

图 3　PROMELA 模型流程图

该模型的验证表达式为 c_code[(Pp_receiv1553->ix<32)],根据本文的验证方法,该程序存在数组越界故障,C 程序反例路径如图 4 所示。由于图 2 中变量 uslength 的值为 32,因此该有向图中的 108→109→108 路径会执行 32 次,此时 ix 的值为 32。而根据验证表达式,ix 的值应小于 32,所以该程序出现数组越界故障,且故障出现在 C 程序的第 109 行。

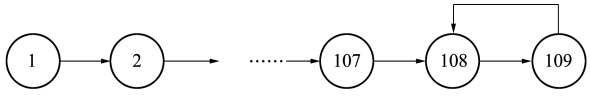

图 4　反例路径有向图示意

3.2　验证方法对比

本文针对违反断言、数组越界、空指针解引用、

死锁及饥饿,覆盖各故障类型的所有验证范围,分别使用 12 个来源于 GitHub 的 C 程序案例进行实验验证,并与相关工作中的其他方法进行对比,如表 7 所示,其中,验证故障类型中的编号为 2.4 节相应表中的编号。本文方法验证的故障类型多、自动化程度高,并展示 C 程序故障出现的反例路径,在这三个方面具有明显的优势。

表 7 C 程序验证方法对比

方法	验证的故障类型					验证的自动化	反例路径
	违反断言	数组越界	空指针解引用	死锁	饥饿		
C2Spin[1]	1	1	1,4	1,3,4,9,11 至 15,17,18	1,3,4,9,11 至 15,17,18	半自动化	-
CIL[2]	1	1	1,4	-	-	自动化	-
BOD[3]	-	1	-	-	-	自动化	-
Boundcheck[4]	-	1 至 5	-	-	-	自动化	-
ACR[5]	-	1	-	-	-	自动化	-
SET[8]	-	-	1,4	-	-	自动化	-
Modex+Spin[9]	1	1	1,4	1,3,4,9,11 至 15,17,18	1,3,4,9,11 至 15,17,18	半自动化	-
本文方法	1 至 5	1 至 6	1 至 6	1,3,4,6 至 9,11 至 15,17,18	1,3,4,6 至 9,11 至 15,17,18	一键式自动化验证	√

4 结 论

本文在定义 C 程序抽象语法树节点到 PROMELA 模型映射规则的基础上,增加了验证属性相关函数到 PROMELA 模型的映射规则,提出了一种 C 程序到 PROMELA 模型自动生成算法,并给出包含违反断言、数组越界、空指针解引用、死锁、饥饿这 5 种故障类型的 C 程序形式化验证方法。在对实验案例分析后,对比现有 C 程序形式化验证方法的优缺点,本文方法具有明显优势。

参考文献:

[1] 王大伟, 张大方, 缪力. 一种自动化模型检测 ANSI-C 程序的实用方法[J]. 计算机工程与科学,2010, 32(4): 79-82
WANG Dawei, ZHANG Dafang, MIAO Li. A practical method of automatic model checking ANSI-C program[J]. Computer Engineering and Science, 2010, 32(4): 79-82 (in Chinese)

[2] JIANG K. Model checking C programs by translating C to PROMELA[D]. Sweden: Uppsala University, 2009

[3] WAGNER D, FOSTER J S, BREWER E A, et al. A first step towards automated detection of buffer overrun vulnerabilities[C]//Network & Distributed System Security Symposium, 2000

[4] 李文明. C 程序内存安全性的运行时验证研究与实现[D]. 南京:南京航空航天大学
LI Wenming. Research and implementation of runtime verification of C program memory security[J]. Nanjing: Nanjing University of Aeronautics and Astronautics (in Chinese)

[5] KLIEBER W, MARTINS R, STEELE R, et al. Automated code repair to ensure spatial memory safety[C]//2021 IEEE/ACM International Workshop on Automated Program Repair, 2021

[6] XUE J, HU C, REN H, et al. Memory errors prevention technology for C/C++ program based on probability[C]//International Conference on Natural Language Processing Andknowledge Engineering, 2012

[7] 易晓东, 杨学军. 一种 C 程序断言的全自动静态验证方法[J]. 计算机科学, 2006, 33(9):5
YI Xiaodong, YANG Xuejun. A passion verification method for C program assertions[J]. Computer Science, 2006, 33(9): 5

(in Chinese)

[8] YONG S H, HORWITZ S . Protecting C programs from attacks via invalid pointer dereferences[C]// Proceedings of the 11th ACM SIGSOFT Symposium on Foundations of Software Engineering 2003 held jointly with 9th European Software Engineering Conference, 2003

[9] HOLZMANN G J, SMITH M H. A practical method for verifying event-driven software[C]// Proceedings of the 1999 International Conference on Software Engineering, 1999: 597-607

[10] 胡军, 陈松, 王明明. AltaRica 3.0 模型到 PROMELA 模型转换与验证方法研究[J]. 计算机工程与科学, 2017, 39(4): 708-716

HU Jun, CHEN Song, WANG Mingming. Research on the conversion and verification method from AltaRica 3.0 model to PROMELA model[J]. Computer Engineering and Science, 2017, 39(4): 708-716 (in Chinese)

[11] DEL MAR GALLARDO MarÍa, MARTÍNEZ JesÙs, MERINO Pedro, et al. αSPIN: a tool for abstract model checking[J]. International Journal on Software Tools for Technology Transfer, 2004, 5(2/3):165-184

[12] YOUNES A B, HLAOUI Y B, AYED L J B. A meta-model transformation from UML activity diagrams to Event-B models[C]// 2014 IEEE 38th International Computer Software and Applications Conference Workshops, 2014

PROMELA based formal verification for safety-critical software

XING Liang[1], DING Chengjun[2], DU Hupeng[2], MA Chunyan[2]

(1.Aeronautic Computing Technology Research Institute, Xi′an 710076, China;
2.School of Software, Northwestern Polytechnical University, Xi′an 710072, China)

Abstract: Based on the PROMELA formal model, this paper studies the techniques for verifying the five types of fault in the C program: assertion violation, array out-of-bound, null pointer dereference, deadlock and starvation. Firstly, it establishes two types of mapping rules from the abstract syntax tree nodes of the C program to the PROMELA model and verifies the attribute-related functions of the PROMELA model. Secondly, according to the mapping rules, the algorithm for automatically generating the PROMELA formal model from the C program is proposed and theoretically analyzed. Then, based on the PROMELA model, the formal verification technique for the five types of fault in the C program respectively is given. Finally, the verification scope of the five types of fault is analyzed. For each type of fault, 12 cases of the C programs are studied. The experimental results demonstrate the effectiveness of the technique.

Keywords: C program; PROMELA model; software failure; formal verification

引用格式: *邢亮, 丁成钧, 杜虎鹏, 等. 基于 PROMELA 模型的安全关键软件形式化验证技术[J]. 西北工业大学学报,2022, 40(5): 1180-1187*

XING Liang, DING Chengjun, DU Hupeng, et al. PROMELA based formal verification for safety-critical software[J]. Journal of Northwestern Polytechnical University, 2022, 40(5): 1180-1187 (in Chinese)