ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.3, March 2010, pp.539−547
doi: 10.3724/SP.J.1001.2010.03429

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种基于安全状态转移的简并测试集生成方法*

程 亮[1,2+], 张 阳[2], 冯登国[2]

[1](中国科学技术大学 电子工程与信息科学系,安徽 合肥 230027)

[2](信息安全国家重点实验室(中国科学院 软件研究所),北京 100190)

Approach of Degenerate Test Set Generation Based on Secure State Transition

CHENG Liang[1,2+], ZHANG Yang[2], FENG Deng-Guo[2]

[1](Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei 230027, China)

[2](State Key Laboratory of Information Security (Institute of Software, The Chinese Academy of Sciences), Beijing 100190, China)

+ Corresponding author: E-mail: chengliang@is.iscas.ac.cn

Cheng L, Zhang Y, Feng DG. Approach of degenerate test set generation based on secure state transition. *Journal of Software*, 2010,20(3):539−547. http://www.jos.org.cn/1000-9825/3429.htm

Abstract: Based on predecessors' work, this propose the concept of degenerate test set (DTS) and an approach that performs test generation and redundancy elimination in the light of the special requirement of verification of the secure operating system. This approach is secure state transition-based for the first time and can generate an efficient test set by reducing the redundant system state transitions and similar properties with model checkers in the test case generation. Furthermore, it discusses the validity of the DTS when only some cases of the set fail and improve the DTS generation algorithm. The experiments prove that this approach can reduce the size of test set efficiently.

Key words: security operating system verification; formal method; model checking; test case optimization; degenerate test set

摘 要: 在总结前人工作的基础上,结合安全操作系统对测试的特殊需求,提出了简并测试集(degenerate test set,简称 DTS)的概念,设计了一种使用模型检测的基于安全状态转移的高效测试集生成方法.该方法以状态转移为化简对象,在利用模型检测技术生成测试用例的同时,归并相同的状态转移并化简需求集中的冗余属性,从而最终达到化简测试集的目的.在此基础上,探讨了单个用例失败时用例集的有效性问题,并对 DTS 生成算法进行了改进.实验结果表明,该方法可以有效地对测试集中的冗余进行化简.

关键词: 安全操作系统测评;形式化方法;模型检测;测试用例化简;简并测试集

中图法分类号: TP311 **文献标识码**: A

随着 Internet 应用的广泛深入,计算机信息系统安全问题的日益凸显,安全操作系统的研究已吸引了人们越

* Supported by the National Natural Science Foundation of China under Grant No.60970028 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant Nos.2007AA01Z465, 2007AA01Z414 (国家高技术研究发展计划(863))

Received 2007-12-06; Revised 2008-05-06; Accepted 2008-08-07

来越多的关注.然而,我们在研究工作中发现,相比于安全操作系统的研发,安全操作系统的测试验证更是一项紧迫的工作.由于安全模型通常是独立于实现开发的,我们并不能从安全模型的正确性推出其实现是安全的[1].因此,比较理想的测试方法是采用自动化的方法生成测试用例,将用例细化成具体的测试数据后,再利用合适的测试驱动在具体实现上自动运行测试用例,以验证实现与规范之间的一致性.

安全操作系统测试的自动化基础是形式化方法.目前已经有很多形式化方法应用于自动用例生成[2−4].形式化方法中的模型检测技术是一种基于模型的属性验证方法,它以系统模型和待验证属性作为输入,其优点在于验证过程的完全自动化,验证速度快、效率高.如果一个属性不满足,它可以立刻反馈出违反该属性的执行路径.对于安全操作系统测试来说,最后一点尤为重要,因为一个违反安全需求的执行路径直接对应安全模型或安全策略配置上的漏洞,对该路径细化之后就可以直接生成测试用例.目前,已经有很多种模型检测器用于工业级的大规模应用,比如 SMV[5,6]和 SPIN[7]等,很多基于模型检测的测试方法[8~10].这些方法大多着眼于满足各式的测试覆盖标准,如状态覆盖(state coverage)、转换覆盖[11](transition coverage)和修正条件/判定覆盖(modified condition/decision coverage)等等,对于生成的用例集的大小并不关注.然而,很多基于模型检测生成的测试用例集都会存在相当的冗余[10,12,13].对于验证安全操作系统来说,重点在于安全属性的验证,如“setuid 程序在执行不可信程序前必须放弃 root 权限”或者“禁止以写模式打开文件到 stderr 或 stdout”等,文献[14]以时态逻辑公式的形式总结了 5 条常见的安全属性,并对如何利用模型检测验证安全属性进行了初步的探讨.相对而言,覆盖率并不是首要关注的指标,而且操作系统状态多,安全需求多,如果不对测试用例进行化简,测试的工作量会非常庞大.

目前针对测试用例化简的研究主要分为对测试需求集进行化简、对模型检测器的算法进行化简以及对已生成的测试集进行化简[15],主要研究方法如下.

Hong[16]等人将测试用例生成转化成使用模型检测器来证明时序逻辑公式的正确性,从而使得测试用例的生成完全自动化,并提出了测试用例生成的启发式算法.然而,他们的方法只能化简测试需求集,对于测试用例内部冗余并没有给予考虑.

Zeng 等人[13]在 Hong 的基础上提出了一种基于树形结构的测试集优化方法,以系统状态作为考察对象,利用测试树来化简测试用例内部重复的系统状态,并在模型检测生成测试用例的同时对测试集进行化简.该方法得到的测试树既保证了不影响原始测试集的覆盖率也可以有效地减少测试集的冗余度.

Hamon[10]等人提出了一种修改模型检测算法生成有效测试集的方法.其主要思想是:当模型检测器针对某属性产生了测试用例后,不是立即返回初始状态开始对新属性 p 的探索,而是在该用例的基础上继续探索状态空间,如果在探索过程中发现路径满足了 p,则将 p 从目标集中删除,从而达到化简用例集的目的.该方法的优点在于完全依赖模型检测器,充分利用了模型检测器的性能,人工干预极少,由此导致的问题是,虽然研究人员对探索算法做了很多改进,然而从现有用例探索对新属性的满足完全是盲目的,而且探索的深度也难以控制,太浅效率难以保证,太深则资源消耗严重.

聂长海等人[15]提出可以根据测试需求之间的相互关系对测试需求集进行划分来实现测试集的化简.章晓芳等人[17]则在此基础上提出了一种基于测试需求约简的测试集优化方法,该方法建立需求集和测试集之间的映射,然后寻找是否存在一个测试用例覆盖多个测试需求的情况,如果存在,则对需求集进行约简.

就目前而言,现有的方法对于测试集的冗余化简效果都不是很理想,化简后的测试集仍存在着继续化简的空间.由上面的讨论可以看出,在模型检测器生成测试用例的同时对测试集进行化简是一种比较主流的途径.基于此,本文在前人的工作基础上针对安全操作系统对测试的特别要求,以系统的状态转移作为测试用例的化简对象,对模型检测器生成的测试用例集进行化简,提出简并测试集(degenerate test set,简称 DTS)的概念,并给出简并测试集的生成算法,从而有效地降低测试用例内部的冗余并同时化简了测试需求.

本文第 1 节阐述简并测试集的形式化定义.第 2 节详细介绍以状态转移为对象的测试集化简思想,给出简并测试集(DTS)的生成算法.第 3 节描述算法的测试结果并进行分析.最后是小结和下一步工作计划.

1 简并测试集的形式化定义

理想的测试集不仅要求用例的数量最少,而且要求用例的总执行长度最短[10].然而,要求同时满足上述两个条件实际上是个 NP-complete 问题[16],因此安全操作系统测试中,我们只能退而求其次:在不损害安全需求的基础上,尽量减少测试集中的冗余.要做到这一点,我们需要在最少用例数量和最短路径长度之间加以取舍.对于操作系统级别的测试来说,验证某个属性的测试步骤经常会达到几千甚至几万的量级,如果步骤中涉及到加密算法或计算大文件的哈希值,尤其是和 TPM 相关的运算,路径长度上的微小增加都会给验证带来很大的开销;另一方面,重新开始新的测试需要做很多的初始化工作,典型的就是重启机器或者重新登陆.因此,我们应选择尽量在某个路径的中间状态开始新的测试,而不是对于任何新的安全需求都重新从初始状态开始测试.

现有的方法,无论是采用二叉判定图来化简测试集冗余,或是利用可编程接口扩展模型检测器所能到达的状态空间,还是采用树型结构化简测试用例的步骤,它们都是以测试路径中的系统状态为操作对象的.实际上,决定测试集大小的直接因素是状态转移的数量,而不是状态的数量,因此我们以状态转移作为考察对象,尽可能地化简测试集中重复的状态转移,从而达到优化测试集的目的.针对状态转移化简后的测试集,我们提出了简并测试集(DTS)的概念.

通常,我们将模型检测器要求的系统模型定义成一个有限状态自动机(finite state automata,简称 FSA).

定义 1. 系统模型 M 是个有限状态自动机,表示为 $M=(S,s_0,L,T,F)$,其中 S 是有限状态集合;S_0 是初始状态集合,$S_0\subseteq S$;L 是一个有限标签集合;T 是转换集合,$T\subseteq(S\times L\times S)$;$F$ 是最终状态集合,$F\subseteq S$.

定义 2. 系统的运行,是一个有序的、可能有限也可能无限的转换集合(序列),表示为

$$\langle(s_0,l_0,s_1),(s_1,l_1,s_2),(s_2,l_2,s_3),\ldots\rangle,$$

即

$$\forall i,(i\geq 0)\rightarrow(s_i,l_i,s_{i+1})\in T\ .$$

系统 M 所有的运行集合记为 R.一个测试用例是系统的一个有限运行.虽然定义 2 中运行的定义是可以无限长的,然而对于测试用例来说,无限长的运行意味着无限的资源消耗,是无法忍受的,因此实际使用中我们把测试用例限制在有限运行这个前提下.

定义 3. 对于一个安全操作系统模型 M,定义:

测试需求集为 SP={所有待验证的安全需求},

测试用例集合为 TS={所有待验证的安全需求的测试用例},

则有,测试需求 $p\in SP$ 到测试用例集 TS 的映射为 $Test(p)$={ $t:TS\cdot$ 所有可以满足 p 的测试用例 t},显然,$\forall p\in SP\rightarrow Test(p)\subseteq TS$.

通常,利用模型检测技术生成测试用例是基于模型检测器的反例生成能力的[10,13]:为了生成满足属性 p 的测试用例,对程序进行模型检测,检验在程序各处属性“$\Box\neg p$”的满足情况,如果出现反例,则表明存在满足 p 的测试用例,如果模型检测器找不到“$\Box\neg p$”的反例,则表明属性 p 是不存在对应测试用例的.一个反例对应一个测试用例[10].由于模型检测器对于一个待验证属性只生成一个反例,因此,一般情况下,利用模型检测器生成测试用例时,一个待验证属性也只对应一个测试用例.即定义 3 中的映射 $Test(p)$得到的集合中或者只有一个测试用例 $t\in TS$,或者为空集(对应模型检测器找不到反例的情况).

由于我们的化简对象是单纯的状态转换(s_{i-1},s_i),也就是说,不同序列中的元素,只要二元组(s_{i-1},s_i)相同,无论其对应的标签是否相同,我们都认为二者是等价的.为此,有必要区别测试序列中每项中的状态转换二元组和标签.可以将测试序列记成如下形式:

定义 4. 对于测试序列 $t=\langle(s_0,l'_0,s_1),(s_1,l'_1,s_2),\ldots,(s_n,l'_n,s_{n+1})\rangle$,$n=\#t-1$,可转记为下面的形式:

$$t=\langle\sigma_1,\sigma_2,\ldots,\sigma_{n+1}\rangle\ ,$$

其中 $\sigma_i=(l_i,e_i)$ 为二元组,$l_i=l'_{i-1}$,$e_i=(s_{i-1},s_i)$, $i\in[1..n+1]$.

这样我们就可以很方便地构造简并测试集如下:

定义 5. DTS 是这样的测试集:首先,它是 TS 的一个子集;其次,简并测试集中任意两个测试序列都不包含相同的状态转移.即

$$\forall t,s \in DTS, s \neq t \rightarrow \neg\exists i:1..\#t, j:1..\#s \mid t\,i.2 = s\,j.2 .$$

2 简并测试集生成算法

首先给出简并测试集的生成算法,然后结合实际测试中可能遇到的问题对 DTS 的形式化描述以及 DTS 生成算法加以改进.

2.1 DTS生成算法

假设 p,q 是 SP 中任意两个属性, T_p 和 T_q 分别是它们对应的测试用例,那么,T_p 和 T_q 之间的关系可能有 3 种情况:

- $T_p \subseteq T_q \vee T_q \subseteq T_p$,即一个测试用例被另一个测试用例完全包含;
- $T_p \cap T_q \neq \varnothing \wedge T_p - T_p \cap T_q \neq \varnothing \wedge T_q - T_p \cap T_q \neq \varnothing$,即两个测试用例有公共的状态转换,但各自都有自己唯一的状态转换;
- $T_p \cap T_q = \varnothing$,即两个测试用例完全没有公共部分.

显然,只有前两种情况可以进行化简,而第 1 种情况亦可视为第 2 种情况 $T_p(\text{or } T_q) - T_p \cap T_q \neq \varnothing$ 时的特例.我们先给出如下定义:

定义 6. *Goals*={所有待验证安全需求的形式化描述},
Knownedges={所有已生成测试用例的状态转移},
Unavailable={模型检测无法生成用例的属性}.

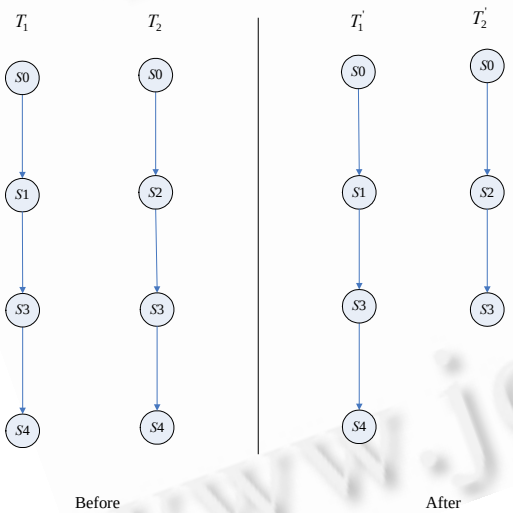

Fig.1 Reducing of state transition
图 1 状态转换的化简

假设有两个测试序列,T_1=⟨$S0$,$S1$,$S3$,$S4$⟩ 和 T_2=⟨$S0$,$S2$,$S3$,$S4$⟩.假设 T_1 中的所有状态转换已经包含在 *Knownedges* 中,T_2 为新生成的测试序列.显然,两个测试序列的最后一步状态转移 $S3 \rightarrow S4$ 从测试角度来说是完全一致的,二者标签上的差别对测试没有任何影响,因此,我们认为二者是完全等价的.即对于 *Knownedges* 来说,T_2 中的 $S3 \rightarrow S4$ 是可以被化简掉的冗余.而 $S0$ 虽然同是出现在两个测试序列中,但分属不同的状态转换 $S0 \rightarrow S1$ 和 $S0 \rightarrow S2$,即使化简,在实际测试时仍然对应两个对状态 $S0$ 的完全不同的输入或操作,因此我们认为化简意义不大.化简后的测试序列如图 1 右侧所示.即

T'_1=⟨$S0$,$S1$,$S3$,$S4$⟩,T'_2=⟨$S0$,$S2$,$S3$⟩.

DTS 生成算法以 *Goals* 和系统模型为输入(算法中未标明),输出为简并测试集 *DTS* 和 *Unavailable*.一般说来,集合 *Unavailable* 中的元素可能有两种来源:或者模型确实无法满足该属性,或者满足该属性所对应的测试路径长度超出了模型检测器所能处理的范围.算法的伪代码见算法 1.

算法 1. *DTS* 测试集生成算法.
Input: *Goals*;
Output: *DTS* and *Unavailable*.
1 *Goals* := the set of requirements; *Knownedges* := empty set;
2 *Unavailable* := empty set; *DTS* :=empty set;
3 **while** *Goals* is not empty **do**
4 SELECT and REMOVE p from *Goals*;
5 CALL model checker to GENERATE a test case T_i for p;

```
6    if successful then
7        foreach q in Goals do
8            if T_i SATISFY q then
9                REMOVE q from Goals;
10           end
11       end
12       E_I = EDGE(T_i) ∩ Knownedges;
13       T_i ← T_i − SEQ(T_i, E_I);
14       Knownedges ← Knownedges + EDGE(T_i);
15       DTS ← DTS + T_i;
16   else
17       ADD p to Unavailable;
18   end
19 end
```

其中,函数 $EDGE(T)$的结果为集合 $\{\forall\sigma:T\cdot\sigma.2\}$,即测试序列 T 中包含的所有不带标签的状态转换,函数 $SEQ(E,T)$的结果为集合 $\{e':E\mid(\exists\sigma:T\mid\sigma.e=e')\cdot\sigma\}$,即测试序列 T 中所有包含了集合 E 中的状态转换的项.算法首先从需求集 *Goals* 中取出一个安全需求 p,调用模型检测器生成关于 p 的测试用例 T_i,若生成不了,将 p 放入集合 *Unavailable* 并重新从 *Goals* 中选出新的 p.如果可以生成测试用例 T_i,且 T_i 在满足 p 的同时满足了 *Goals* 中的其他属性 q,则将 p 和 q 同时在 Goals 中删除,否则仅删除 p.第 2 步,找出 T_i 中所有状态转移集 E_T 和已知状态转移集合 *Knownedges* 的交集 E_I,E_I 即为 T_i 中已经被之前的测试用例覆盖的部分.最后将 T_i–$SEQ(E_I, T_i)$作为 T_i 的最终形式放入测试集 *DTS*,将 $EDGE(T_i)$加入 *Knownedges*,如此反复,直至 *Goals* 为空.

需要注意的是,*SP* 中两个不同的 p 可能会映射到相同 $t\in TS$.由于没有事先对需求集进行化简,所以不可避免的 *Goals* 中存在冗余,因此算法在生成测试用例成功之后检查当前用例是否能够满足 *Goals* 中的其他属性,如果满足,则将这些冗余的属性删除,从而减少算法最外层 while 循环的次数,提高算法的效率.

最后的输出结果 *DTS* 为简并测试集,而 *Unavailable* 为系统模型不可满足的属性集合.

2.2 DTS生成算法的改进

虽然算法 1 生成的简并测试集可以达到测试集的状态转换数最少的目标,但却存在一个问题:如果有相同路径的两个测试用例中有一个在具体测试中被证明不成立,那么另一个测试用例即使通过也不能保证它对应的属性成立.也就是说,当模型和实现不一致时,*DTS* 不能保证其有效性.仍以图 1 为例,*DTS* 的化简结果为 T'_1=⟨*S*0,*S*1,*S*3,*S*4⟩,T'_2=⟨*S*0,*S*2,*S*3⟩.如果 T'_1 不成功,那么仅仅 T'_2 成立并不能保证 T_2 是否满足,因为 $S3\rightarrow S4$ 的状态无法从失败的 T'_1 中推出.同样的,文献[10,13]得到的测试集也存在这样的问题.

一个很自然的改进方法是:修改测试序列中项的结构,加上该项是否成功的标签,同时第 1 项改成编号集合.对测试集中的测试用例进行编号,每个测试用例拥有唯一的编号.如果一个测试用例中的某个项出现在多个测试序列中,则将这些序列的编号全部都放到这个编号集合中.修改后项的定义如下:

定义 7. 对于序列 $T_i=\langle\sigma_1,\sigma_2,...,\sigma_n\rangle$,$T_i\in TS$,它的项是一个三元组:

$$\sigma_k=(I,e,hold)\in P(N)\times E\times\{0,1,NE\},\ k\in[1..\#T_i],$$

其中,集合 $I=\{T_i:TS\mid(\exists m:[1..\#T_i]\mid\sigma_m\in T_i\wedge\sigma_m.2=e)\cdot i\}$,用于记录所有包括了状态转移 e 的测试用例的编号,初始状态时 $I=\{i\}$.*hold* 是一个标志位.

$$hold=\begin{cases}0, & \text{if } \sigma_i \text{ not hold}\\ 1, & \text{if } \sigma_i \text{ hold}\\ \text{NE}, & \text{if } \sigma_i \text{ have not execute yet}\end{cases}.$$

这样,在修改测试序列的结构之后,当上例中 T'_1 不满足时,我们可以检查究竟是哪一项不满足,假设是 σ_k 不满足,则对于 $\forall i\in\sigma_k.I,T_i\in TS$ 均不满足.仍以图 1 为例,如果在执行 T'_1 时 $S3\rightarrow S4$ 失败,那么只需检查它对应的 $\sigma.1$ 的元素就可以推断出 T_2 也是无法执行的;如果在执行 T'_1 时失败的是 $S0\rightarrow S1$,虽然我们无法从它对应的 $\sigma.1$

中得出 T_2 成立与否(因为 T_2 中并不包括 $S0 \rightarrow S1$),但当 DTS 中所有的状态转移都被遍历一遍之后,我们可以得到所有失败的测试集合: $failure = \{T_i : DTS \mid \exists k \cdot \sigma_k \in T_i \wedge \sigma_k.3 = 0\}$.

修改后的算法 1 对上述问题可以做很好的处理.然而在实验中我们发现,它对于图 2 类型的测试用例生成方式却不能得到满意的结果.以图 2 左侧的情况为例,3 个测试用例的状态转移集呈递增关系.算法 1 对此会得到 3 个独立的较短的测试用例,而不是如图 2 右侧所示的一个较长的测试用例.显然这违背了本文开始所提出的原则——两次初始化系统环境的代价远比系统多转换两次状态的代价要大得多.理想情况下的最优解应该是右侧的结果.

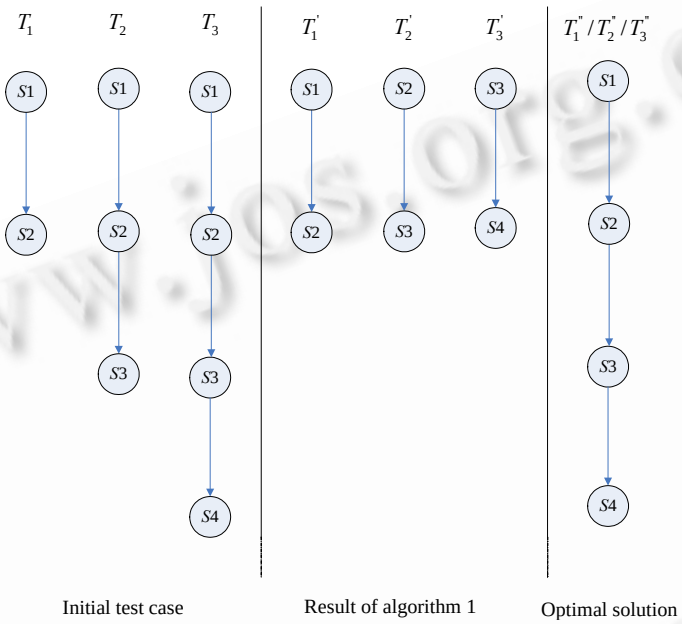

Fig.2 An extreme condition of DTS generation algorithm

图 2 DTS 生成算法的一种极端情况

为了得到理论最优解,我们需要修改算法 1.首先,引入集合 $Replaced = \{T_i : DTS \mid \forall \sigma \in E_I \cdot i \in \sigma.1\}$,它定义了拥有全部 E_I 中状态转换的测试用例的集合.接着,在求出 E_I 之后,我们并不简单执行 $T_i = T_i - E_I$,而是检查集合 *Replaced* 是否为空,如果不为空,则保留其中最长的 T_i,删除剩下元素中的 E_I 部分,这样可以保证留在 *DTS* 中的 E_I 始终在最长的测试用例中;如果 *Replaced* 为空,则表明不存在图 2 的情况.

算法 2. 优化后的简并测试集生成算法.

Input: *Goals*;

Output: *DTS* and *Unavailable*.

Steps:

```
1 Goals:= the set of requirements; Knownedges:=empty set;
2 Unavailable:= empty set; DTS:=empty set;
3 while Goals is not empty do
4     Replaced:= empty set;
5     SELECT and REMOVE p from Goals;
6     CALL model checker to GENERATE a test case T_i for p;
7     if successful then
8         foreach q in Goals do
9             if T_i SATISFY q then
```

```
10                    REMOVE q from Goals;
11                end
12            end
13            E_I = EDGE(T_i) ∩ Knownedges;
14            Replaced = {T_k : DTS | ∀σ ∈ E_I · k ∈ σ.1} ;
15            if Replaced ≠ ∅ then
16                T_M ← the longest T ∈ Replaced ;
17                forall T∈Replaced EXCEPT TM do
18                    T ← T − SEQ(T_i, E_I);
19                end
20            end
21            Knownedges ← Knownedges + EDGE(T_i);
22            DTS ← DTS + T_i;
23        else
24            ADD p to Unavailable;
25        end
26    end
```

3 算法测试与分析

3.1 算法测试

为了验证算法的优化性能,我们在 3.00GHz,512M 内存的 Dell OptiPlex GX280 机器上做了实验,操作系统是 Fedora 6,内核版本 2.6.9.测试需求为分别检验系统中的 sendmail-8.10.0,ppp-2.4.3,krb5-1.4.3 以及 vixie-cron-4.1-3 工具是否存在非法权限提升的可能.我们对以上的程序源代码做了一定的简化,将其中和测试需求无关的外部函数用 stub 函数加以代替,去掉了无关的预处理指令代码,以方便模型检测器进行语法分析.实验结果见表 1,第 2 列为源代码长度,DTS 一列即为算法 2 所得到的测试用例长度.其中,原始测试用例参考了文献[9]描述的方法,为了说明 DTS 算法的效率,我们在实验时保留了文献[9]中生成测试用例时采用的优化措施.最后,我们还对文献[13]的方法进行了实验(结果见表 1 中 Test Tree 一列),作为对比,表 1 的最后一列给出了 DTS 算法结果的化简比例.

从表中数据可以看出,在采用了 DTS 算法优化之后,测试集的总长度比原始测试序列长度缩短了约 17%.由于算法要求被化简的冗余测试序列“片段”在最长的测试序列中保留一份(即算法 1 中的理论最优解情况),因此,实验中原始测试序列最长的 krb5-1.4.3 保留了被化简的冗余测试序列“片段”,从而,它所对应的测试序列的化简效果相对于其他 3 个程序来说不太明显.实验结果表明,由于测试需求为检查各自程序返回时谓词“uid=0”是否为真,而降权操作发生在各个程序主函数的中间或最后,因此,文献[13]的算法并不能对此类冗余进行优化.

Table 1 Result of our experiment

表 1 实验结果

Packages	Src LOC (lines)	Original test case	Test tree[13]	DTS	Ratio (%)
sendmail-8.10.0	1 035	353	353	311	88
ppp-2.4.3	682	370	370	247	67
krb5-1.4.3	2 557	548	548	507	93
vixie-cron-4.1-36	270	165	165	124	75
Total	4 544	1 436	1 436	1 189	83

3.2 算法分析

与现有方法相比,以状态转换作为化简对象生成简并测试集有以下几个特点:

(1) 提高化简效率.在测试中,输入以及各项操作直接对应系统状态的转换,简并测试集的定义可以保证所有重复的操作都只进行一次,从而最大限度地化简了测试集中的冗余.

(2) 节省状态空间.假设两个长度为 n 的测试序列,其中有 m 个状态转换是可化简的,那么简并测试集无论可化简状态所在位置如何,都只需要储存 $2n-m$ 个节点.而基于状态化简的测试集在最好情况下需要 $2n+1-m$ 个节点,在最坏情况下则可能需要储存 $2\times(n+1)=2n+2$ 个节点.可见,当 m 和 n 的数量级相当时,简并测试集对状态空间的节省就相当可观了.

(3) 与测试用例的生成顺序无关.假设两个测试序列 t 和 s,有 $t\subseteq s$,那么无论 t 和 s 哪一个生成在先,上述化简后的测试序列大小都是相同的,也就是说,对于同一个测试需求集 SP,不会出现由于测试需求选择的随机性而导致简并测试集不等价的结果.

(4) 生成的测试集更具有实用性.引入集合 *failure* 包含了所有在实际测试中失败的用例集合,避免了由于化简测试用例而导致的测试失败时无法对应测试需求的现象.

另外,DTS 生成算法是与模型检测器提供的 API 无关的,测试集的化简并不依赖于模型检测器所提供的接口调用方式以及模型检测器所采用的搜索算法.

4 小 结

测试是验证软件质量最有力也是最有说服力的手段,但对于任何软件的开发来说,测试都是相当耗时耗力的,对于安全操作系统来说更是如此.虽然模型检测器的初衷是为了验证模型对时序逻辑属性的满足性,然而它自动记录反例对应路径的功能却为测试用例的自动生成提供了便利.因此,将测试技术和模型检测技术结合起来可以大大提高测试的自动化程度,既可以减少人工,也可以避免由人为因素导致的误差.

本文提出了一种利用模型检测技术化简测试用例的方法,该方法是模型检测器无关的.对算法原型的实验表明,它可以有效地消除测试用例中冗余的需求属性和测试序列中的冗余步骤,对于 Zeng[13]所不能处理的测试序列也可以很好地化简.最后生成的测试集 DTS 中的每个元素对应一个安全需求的测试序列,由于消除了序列中可能的冗余,DTS 减少了测试集时间和空间上的开销.

然而,该算法还有改进的空间,例如对冗余属性的化简本文只采用了最简单的穷举搜索,求交集的部分也需要使用高效的算法.很明显,本文的算法只能对生成的用例进行化简,而没有对模型检测的过程进行优化,而自动测试用例生成中最大的开销来源于模型检测的效率.最坏情况下,模型检测的代价和系统可达状态空间的大小是呈线性关系的,因此模型检测器所采用的搜索策略就显得相当重要.Hamon[10]等人针对如何利用模型检测算法将现有测试用例扩展到未处理的属性做了广泛的研究,成果值得借鉴.下一步的工作将着眼于算法复杂度的优化以及与模型检测器的整合,使其能应用于工业级的系统.

References:

[1] Wimmel G, Jürjens J. Specification-Based test generation for security-critical systems using mutations. In: Proc. of the 4th Int'l Conf. on Formal Engineering Methods: Formal Methods and Software Engineering. London: Springer-Verlag, 2002. 471−482.

[2] Liu L, Miao HK. A framework for specification-based class testing. In: Proc. of the 8th IEEE Int'l Conf. on Engineering of Complex Computer Systems (ICECCS 2002). Washington: IEEE Computer Society Press, 2002. 153−162.

[3] Rusu V, du Bousquet L, Jeron T. An approach to symbolic test generation. In: Proc. of the Int'l Conf. on Integrating Formal Methods, LNCS 1945. Dagstuhl: Springer-Verlag, 2000. 338−357.

[4] Offutt AJ, Liu S, Abdurazik A, Ammann P. Generating test data from state-based specifications. Software Testing, Verification & Reliability, 2003,13(1):25−53.

[5] Huth M, Ryan M. Logic in Computer Science: Modeling and Reasoning about Systems. London: Cambridge University Press, 2004.

[6] McMillan KL. Symbolic model checking [Ph.D. Thesis]. Pittsburgh: Carnegie Mellon University, 1992.

[7] Holzmann GJ. The model checker SPIN. IEEE Trans. on Software Engineering, 1997,23(5):279−295.

[8] Heimdahl MPE, Rayadurgam S, Visser W, Devaraj G, Gao J. Auto-Generating test sequences using model checkers: A case study. In: Proc. of the 3rd Int'l Workshop on Formal Approaches to Testing of Software (FATES 2003). Montreal: Spinger-Verlag, 2003. 42−59.

[9] Beyer D, Chlipala AJ, Henzinger TA, Jhala R, Majumdar R. Generating tests from counterexamples. In Proc. of the 26th Int'l Conf. on Software Engineering (ICSE 2004). Los Alamitos: IEEE Computer Society Press, 2004. 326−335.

[10] Hamon G, Moura L, Rushby J. Generating efficient test sets with a model checker. In: Proc. of the 2nd Int'l Conf. on Software Engineering and Formal Methods (SEFM). Beijing: IEEE Computer Society Press, 2004. 261−270.

[11] Offutt J, Xiong Y, Liu S. Criteria for generating specification-based tests. In: Proc. of the 1st IEEE Conf. on Engineering of Complex Computer Systems. Las Vegas: IEEE Computer Society Press, 1999. 119−129.

[12] Jeffrey D, Gupta N. Test suite reduction with selective redundancy. In: Proc. of the 21st IEEE Int'l Conf. on Software Maintenance (ICSM 2005). Budapest: IEEE Computer Society, 2005. 549−558.

[13] Zeng HW, Miao HK, Liu J. Specification-based test generation and optimization using model checking. In: Proc. of the 1st Joint IEEE/IFIP Symp. on Theoretical Aspects of Software Engineering. Shanghai: IEEE Computer Society, 2007. 349−355.

[14] Chen H, Dean D, Wagner D. Model checking one million lines of C code. In: Proc. of the 11th Annual Network and Distributed System Security Symp. 2004. 171−185. http://www.cs.ucdavis.edu/~hchen/paper/ndss09.pdf

[15] Nie CH, Xu BW. A minimal test suite generation method. Chinese Journal of Computers, 2003,26(12):1690−1695 (in Chinese with English abstract).

[16] Hong HS, Cha SD, Lee I, Sokolsky O, Ural H. Data flow testing as model checking. In: Proc. of the 25th Int'l Conf. on Software Engineering. IEEE Computer Society Press, 2003. 232−242.

[17] Zhang XF, Xu BW, Nie CH, Shi L. An approach for optimizing test suite based on testing requirement reduction. Journal of Software, 2007,18(4):821−831 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/821.htm

附中文参考文献:

[15] 聂长海, 徐宝文.一种最小测试集生成方法.计算机学报,2003,26(12):1690−1695.

[17] 章晓芳,徐宝文,聂长海,史亮.一种基于测试需求约简的测试集优化方法.软件学报,2007,18(4):821−831. http://www.jos.org.cn/1000-9825/18/821.htm

程亮(1982－),男,安徽休宁人,博士,主要研究领域为操作系统安全.

冯登国(1965－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络与系统安全.

张阳(1971－),女,博士,工程师,主要研究领域为操作系统安全.