

基于模型的有限状态机软件动态测试方法

朱玉文 刘 俐 杨家宁 刘万春

(北京理工大学计算机科学技术学院, 北京 100081)

摘 要: 介绍了一种基于模型的软件动态测试方法,该方法通过将被测程序抽象成有限状态机,将测试程序及测试用例的编写集中到单个状态上。在软件结构或逻辑发生改变时,能够通过对旧状态测试程序的重新组合进行新的测试。无需大量修改测试程序从而保存了先前的测试用例。

关键词: 基于模型的测试;有限状态机;自动测试

中图分类号: TP206+.1

引 言

采用计算机系统自动进行软件动态测试取代传统的人工测试,软件测试的效率大大提高。自动测试程序同其他程序一样存在需求变化的问题,当被测试软件发生变化时,测试程序也要同时做改动^[1],本文利用有限状态自动机是为研究有限内存的计算过程和某些语言类而抽象出的一种计算模型。有限状态自动机拥有有限数量的状态,每个状态可以迁移到零个或多个状态,输入字符串决定状态的迁移。有限状态自动机可以表示为一个有向图,一个确定有限状态自动机 M 由 Q 、 δ 、 q_0 和 F 元素构成的五元组^[2]

- (1) 有穷状态集合 Q ;
- (2) 有穷输入字母表 Σ ;
- (3) 转移函数 $\delta: Q \times \Sigma \rightarrow Q$;
- (4) 初始状态 q_0 ;
- (5) 终结状态集合 F , F 包含于 Q 。

自动机从初始状态 q_0 起,逐一读入输入串的每一个字母,根据当前状态、输入字母和转移函数决定自动机的下一步状态。如果输入串结束,自动机处于终结状态集合 F 的某一个状态,表示自动机接受该字符串;否则不接受该字符串。

1 状态测试系统 TAG

我们依据有限状态自动机等理论,结合大量测

试实例,设计了基于模型软件动态测试方法的系统^[3],其工作流程如图1所示,主要工作流程有6步。

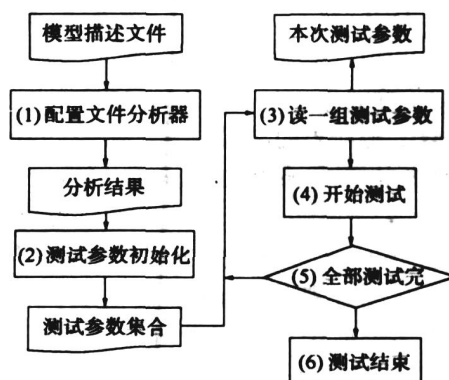


图1 TAG系统工作流程

Fig. 1 Workflow of the TAG system

(1) TAG 配置文件分析器 SpecInterpreter 读入“模型描述文件”。然后对它进行分析。输出“分析结果”。并通过 GrammarInfo 类的对象保存状态集合、动作集合、事件集合、状态转换集合、起始状态以及中止状态集合。

(2) 测试参数初始化, TR (TestRunner 类) 根据“分析结果”进行测试参数初始化。测试参数的初始化包括事件对象初始化和状态测试程序参数的初始化。状态测试参数的初始化,是对所有可能使用到的状态测试程序的参数进行组合。该组合使用一个队列结构保存在 TR 的 `m_parameter` 成员中,即图1中的“测试参数集合”。

(3) TR 从“测试参数集合”中读取组测试参数。

(4) TR 根据这组参数“开始测试”。

收稿日期: 2007-05-10

第一作者: 男, 1950年生, 副教授

E-mail: L904@163.com

(5) TR 在结束测试后判断“测试参数集合”中是否还有剩余的测试参数组合。如果有跳转至第 3 步否则跳转到第 6 步。

(6) 测试结束。

2 状态测试程序接口规范

TAG 系统对状态测试程序进行约束:状态测试程序的返回值为 0 表示测试通过,返回其他值表示对该状态的测试失败。

状态测试程序采用事件出发的方式来通知 TAG 系统进行转换。名称前加“Global \”表示是一个全局事件。例如:在状态描述文件中说明了一个名为 accessAccepted 的事件。若需要在状态测试程序中触发需要使用如下代码:

打开一个定义好的事件

```
HANDLE hEvent = OpenEvent(
EVENT_ALL_ACCESS,
FALSE, T("Global \ accessAccepted"));
```

触发这个事件 SetEvent(hEvent);

2.1 TAG 分析器的实现

TAG 语言的分析由 SpecInterpreter 类来完成,该类有 4 个属性由 5 个成员函数组成:属性 m. grammarInfo 保存有最终分析数据,该数据提供 TestRunner 类进行测试的必要参数;属性 m. dataStack 是个堆栈,用于语法分析时多状态之间传递数据; m. analysisStack 是语法分析时存储单词的堆栈; m. tokens 用于语法分析时存储字符的缓冲区;该类的成员函数 SpecInterpreter(const char * szFilename) 构造函数,用于获得模型描述文件的文件名并将配置文件读入;成员函数 Interpret() 对读入的模型配置文件进行具体分析,其顺序调用 LexicalAnalysis()、GrammarAnalysis() 和 SemanticAnalysis() 按顺序进行词法分析、语法分析和语义分析。其中类成员函数 LexicalAnalysis() 用来进行词法分析。成员函数 GrammarAnalysis() 用来进行语法分析。成员函数 SemanticAnalysis() 用来进行语义分析。该类对模型描述文件的具体分析流程如图 2 所示。

“描述文件”用 LexicalAnalysis() 进行扫描完成词法分析工作。词法分析结果保存在“单词表”中。用 GrammarAnalysis() 进行词法分析得到结果即“单词表”。根据对应语法调用 SemanticAnalysis() 函数进行具体的语义分析。语义分析的结果保存在 m.

grammarInfo 中作为“分析结果”供 TestRunner 使用。

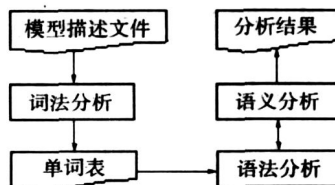


图 2 SpecInterpreter 类解释模型描述文件流程图

Fig. 2 Flowchart of model description file interpretation by the SpecInterpreter class

(1) 词法分析

词法分析器使用一个确定的有限状态机对描述文件进行扫描分析。图 3 给出了词法分析所使用的状态机的状态转换。

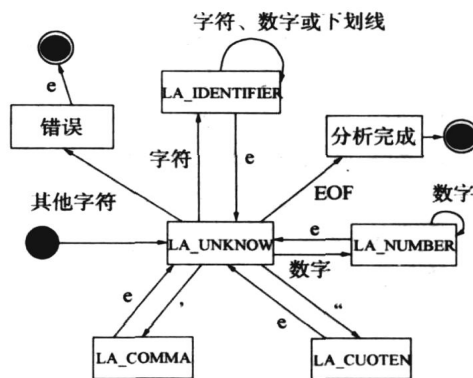


图 3 词法分析状态转换图

Fig. 3 State transition diagram for syntax analysis

图 3 中,词法分析程序首先进入 LA. UNKNOWN 状态,从文件中读取一个字符并放入缓冲区 chTokenBuffer,同时向后移动缓冲区指针 nTokenIterator。之后根据输入字符决定状态转换。当缓冲区中的内容为 并且被确定为符号类型 T 后,将二元组(,)存入 m. tokens 中,同时重置缓冲区指针 nTokenIterator。返回 LA. UNKNOWN 状态继续读取字符。当读取的文件结束符后,词法分析扫描结束。

分析结果是一个二元组(,)符号表。其中是符号, 是该符号的类型。这张符号表使用队列保存在 m. tokens 成员变量供语法分析程序(GrammarAnalysis())使用。

(2) 语法分析

使用下推自动机进行简单的分析。语法分析程序首先从词法分析所产生的符号表(m. tokens)中

逐个读取二元组(,)。并根据每个二元组的型进行状态转换,同时将 压入分析栈(. m. analysisStack) 当发现可归约的短语时,就将分析栈(. m. analysisStack) 的内容传递给语义分析程序,进行语义分析。语义分析完成后,如果该句子可被识别则清空分析栈,继续读取新的二元组。否则,根据语义分析得结果继续进行状态转换。当符号表(. m. tokens) 为空时,语法分析结束。

(3) 语义分析

语义分析器(SemanticAnalysis()) 根据语法分析程序产生的 . m. analysisStack 分析栈进行语义判断。如果分析栈中的内容完整则产生分析结果,如果不完整则通过返回值告知语法分析程序进一步进行语法分析。

(4) 分析结果

SpecInterpreter 将最终的分析结果保存在 . m. grammarInfo 中。 . m. grammarInfo 是 GrammarInfo 类的对象。

2.2 执行参数生成

SpecInterpreter 对描述文件进行分析后,把每个状态测试程序的参数保存到一个散列表(stdehash. map) 中。其中键值是状态的名称,内容是一个一维数组,保存有该状态测试程序参数。这个散列表将用于产生参数表(TestRunner . m. result) 即所有状态测试程序参数的组合。TestRunner . m. result 是一个二维数组。该数组的行是一组参数组合,列是状态的索引值。可采用如下方法求出该参数组合

(1) 计算出所有参数组合的总个数并申请内存空间保存结果,结果保存到 TestRunner . m. result 中;

(2) TestRunner . m. parameter 读取一个状态测试程序的所有数,并把这些参数循环写入 TestRunner . m. result 对应的列中;

(3) 若 TestRunner . m. parameter 还有参数没处理,转(2)继续处理,否则参数表生成完毕。

3 TAG 测试 Windows 的记事本

以 Windows XP 操作系统的记事本程序作为测试对象进行测试,演示测试过程的主要环节。记事本位于 %WINDIR % \ SYSTEM32 \ 。

测试程序的要求是:建立一个新文件,测试再次打开该文件的过程是否正确,以及被保存的文件是

否能被正确的打开。TAG 系统在控制台输出信息会提示测试人员测试是否正确。

3.1 测试过程

使用 TAG 系统进行测试按如下步骤操作。

(1) 建立模型

根据测试过程,建立如图 4 模型,该模型包含 5 个独立的状态。其中初始状态为初始化测试(Init-Notepad) 负责打开一个新的记事本程序。如果存在一个文件则转换到 OpenFile 状态,否则转换到 SaveNewContent 状态。若执行进入了 OpenFile 状态,则打开文件,转入校验文件内容(ValidFileContent) 状态。该状态负责校验文件的内容是否符合要求,并将结果输出在控制台。当校验完成后,转入关闭记事本(CloseNotepad) 状态。这个状态是一个终结状态。如果执行进入了 SaveNewContent 状态,状态测试程序自动将一个文本输入到记事本中并将其保存。而后进入终结状态“关闭记事本程序”。通过向“建立一个新文件”状态的状态测试程序传递不同的参数,来使用不同的测试用例。

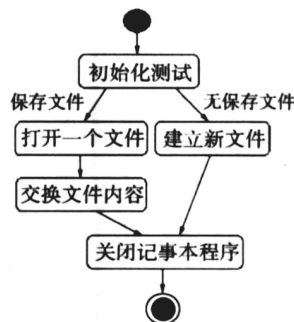


图 4 建立模型流程图

Fig. 4 Flowchart for model generation

(2) 测试用例设计与测试结果数据

采用两个文本作为测试用例。用例一是岳飞的满江红,用例二是文本联通。用例一为普通用例可以触发记事本程序的正确行为。用例二,则会由于其 GB2312 内码编码近似于 Unicode 编码而导致记事本无法在自动打开时做出正确判断而造成显示出错。

运行 TAG 系统后,从控制台得到相关输出。下面分几部分对输出进行详细地解释。

首先从控制台得到如下输出:

Test case Automatic Generator

sage: TAG.exe [Specification File]

Copyright (R) jianing yang, beijing institute of

technology

这些输出的信息是 TAG 的使用方法和版权信息。随后从控制台得到如下输出:

TAG: Test run started, 4 paths loaded.

TAG: New path started with parameter set 0.

该输出信息说明本次测试的参数设置决定了本次测试需要 4 组参数来覆盖全部测试用例。下面测试正式开始,并从控制台得到下面的输出:

TAG: execute action ACT. InitNotepad commend ". . \ Demo Test \ Debug \ InitNotepad.exe 2 "

InitNotepad: Entering State InitNotepad with argument 2

InitNotepad: Try to delete old file

D: \ TEMP \ 1. TXT[FAILED]

InitNotepad: use testcase:

联通

InitNotepad: CreateProcess successfully.

InitNotepad: Process Handle 0x0000004c

InitNotepad: Main Window Handle 0x00030806

event " ENT. SaveNewContent " set

TAG: execute action ACT. SaveNewContent commend ". . \ Demo Test \ Debug \ SaveNewContent.exe 0 "

SaveNewContent: Entering State SaveNewContent

SaveNewContent: Set edit text to

联通

SaveNewContent: Save a new file D: \ TEMP \ 1. TXT[OK]

event " ENT. CloseNotepad " set

TAG: execute action ACT. CloseNotepad commend ". . \ Demo Test \ Debug \ CloseNotepad.exe 0 "

TAG: Arrived at a final state < CloseNotepad > .

从以上输出信息可以获知,测试程序首先从 InitNotepad 状态开始执行并使用参数“2”指示其使用 2 号测试用例。随后系统按照模型描述文件描述的状态机,并根据状态测试程序所触发的事件进行状态转换,然后依次进入 SaveNewContent 和 CloseNotepad 状态。至此一个使用用例二作为内容的文件已经被保存在磁盘上了。

在文件保存完成之后,系统开始使用新的参数继续执行状态机。这次通过向 InitNotepad 状态的状态测试程序传递参数“3”,来指示状态测试程序对刚刚保存的文件进行内容校验。请看系统此时的输

出:

TAG: New path started with parameter set 1.

TAG: execute action ACT. InitNotepad commend ". . \ Demo Test \ Debug \ InitNotepad.exe 3 "

CloseNotepad: Entering State CloseNotepad

InitNotepad: Entering State InitNotepad with argument 0

CloseNotepad: Notepad closed.

联通——(与上面显示相同,不重复给出)

TAG: execute action ACT. OpenFile commend ". . \ Demo Test \ Debug \ OpenFile.

OpenFile: Entering State OpenFile

OpenFile: File opening [OK]

event " ENT. ValidFileContent " set

ValidFileContent: EnteringState ValidFileContent

ValidFileContent: Current edit text is

Valid FileContent: !! Invalidate !!

Valid FileContent: text should be

联通

——(与上面显示相同,不重复给出)

系统按照模型描述文件描述的状态并根据状态测试程序触发的事件进行状态转换,依次进入 OpenFile 状态、ValidFileContent 状态及 CloseNotepad 状态。

从 ValidFileContent 状态进行校验可见,记事本打开以前保存好的文件时出现了错误。本应显示“联通”二字,而实际显示却为“ ”。

3.2 结果显示

记事本程序测试过程反映测试进行中的真实情况:用例一的显示完全正确,用例二的显示出现了错误。这与测试程序所报告的结果吻合。使用用例一的测试完全正确。而使用用例二的测试,再次打开文件时文件内容出现错误。出现错误的原因是记事本程序使用 Windows 的 IsTextUnicode API 来判断打开的文件是否是一个 Unicode 编码文件。在采样较少的情况下,容易出现判断失误导致记事本程序使用错误的编码方式打开文件。

4 结论

TAG 系统对所有状态测试程序的参数进行组合,并用这些组合来逐个测试。但如果状态机中存在至少两个状态且这两组状态仅由一个唯一的转换确定时,在系统进入其中一组状态时,另一组状态所要使用的参数将变得没有意义。因此以后在参数读

取之前,先对参数表进行约简以去除无意义的参数组合。

本文所述的基于模型的测试系统能够很好的应对这种变化,从而提升测试效率。

参考文献:

[1] APFELBAUM L, DOYLE J. Model-based testing[C]

Software Quality Week Conference in May, 1997.

[2] GURARI E. An introduction to the theory of computation[M]. Jones and Barlett Publishers, Inc, 2001.

[3] DALAL S R, KARUNANITHI J N, LEATON J M, et al. Model-based testing in practise [M]. ACM Press, 1999.

Dynamic software testing method based on finite state machine model

ZHU YuWen LIU Li YANG JiaNing LIU WanChun

(College of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)

Abstract: In this paper, model-based testing approach is introduced. By abstracting a finite state machine (FSM) from the System under Testing (SUT), this approach can increase the test coverage by developing test cases for each states and transition, making the implementation of test tool and cases easier. When the SUT changes, only the test tool need to be rebuilt while the previous test cases can still be used without modification.

Key words: model-based testing; FSM; auto testing

(上接第 118 页)

A secure routing protocol of wireless sensor networks

LI ShaoHeng ZHANG Kun WANG CuiRong

(Northeast University at Qinhuangdao, Qinhuangdao Hebei 066004, China)

Abstract: A TEEN protocol-based security-efficient protocol STEEN is presented. This protocol addresses the problems of inadequate consideration for security in some of existed routing protocols for wireless sensor networks. The protocol, which takes saving of energy consumption and improving of routing security as its design targets, solves the problems of authentication and secure communication between nodes through pre-distribution of key and the random key pair-wise management. Analysis on security indicates that this protocol can be used to defend many attacks at network layer.

Key words: wireless sensor network; routing protocol; security; key management