利用变量状态转换模型进行部分软件错误的检测

张广梅[1] 李景霞[2]
(山东农业大学信息科学与工程学院 泰安 271000)[1] (安徽农业大学信息与计算机学院 合肥 230036)[2]

摘 要 应用程序中的功能通常是通过对变量的操作来实现。应用程序中变量的操作包括赋值、引用等不同的方式。针对普通变量和指针变量在程序中的使用方式,对变量的状态进行了分析,并根据变量使用的特点,定义了普通变量和指针变量的状态转换模型。在此基础上,给出了与变量有关的软件错误的定义,并讨论了基于变量切片的软件错误的检测方法。

关键词 变量状态转换模型,程序切片,软件错误
中图法分类号 TP311 **文献标识码** A

Detecting Software Error by Using State Transition Model of Variable

ZHANG Guang-mei[1] LI Jing-xia[2]
(College of Information Science and Engineering, Shandong Agricultural University, Taian 271000, China)[1]
(School of Information and Computer, Anhui Agricultural University, Hefei 230036, China)[2]

Abstract Variables are used in a program in order to implement the function of a program. There are different operations about a variable in a program and the operation on a variable can change the state of a variable. According to the different usage of a variable, different states of a variable were analyzed. First, the safe and unsafe states of a normal variable and a pointer variable were defined in this paper. Then the rules about the change between different states were also defined. After that, the state transition model of variable was provided. By using the state transition model of variable and the theory of program slice, a variable's unsafe state can be traced.

Keywords State transition model of variable, Program slice, Software error

1 引言

在应用程序编写的过程中,会大量使用各种类型的变量。应用程序处理的过程中,根据算法的要求,利用各种不同的公式进行变量值的计算从而一步步地实现应用程序的功能。应用程序中变量的异常使用会导致软件错误的发生。比如在C、C++等应用程序中,所有的变量必须遵循先声明,后引用的原则[1],对于某些局部变量而言,如果没有赋初值,变量处于未赋初值的状态,当变量处于此状态时,变量的值是一个不确定的随机值,这个不确定的随机值会导致程序运行结果的异常;除此之外,在C、C++应用程序的开发过程中,会大量使用动态内存,动态内存的使用遵循使用前申请,使用后回收的过程[1],如果在程序设计中不遵循这一原则,会导致包括内存泄露等各种动态内存错误的发生。

为有效地检测应用程序中潜在的各种由于变量的异常使用导致的软件错误,Sattar[2]等从路径测试的角度进行了软件错误检测方法的讨论;Zhenbo Xu[3,4]等从动态内存状态的使用角度出发对程序中动态内存泄漏的检测方法进行了讨论。本文从变量的状态分析出发,对应用程序中变量的状态进行跟踪,利用变量状态的变化,对程序中潜在的错误进行检测。

2 变量的状态分析

在应用程序中频繁地使用各种不同类型的变量解决问题。根据解决问题的需要,变量可以在赋值号的左边出现,也可以在赋值号的右边,或者在循环语句或条件语句中作为控制条件出现。C、C++等应用程序中的变量有普通变量和指针变量两类不同的变量,由于指针变量的特殊性,下面分别对这两种类型的变量的状态进行讨论。

2.1 普通变量的几种状态

2.1.1 变量被声明状态(用 V_DC 表示)

在C、C++等强类型的程序设计语言中,变量的使用必须遵循先声明再引用的原则。当在声明语句中对变量进行声明之后,变量处于被声明状态。

2.1.2 变量被定值状态(用 V_DF 表示)

在应用程序中,大量使用赋值语句完成变量的赋值操作。当变量在赋值号的左边出现时,变量处于被定值状态。

2.1.3 变量被引用状态(用 V_U 表示)

在应用程序中,变量可以在赋值运算符的右侧出现,或在循环语句、条件语句中作为控制条件出现,此时变量所处的状态为被引用的状态。应用程序中变量一旦被定值之后,可以根据算法的需要被任意多次的引用从而实现程序的功能。

张广梅(1972—),女,博士,副教授,主要研究方向为软件测试,E-mail:lotus@sdau.edu.cn;李景霞(1976—),女,博士,副教授,主要研究方向为模型分析与验证。

2.2 指针变量的几种状态

上面关于变量状态的定义适用于非指针变量。对于指针变量而言，由于指针变量的特殊性，在使用之前必须先进行存储空间的分配；如果是在堆上进行的动态内存分配，则需要在适当的时候对指针变量所指向的存储空间进行回收。下面针对指针变量的特点，对指针变量的状态进行讨论。

2.2.1 指针变量被声明状态(用 PV_DC 表示)

当用声明语句对变量进行声明之后，指针变量处于被声明状态。

2.2.2 指针变量被分配动态内存状态(用 PV_H 表示)

根据指针变量的特点，指针变量是一个用来记录内存地址的变量。指针变量的值是一个内存地址。在完成指针变量的声明之后，指针变量的值是不确定的，因此，必须通过赋值语句完成指针变量的赋值运算。应用 C、C++程序中，可以通过 malloc 函数或 new 运算符进行指针变量的赋值。在 malloc 或 new 运算过程中，会在堆上进行动态存储空间的申请。此时指针变量指向堆空间，指针变量处于 PV_H 状态。

2.2.3 指针变量被分配栈内存状态(用 PV_S 表示)

在 C、C++应用程序中，除了可以使用 new 和 malloc 完成指针变量的赋值，还可以使用“&”运算符取出变量的地址，并将这一结果赋值给指针变量。通常情况下，此时获得的内存地址通常是栈上内存的地址，因此用 PV_S 表示这种状态。

2.2.4 指针变量所指向的存储空间被使用状态(用 PV_U 表示)

在 C、C++程序中，可以使用“*”运算符对指针变量所指向的内存进行读、写操作。因此，在应用程序中使用“*”运算符对指针变量所指向的存储空间进行操作时，指针变量处于该状态。

2.2.5 指针变量已被检查状态(用 PV_C 表示指针变量处于已经被检查的状态)

对于应用程序中的指针变量，通过 new 或 malloc 进行进行动态内存申请之后，必须对申请的结果进行检查，否则，由于堆空间的限制可能造成动态内存的申请失败，使得接下的动态内存访问出错。因此，在使用动态内存之前必须对动态内存申请的结果进行检查。用 PV_C 表示已经进行了申请结果的检查。

2.2.6 指针变量所指向的存储空间被回收状态(用 PV_F 表示)

在应用程序中，当使用 free 或 delete 操作后，指针变量所指向的动态内存被回收，此时，指针变量处于该状态。

2.3 变量异常使用的状态

以上对普通变量和指针变量的几种状态进行了讨论。这几种状态都是变量使用过程中的正常状态。在应用程序中由于变量的异常使用会造成变量处于异常的状态。下面对程序中变量的几种异常状态进行讨论。

2.3.1 变量的未定值引用状态(用 UD_U 表示)

在应用程序的编写过程中，由于各种原因，可能存在着在未对变量进行赋值之前引用变量的值的情况。此时，由于变量未被赋初值，使得变量的值为一个不确定的值，从而造成潜在的运行结果的异常。因此，把这种状态定义为变量的未定值引用状态。当变量处于该状态时，意味着程序的运行结果存在着不确定性，程序中存在着一个潜在的错误。

2.3.2 指针变量的未分配使用状态(用 PVUA_U 表示)

对于应用程序中的指针变量，必须先进行赋值确定指针变量的值，之后才能对指针变量所指向的内存(堆或栈)进行读写操作。否则，会造成指针变量处于 PVUA_U 状态。

2.3.3 对指针变量重复进行内存分配操作(用 PV_RA 表示)

PV_RA 状态是指在应用程序中已经对指针变量进行了动态内存分配操作之后，在未对指针变量所指向的内存进行回收之前，重新对该指针变量进行内存分配操作(指针变量的赋值操作)，新的内存分配操作会造成之前的动态内存处于悬挂状态，造成内存泄露。

2.3.4 指针变量所指向的存储空间未被回收状态(用 PV_UF 表示)

PV_UF 状态是指在应用程序中已经对指针变量进行了动态内存分配操作之后，在应用程序结束之前未进行动态内存的回收操作，此时同样会造成内存泄露。

3 变量状态转换模型

对应用程序中的变量的操作主要有读、写两种操作。变量读、写操作会引起变量状态的改变。下面针对普通变量和指针变量两种不同类型的变量，对变量的状态变化进行讨论。

3.1 普通变量的状态转换模型

普通变量的状态转换模型用五元组 $M1=(S,\Sigma,\delta,s_0,F)$ 进行描述。

S 代表状态的集合，根据第 2 节中关于变量状态的讨论，集合 S 表示为：$S=\{V\_DC,V\_DF,V\_U,VUD\_U\}$

Σ 是对变量所进行的操作的集合。根据程序中变量的使用方式，对于普通变量而言，变量的操作分为读写操作：当变量出现在赋值运算符的左侧，或变量出现在输入语句中时，进行变量的写操作；变量的其他引用形式成为变量的读操作。Σ 集合定义为 $\Sigma=\{R,W\}$，R 表示对变量进行读操作，W 表示对变量进行写操作。

s_0 表示初态。在变量状态转换过程中，变量的 V_DC 被定义为初态。变量的状态转换从该状态开始。

F 表示状态转换过程中的状态转换中的异常状态的集合。根据第 2 节中关于变量状态的讨论，对于普通变量而言，该集合被定义为：$F=\{VUD\_U\}$。

δ 代表状态转换函数，可以利用下面的状态转化矩阵对该函数进行描述。

表 1 普通变量的状态转换矩阵

变量状态 \ 训变量操作	R	W
V_DC	VUD_U	V_DF
V_DF	V_U	V_DF
V_U	V_U	V_DF
VUD_U	VUD_U	V_DF

根据状态转换模型的定义，普通变量的状态转换如图 1 所示。

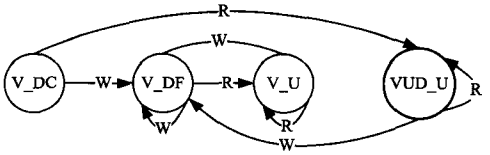

图 1 普通变量的状态转换图

3.2 指针变量的状态装换模型

指针变量的状态转换模型用五元组 $M2=(S,\Sigma,\delta,s_0,F)$ 进行描述。

S 代表状态的集合，根据第 2 节中关于变量状态的讨论，集合 S 中包含 9 个状态，可表示为：$S=\{PV\_DC,PV\_H,PV\_S,PV\_U,PV\_C,PV\_F,PVUA\_U,PV\_RA,PV\_UF\}$。

Σ 是对变量所进行的操作的集合。程序中指针变量相关的操作主要包括：通过 malloc 函数和 new 运算符完成动态内存分配操作、通过取址运算符将栈地址赋值给指针变量的操作、通过比较运算对动态分配结果进行检查的操作、通过 free 或 delete 进行动态内存的回收操作、通过“*”运算符进行动态内存的读操作。除了上述 5 种操作会对变量的状态产生影响之外，程序的结束语句也会造成指针变量状态的变化，为描述方便，分别用符号 $HA,SA,HC,HF,RM,RETURN$ 表示这 6 种操作。由此，Σ 集合定义为 $\Sigma=\{HA,SA,HC,HF,RM,RETURN\}$。

s_0 表示初态。在指针变量状态转换过程中，变量的 PV_DC 被定义为初态。变量的状态转换从该状态开始。

F 表示状态转换过程中的状态转换中的异常状态的集合。根据第 2 节中关于变量状态的讨论，对于普通变量而言，该集合被定义为：$F=\{PVUD\_U,PV\_RA,PV\_UF\}$。

δ 代表状态转换函数，可以利用下面的状态转化矩阵对该函数进行描述。

表 2 指针变量状态转换矩阵

状态＼操作	RM	HA	SA	HF	HC	RETURN
PV_DC	PVUA_U	PV_H	PV_S	PVUA_U	PVAU_U	PV_DC
PV_H	PVUA_U	PV_RA	PV_RA	PVUA_U	PVUA_U	PV_UF
PV_S	PV_S	PV_H	PV_S	PVUA_U	PVUA_U	PV_S
PV_U	PV_U	PV_RA	PV_RA	PV_F	PV_C	PV_UF
PV_C	PV_U	PV_RA	PV_RA	PV_F	PV_C	PV_UF
PV_F	PVUA_U	PV_H	PV_S	PVUA_U	PVUA_U	PVUA_U
PVUA_U	PVUA_U	PVUA_U	PVUA_U	PVUA_U	PVUA_U	PVUA_U
PV_RA	PVUA_U	PV_RA	PV_RA	PVUA_U	PVUA_U	PV_UF

根据状态转换模型的定义，指针变量的状态转换图如图 2 所示。

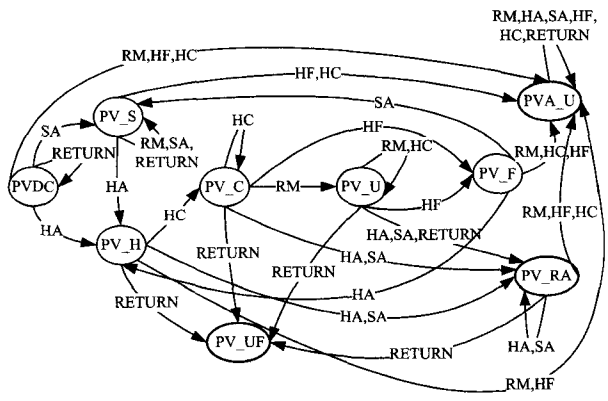

图 2 指针变量的状态转换图

4 与变量状态有关的软件错误分析

根据第 3 节中关于变量状态模型的分析，在应用程序中，一旦变量状态的变化到达状态模型的终态时，意味着程序中对变量的引用出现错误，即程序中存在错误。根据导致变量状态异常的原因进行分析，程序中存在的与变量状态有关的错误包括以下几种。

4.1 变量的未定值引用错误

应用程序中，一个未被显示定值的局部变量，其值是一个随机数，因此，在未对局部变量进行赋值的情况下引用该变量可能会造成程序运行结果的异常。编译器会对变量的未赋值引用做一些检测，但编译器无法检测应用程序中对变量的条件定值，即在只在某种条件成立的情况下才对变量进行赋值的情况，这同样会造成变量未定值引用错误的出现。

4.2 无用变量错误

在应用程序结束前，变量没有达到的 V_U 状态，意味着在程序中没有引用该变量的值进行操作，因此这个变量是应用程序中的无用变量。

4.3 动态内存未分配使用错误

通过图 2 对指针变量状态转化的描述，当在程序中声明指针变量之后，在未进行动态内存分配之前进行的没存的读操作、动态内存分配结果的检查操作、动态内存的回收操作均可以造成动态内存未分配使用错误。除此之外，在对指向栈内存的指针变量进行内存回收操作同样也会造成该错误的出现。

4.4 内存泄露错误

根据指针变量的状态转换关系，在应用程序结束之前，如果指针变量的状态不是 PV_F 状态，意味着在应用程序结束之前该指针变量所指向的动态内存没有被回收，存在着内存泄露的错误。

4.5 指针变量被重复赋值错误

根据图 2 的介绍，在对指针变量进行动态内存分配操作之后，在未释放动态内存之前，重新对指针变量进行赋值操作会造成变量的重复赋值错误发生。一旦对某个指针变量进行了动态存储空间分配操作之后，重新对该指针变量进行赋值操作可能会导致内存泄露的发生，这是造成系统老化的原因之一。

5 基于变量状态分析的软件错误检测

应用程序中变量使用正确与否决定了程序的运行状态。因此，可以借助变量状态分析对代码中的潜在的错误进行检测。

5.1 基于程序控制结构获取变量切片

程序切片的基本思想由 M. Weiser 于 1979 年在他的博士论文中提出[5]。所谓程序切片，是指与某个输出有关的语句和谓词构成的程序。程序切片是一种分析和理解程序的技术，是通过对源程序中每个兴趣点分别计算切片来达到对程序的分析和理解。程序中某个兴趣点的程序切片不仅与在该点定义和使用的变量有关，而且与影响该变量的值的语句和谓词以及受该变量的值影响的语句和谓词有关。为对程序中变量状态的变化进行跟踪，借助于程序切片的思想，从程序中抽取出于某个变量相关的语句和谓词构造变量切片。下面给出一个变量切片的示例。

```
int* x,y;
x=new int;
*x=10;
```

```
if(y>0)
cout<<"****";
cout<<*x;
```

根据 weiser[5] 提出的程序静态切片的基本思想，得到程序中变量 x 的切片代码如下：

```
int* x,y;
x=new int;
*x=10;
cout<<*x;
```

5.2 对变量切片中变量所进行的操作进行标记

在获取变量切片的基础上，根据第 3 节中关于普通变量和指针变量的操作的分析，对切片中所涉及到的变量的操作进行标记，通过对变量的操作标记确定变量的操作。对 5.1 节中关于变量 x 的切片，针对涉及到变量 x 的操作进行如下标记：

```
1. int* x,y;
2. x=(int*)malloc(sizeof(int));[HA]
3. *x=10;[RM]
4. cout<<*x;[RM]
```

5.3 利用变量状态转换模型对变量切片中所涉及的变量的状态转换过程进行分析

下面以 5.2 节中的变量切片中的代码为例，利用第 3 节中的指针变量状态转换模型进行分析。

从初态出发，进行 HA 操作，根据变量的状态转换模型，确定后继状态为 PV_H；在 PV_H 状态下，进行 RM 操作，进入 PVUA_U 状态，该状态为状态转换模型中的终态，此时意味着程序中存在着动态内存未分配引用的错误，造成这种错误的原因是在动态内存分配之后没有进行分配结果的检查；在 PVUA_U 状态下，继续进行 RM 操作进入 PVUA_U 状态。在变量切片中的关于指针的所有操作都执行结束之后，指针变量的状态没有进入 PV_F 状态，意味着程序中存在着内存泄露的错误。

结束语 本文首先对应用程序中的变量的状态进行了分析，在此基础上，根据程序中变量引用的方式，建立了普通变量和指针变量的状态转换关系，并讨论了利用变量切片进行软件错误检测的方法。该方法在对程序进行静态分析的基础上进行，克服了动态检测方法中由于执行分支的限制而造成的路径覆盖不完整而引起的不能暴露所有错误的问题，为方便地检测程序中潜在的由于变量的异常使用导致的软件错误提供了帮助。

参 考 文 献

[1] 李普曼，拉乔伊，等. C++ Primer 中文版(第 5 版)[M]. 王刚，杨巨峰，译. 北京：电子工业出版社，2013：23-50

[2] Sattar H, Bajwa I S, et al. Automated DD-path testing: A challenging task in software testing[C]// Ninth International Conference on Digital Information Management. Phitsanulok, IEEE, 2010: 230-236

[3] XuZhen-bo, Zhang Jian, Xu Zhong-xing. Memory Leak Detection Based on Memory State Transition Graph[C]// 18th Asia Pacific Software Engineering Conference, 2011. Ho Chi Minh, IEEE, 2011: 33-40

[4] Sor V, Ou P, et al. Improving Statistical Approach for Memory Leak Detection Using Machine Learning[C]// 29th IEEE International Conference on Software Maintenance, 2013. Eindhoven, IEEE, 2013: 544-547

[5] 李必信. 程序切片技术及其应用[M]. 北京：科学出版社，2006：3-4

(上接第 478 页)

[4] 王智学，董庆超，陈剑. 基于能力的复杂系统需求分析[C]// 江苏省系统工程学会军事系统工程委员会第十届学术年会. 2008：115-121

[5] 王智学，董庆超，陈彬，等. 基于 UML 模型的 C4ISR 系统能力需求分析与验证[J]. 系统工程与电子技术，2009，31(9)：2167-2171

[6] 董庆超，王智学，陈剑，等. 基于描述逻辑的能力需求模型验证方法[J]. 系统工程与电子技术，2010，32(3)：533-539

[7] Vincent C, Matthieu R. Interoperability constraints and requirements formal modelling and checking framework [C]// Ternational Federation for Information Processing. 2010: 219-226

[8] Wagenhals L W, Haider S, Levis A H. Synthesizing executable models of object oriented architectures[C]// Workshop on Formal Methods Applied to Defence Systems. Adelaide, Australia, 2002

[9] 张炜钟. 指控系统能力需求的可执行建模及仿真评估技术研究[D]. 南京：解放军理工大学，2012

[10] 陈章耀. 模型驱动的业务生成技术中业务过程模型的研究与应用[D]. 北京：北京邮电大学，2008

[11] Group C4ISR Architecture Working. C4ISR architecture framework version2. 0 [R]. The United States: Department of Defense, 1997

[12] Brockmans S, Volz R, Eberhart A. Visual modeling of OWL DL ontologies using UML[M]// Lecture Notes on Computer Science 3298. 2004: 198-213

[13] Van Der Straeten R. Inconsistency management in model-driven engineering: An approach using description logics [D]. Vrije Universiteit Brussel, 2005

[14] W3C Working Draft. SWRL: a semantic web rule language combining OWL and RuleML[EB/OL]. 2009-07-02. http://www.w3.org/Submission/SWRL/

[15] Motik B, Sattler U, Studer R. Query answering for OWL-DL with rules[C]// Proc. of the 3rd International Semantic Web Conference. 2004: 549-563

[16] Glimm B, Horridge M, Parsia B, et al. Asyntax for rules in OWL 2[R]. Oxford: University of Oxford, 2008

[17] 朱雪峰，金芝. 关于软件需求中的不一致性管理[J]. 软件学报，2005，16(7)：1221-1231