嵌入式 C编译器测试用例生成工具的设计*

陈国梁[1] 郭东辉[1,2]

([1]厦门大学物理系 厦门 361005 [2]厦门大学电子工程系 厦门 361005)

摘要: 首先提出了一种嵌入式编译器测试验证方法, 即基于串口传输的变量值验证法, 在此基础上设计了一款针对嵌入式 C编译器测试的测试用例生成工具 ECPAG。该工具根据嵌入式 C语法, 采用随机算法产生符合规则的任意语法组合, 采用概率算法限定各语法要素的生成概率, 成功地将基于深度优先搜索的有向图拓扑排序方法应用于函数随机调用中的递归问题的解决。工程应用表明: 该自动化工具生成的测试用例集合能够较好地覆盖嵌入式 C语法, 达到 75% 以上的块测试覆盖率。

关键词: 嵌入式 C编译器 测试用例生成 递归调用

Design of an Automatic Test Case Generator for Embedded C Compilers

CHEN Guoliang[1], GUO Donghui[1,2]

([1] Department of Physics, Xiamen University, Xiamen 361005, China; [2] Department of Electronic Engineering, Xiamen University, Xiamen 361005, China)

Abstract The paper firstly proposes an approach to test and verify embedded C compilers, where the variables outputted from the tested program are sent back via serial port to verify the correctness of the compiler. And then an automatic test case generator for embedded C compiler which is named ECPAG (Embedded C Program Automatic Generator) is designed. The ECPAG uses a stochastic algorithm to randomly combine different syntax according to embedded C language syntax so as to generate test cases (in our context – a program), and utilizes a probabilistic algorithm to restrict the generation probability of each syntax. The generator also utilizes a depth-first search on the function call graph to avoid the appearance of recursive calls in random invocations. Practice evaluation shows that test cases generated by the automatic tool can cover syntax and semantics of embedded C language well, and a block-coverage in the excess of 75% is achieved.

Keywords Embedded c compiler, Test case generating, Recursive calls

1 引言

嵌入式 C编译器作为嵌入式 C系统开发的重要工具, 广泛应用于嵌入式研发、生产以及应用的各个领域, 其可靠性的高低直接关系到产品的成败, 因而, 对嵌入式 C编译器功能准确性进行测试成为了编译器开发商们的迫切需求。嵌入式 C编译器测试是一项耗费巨大、困难复杂的工程, 需要设计规模庞大的测试用例集[1]。传统的手工编写用例的方式不能满足编译器测试这类软件测试项目的需求, 通过自动化的方式能在较短时间内产生大量符合要求的测试用例, 节约测试成本[1-3]。

目前用于编译器测试的用例生成工具研究主要是面向 ANSI-C/C++本地编译器和其特定功能的测试需求[3,4], 文献[5, 6]设计了能够自动覆盖所有程序调用规约的用例生成工具; 文献[7]设计了一款 ANSI-C程序自动生成工具来检验编译器对于 volatile修饰的变量执行结果是否正确; 文献[8]设计了一款测试

本文于 2010-05-07收到。

* 基金项目: 福建省自然科学基金 (2008J0220), 厦门市科技计划项目(3502Z20093002)。

浮点类型变量全部应用的程序自动生成工具; 文献[9]开发了一款面向对象编译器测试工具 O-OCTT, 实现了 C++编译器测试用例的自动生成及测试过程的自动化。由于嵌入式 C语言和 ANSI-C/C++存在一定的区别, 现有的测试用例生成工具无法直接应用于嵌入式 C编译器测试中。

嵌入式 C编译器测试与 ANSI-C编译器测试的区别体现在: (1)嵌入式系统自身硬件资源的局限, 决定其测试方法与普通软件测试有很大的不同[10], 如何选择一种合适的测试用例执行验证方法来检验嵌入式编译器功能正确性, 成为亟待解决的问题。(2)嵌入式 C语法会根据实际应用需要对 ANSI-C进行扩展和限制[11], 如增加嵌入式汇编、中断服务函数和内建函数, 不支持多维数组, 不支持长字节的变量, 不支持函数的递归调用。其中, 由于嵌入式微处理的堆栈空间有限, 对应的嵌入式语法一般不支持函数的递归调用, 因此如何在测试用例中避免函数的递归调用, 成为了嵌入式编译器测试中用例生成技术面临的一个难点[12]。文献[9]通过在函数调用树中增加"下层函数不能调用上层函数"的约束避免函数递归; 文献[12]将函数按照生成时间的先后排成一个链表, 在调用时每个函数只能调用链表中位于它之前的函数, 不能调用位于它之后的函数; 文献[13]给 Java的类方法赋予不同的优先级, 级别高的类方法才能调用级别低的类方法。这些方法虽能有效地防止递归, 但是人为增加了调用约束, 无法真实地反映函数间的随机调用, 造成对函数调用语法点测试不够完整。因此, 有必要寻找一种既能有效避免递归又能尽可能实现随机调用的方法。

本文首先提出了一种嵌入式编译器执行验证的方法, 在此基础上设计了一款针对嵌入式编译器测试的用例自动生成工具 ECPAG(Embedded C Program Automatic Generator), 给出了该工具的总体框架和关键算法, 并对 ECPAG在实际测试项目中的应用结果进行了分析, 最后一节对本文工作进行总结。

2 基于串口传输的变量值验证

针对嵌入式硬件资源的局限, 本文提出了一种基于串口传输的变量值验证测试策略来检验编译器功能的正确性。图 1是基于该策略建立的嵌入式交叉测试环境, 测试用例集合经过宿主机端的编译器编译链接后, 生成的可执行代码下载到目标机端的硬件设备上运行, 获得测试用例的运行结果, 这些运行结果不是在目标机端进行分析, 而是通过串口传输回宿主机端保存。宿主机端的数据比较工具收集这些运行结果, 通过和参考编译器运行结果进行比较的方式[10, 13, 14]验证待测编译器的功能是否正确, 并将测试结果记录到测试报告中。

为了便于分析待测编译器功能的正确性, 并根据错误准确定位待测编译器在语法语义处理上存在的缺陷, 我们在测试用例中每个函数结束的位置植入变量输出语句, 将函数所用到的所有全局变量和局部变量的数值传送回宿主机端, 宿主机端的数据比较工具对比待测编译器和参考编译器运行后的变量结果值, 如果发现变量运行结果不一致, 测试者便可以结合测试用例检查对该变量执行的相应语法操作是否存在问题, 发现待测编译器存在的缺陷。为此, 我们针对不同的变量类型, 定义了相应的变量输出语句用于传输测试用例中各种数据类型的变量, 输出语句的详细介绍如表 1所示。

3 ECPAG的设计实现

基于上节提出的变量值验证测试策略, 我们设计了一款针对嵌入式 C编译器测试的用例自动生成工具 ECPAG, 产生用于嵌入式编译器功能测试的测试用例集合。

表 1 变量输出语句介绍

语句形式	功 能	传输字长(byte)
sendChar(char data);	传输 signed char型变量	1
sendUChar(unsigned char data);	传输 unsigned char型变量	1
sendInt(int data);	传输 signed int型变量	2

续表

语句形式	功　能	传输字长 (byte)
sendUInt(unsigned int data);	传输 unsigned int型变量	2
sendLong(long data);	传输 signed long型变量	4
sendULong(unsigned long data);	传输 unsigned long型变量	4
sendFloat(float data);	传输 Float型变量	4
sendDouble(double data);	传输 Double型变量	4
sendString(unsigned char* data);	传输 String类型变量	可变长度
sendComType(unsigned char* str, int len);	传输复合类型变量	可变长度

3.1 ECPAG 总体框架

ECPAG的总体框架如图 2所示, 该工具主要由六个部分组成: 用户界面、初始化模块、生成模块、美化模块、输出模块以及自定义参数表。使用者通过用户界面设定测试用例的规格, 如每个测试用例的最大函数个数、最大语句嵌套深度、语句生成概率等, 定义好的参数存储于自定义参数表中。生成测试用例时, 初始化模块提取自定义参数表中的用户配置, 将其传递给生成模块中的对应参数, 生成模块根据用户设定的测试用例规格参数, 调用测试用例生成方法进行语法的随机组合和嵌套, 生成符合用户要求的测试用例, 并将测试用例规格参数以注释的形式附加到程序的起始位置。美化模块按照 C程序编程风格, 对生成的测试用例内容进行缩进排版, 生成便于测试者阅读的测试用例。最后, 输出模块对生成的测试用例进行统一命名, 并将用例集合存放到测试者指定的目标文件夹中。

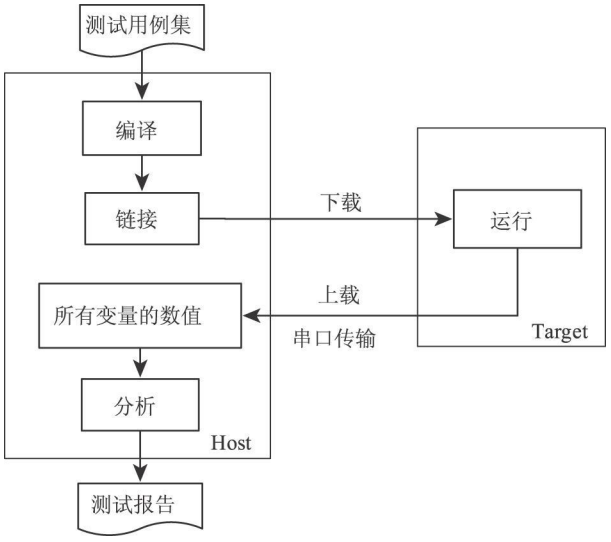


图 1　嵌入式交叉测试环境

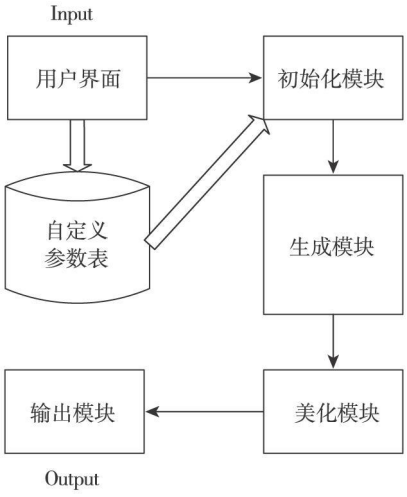


图 2　ECPAG总体框架

3.2 测试用例生成方法

采用随机算法和概率算法相结合的思想组合嵌入式 C语法, 设计了测试用例的生成方法。该方法遵循“函数—语句—变量—数据类型”的思路产生测试用例结构, 将语法点按类别封装成语法模块, 把具有类似功能的语法模块归类成组, 总共有三个组: 语句组、变量组和数据类型组。测试用例生成中各语法模块调用流程如图 3所示, 测试用例首先调用函数模块生成函数框架, 函数模块随机调用语句组中的任意一种语句模块生成相应的语句框架, 在语句深度未达上限的情况下, 语句模块还可以随机调用该组中任意一种语句模块继续生成语句框架。当所有语句框架生成完毕, 各语句模块随机选择变量组中的全局变量或者局部变量

填充到框架中的适当位置,变量又随机选择一种数据类型,根据对应的数据类型对该变量进行初始化。在测试用例的生成过程中,各语法模块的组合和嵌套都是随机挑选的,函数模块可以随机调用任意一种语句模块,语句模块随机挑选其他语句模块和变量组中的各种变量,变量随机选择数据类型,从而可以生成大量不同语法组合的测试用例,达到任何语法点和语法组合都可能测试到的效果,排除了人为设计用例造成的语法组合不够完整的影响。

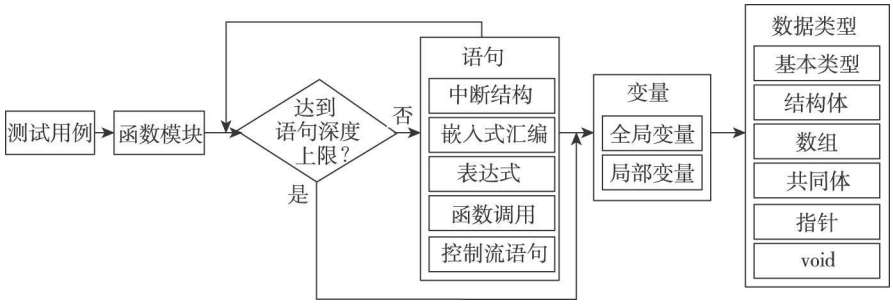


图 3 测试用例生成中语法模块调用流程

针对嵌入式 C对 ANSI– C的扩展和限制,引入概率限定的思想,通过编写脚本设定各语法点在测试用例中出现的概率,当随机算法挑选到某种语法后,便会读取脚本中测试者对该语法的概率设置,从而决定该语法结构被选中的可能性有多大。通过这种方式,我们可以增大一些重要语法如 bit数据位操作、嵌入式汇编、中断服务函数和内建函数的产生概率,限制大规模数组、长字节变量、多层循环结构嵌套的出现,从而有针对性地测试某些语法点。

3.3 函数调用算法

由于测试用例生成方法采用的是随机算法,调用函数和被调用函数是随机选择的,所以不可避免会在测试用例中出现函数的递归调用。函数的递归调用是指在调用一个函数的过程中又出现直接或者间接调用该函数本身[15],图 4(a)是一种函数直接递归调用的结构,在调用 fun1()的过程中又出现了调用它本身的情况,图 4(b)是一种函数间接递归调用的结构,在调用 fun1()的过程中调用了 fun2(),而在调用 fun2()过程中又调用了 fun1(),这些都会在函数调用中形成调用环路,从而使编译器进入函数调用的死循环。嵌入式微处理由于堆栈空间有限,一般不支持函数的递归调用。

```
fun 1()
{
        …
        fun 1();
        …
}
```

(a) 函数直接递归调用

```
fun 1()
{
        fun 2();
        …
}
fun 2()
{
        fun 1();
        …
}
```

(b) 函数间接递归调用

图 4 函数递归调用示例

为了防止函数的随机调用中出现递归,我们采用基于深度优先搜索的有向图拓扑排序方法来解决。将有向图理论应用于函数调用中,用有向图表示函数间的随机调用关系,有向图中的节点表示函数,有向图中的有向弧线表示两个函数之间的调用关系,箭头由调用函数指向被调用函数,因此,函数的递归调用便可以表示为有向图的节点循环。在有向图理论中有判定循环的定理:有向图中有环的产生,当且仅当对有向图进行深度优先搜索发现反向边[16]。根据该定理,要使函数调用中不出现递归调用,只要保证函数调用所对应的有向图不存在环,即对该有向图进行深度优先搜索没有发现反向边。我们设计了一种防止递归的函数调用算法,实现如下:算法执行前,初始化有向图 G 来表示函数间的随机调用关系,定义邻接矩阵 $edges$ 存储 G 中的有向弧线信息,邻接矩阵的维数随函数个数增加而动态增长,如果函数 i调用函数 j,$edges[i][j] = 1$,

否则为 0。在函数调用过程中, 假设当前有 n个函数, 调用函数为 j, 首先判断是否生成一个新函数进行调用, 如果是的话, 由于调用新生成的函数不会导致有向图中环的产生, 只需在有向图 G中增加节点 (n+ 1)和有向弧线 $\vec{i}$ (n+ 1), 同时 edges维数增 1, edges[i] [n+ 1] = 1; 如果调用已经生成的函数, 假设为 j, 将有向弧线 $\vec{i}$ j添加到 G中, 设 edges[i] [j] = 1, 以 i为起始节点对有向图进行基于深度优先搜索的拓扑排序, 生成包含所有节点的拓扑序列 L。从 L的末节点开始搜索 L是否存在反向边, 如果 L存在反向边, G中有循环产生, 函数调用出现递归, 则删除先前添加的有向弧线 $\vec{i}$ j, 设 edges[i] [j] = 0, 重新挑选被调用函数; 如果 L不存在反向边, 则 G中没有环的产生, 函数调用没有出现递归, 调用结束。

由于调用函数和被调用函数是根据随机算法在有向图中随机选择的, 既可以调用已经生成的函数, 也可以调用一个新生成的函数。只要保证不出现调用循环, 即可构成函数调用关系, 从而在防止函数递归调用的前提下尽可能地实现函数的随机调用。

4　ECPAG的应用与评价

4.1　ECPAG的应用

我们将 ECPAG应用于某嵌入式 C编译器的功能测试项目中, 采用目前常用的参考编译器对比方式[10, 13, 14]设计了自动化测试流程, 验证待测编译器功能的正确性, 同时结合代码覆盖测试工具分析编译器源码的执行情况, 客观评价 ECPAG的有效性[17]。

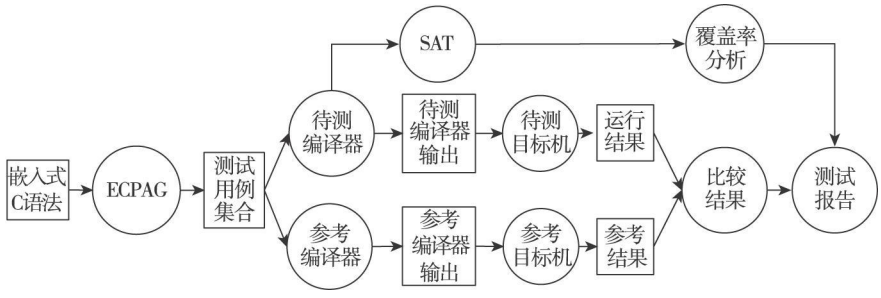


图 5　嵌入式 C编译器自动化测试流程

嵌入式 C编译器自动化测试流程如图 5所示, 引入一款经过长期实践证明的成熟的商用编译器作为参考编译器, 利用块覆盖测试工具 SAT[17]分析代码的块覆盖率情况。首先, 调用 SAT对待测编译器的源码进行结构分析, 划分各基本块并植入跟踪探针, 编译生成插装后的待测编译器程序; 将 ECPAG产生的测试用例集合分别输入插装后的待测编译器和参考编译器编译, 生成的可执行代码分别下载到待测目标机和参考目标机上运行, 获得测试用例的运行结果并通过串口传送回主机端, 主机端的数据比较工具对比两种编译器的运行结果是否相同, 如果相同, 说明待测编译器能实现和参考编译器相同的功能, 测试通过, 反之, 则说明待测编译器可能存在错误。在待测编译器运行过程中, SAT收集编译器源码中各基本块的执行信息, 计算各语法文件的块覆盖率。

4.2　覆盖率评价

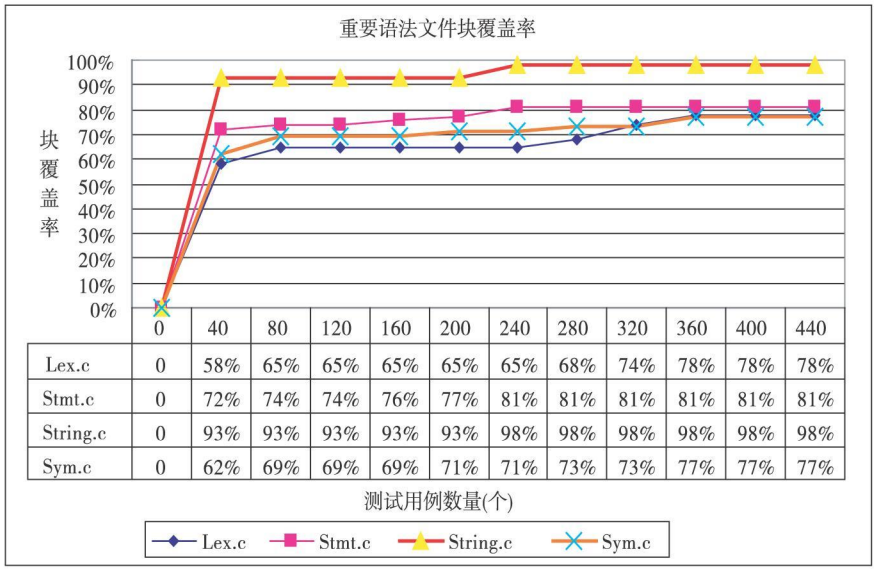
随机选取编译器源码中四个重要语法文件进行块覆盖率分析: Lex. c、Stmt. c、String. c、Sym. c。其中, Lex. c负责测试用例中的词法分析, Stmt. c负责各种语句类型的语义分析, String. c负责各种数据类型的识别和处理, Sym. c负责测试用例中的各种标识符的管理, 四个语法文件的功能介绍如表 2所示。

表 2　重要语法文件功能介绍

文件名	功能说明
Lex. c	词法分析
Stmt. c	语句类型的语义分析
String. c	数据类型的识别和处理
Sym. c	标识符的管理

图 6是运行 ECPAG生成的 440个测试用例后

四个重要语法文件的块覆盖率增长情况。从图 6可以发现: 当测试用例数量达到 40个的时候, String. c的覆盖率已经达到 93%, 说明 ECPAG能够比较完整地生成待测编译器所支持的大部分数据类型, 而其它语法文件的块覆盖率相对较低一些; 随着测试用例数量的增加, 语法组合的种类和数量逐渐增加, 各语法文件的块覆盖率逐渐提高; 当测试用例数量达到 360个的时候, 各语法文件的块覆盖率达到最高值, Lex. c为 78%, Stmt. c为 81%, String. c为 98%, Sym. c为 77%, 通过覆盖率情况可知 ECPAG生成的测试用例能够较好地覆盖待测编译器的源代码, 以较少的测试用例数量达到较高的测试覆盖率 (75% 以上), 从而达到测试嵌入式编译器在语法语义处理是否正确的目的。



	0	40	80	120	160	200	240	280	320	360	400	440
Lex.c	0	58%	65%	65%	65%	65%	65%	68%	74%	78%	78%	78%
Stmt.c	0	72%	74%	74%	76%	77%	81%	81%	81%	81%	81%	81%
String.c	0	93%	93%	93%	93%	93%	98%	98%	98%	98%	98%	98%
Sym.c	0	62%	69%	69%	69%	71%	71%	73%	73%	77%	77%	77%

图 6 重要语法文件的块覆盖率增长情况

从图 6还可以发现: 当测试用例数量超过 360个的时候, 即使增加新的测试用例, 由于新生成的语法类型、组合与先前的用例出现重复, 块覆盖率无法继续提高。通过对源代码的分析, 我们得出覆盖率无法继续上升的三点原因: 第一, 源代码中存在大量的不可达代码块和不可达函数, 这些代码的存在是阻碍块覆盖率进一步提高的主要原因; 第二, ECPAG只能生成符合编译器语法文档的测试用例, 无法生成错误的测试用例, 导致编译器源码中涉及错误处理的相应代码片段无法被执行; 第三, ECPAG所支持的语法类型还不够完整, 无法生成一些较为复杂的语法结构, 如函数返回类型为 struct, 函数指针等, 这些不足之处有待今后的工作继续改进。

5 结束语

本文提出了一种嵌入式编译器测试验证方法, 即基于串口传输的变量值验证的测试策略, 并在此基础上设计了一款嵌入式编译器测试用例自动生成工具 ECPAG, 该工具采用随机算法进行任意语法组合, 达到任何测试点都可能测试到的效果, 采用概率算法限定语法组合的生成概率, 从而有针对性地测试某些语法点, 并将基于深度优先搜索的有向图拓扑排序方法用于函数随机调用中的递归问题的解决。我们将该工具应用于某嵌入式编译器的测试项目中, 测试结果表明: 该工具生成的测试用例集合能够较好地覆盖待测编译器语法, 达到 75% 以上的块测试覆盖率, 基本达到测试嵌入式编译器在语法语义处理是否正确的目的。

参 考 文 献

[1] A. S. Boujarwah and K. Saleh. Compiler test case generation methods: a survey and assessment[J]. Information and Software Technology, 1997, 39(9): 617-625

[2]Ron Patton. 软件测试[M]. 北京: 机械工业出版社, 2002.

[3]Morgan, Olwen. Certified Testing of C Compilers for Embedded Systems[C]. Automotive Electronics, 2007 3rd Institution of Engineering and Technology Conference, 2007.

[4]Flash Sheridan. Bibliography for "Practical Testing of a C99 Compiler Using Output Comparison" [EB/OL]. http://web.me.com/flashsheridan/compiler_testing_bibliography.html, 2007, 2010.

[5]Christian Lindig. Random testing of C calling conventions[C]. Proceedings of the 6th International Symposium on Automated and Analysis-Driven Debugging, AADEBUG 2005, 2005.

[6]Mark W. Bailey, Jack W. Davidson. Automatic Detection and Diagnosis of Faults in Generated Code for Procedure Calls[J]. IEEE Transactions on Software Engineering, 2003, 29(11): 1031-1042.

[7]Eric Eide, John Regehr. Volatiles are miscompiled, and what to do about it[C]. Proceedings of the 8th ACM International Conference on Embedded Software, EMSOFT'08, 2008.

[8]Bridgitte Verdonk, Annie Cuyt, Dennis Verschaeren. A precision- and range-independent tool for testing floating-point arithmetic I: basic operations, square root, and remainder[J]. ACM Transactions on Mathematical Software, 2001, 27(1): 92-118.

[9]何群, 陈英, 周激流. 面向对象语言编译器自动测试平台[J]. 计算机工程, 2005, 31(14): 99-101.

[10]余盛季. 嵌入式软件系统测试平台研究[D]. 成都: 电子科技大学, 2004.

[11]Popovic, M., Kovacevic, V., Temerinac, M. Popovic, M., Kovacevic, V., Temerinac, M. Software testing concept used for MAS/C-compiler[C]. Euromicro Conference, 2000. Proceedings of the 26th, 2000.

[12]孙文灿, 陈英, 史晋, 黄菲. O-OCTT平台测试用例的自动生成[J]. 计算机应用, 2003, 23(12): 46-47.

[13]Takahide Yoshikawa, Kouya Shimura, Toshihiro Ozawa. Random Program Generator for Java JIT Compiler Test System[C]. Proceedings of the Third International Conference on Quality Software, 2003.

[14]Flash Sheridan. Practical testing of a C99 compiler using output comparison[J]. Software - Practice and Experience, 2007, 37(14): 1475-1488.

[15]谭浩强. C程序设计(第二版)[M]. 北京: 清华大学出版社, 2002.

[16]Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to Algorithms, Second Edition[M]. New York: McGraw-Hill, 2001.

[17]徐晓峰, 陈艳, 林晓鹏, 李伊飏, 郭东辉. 一种块覆盖软件测试自动化工具的设计与实现[J]. 智能系统学报, 2008, 3(Supp1): 198-202.

作者简介

陈国梁, 男, (1984-), 硕士研究生, 研究方向: 软件自动化测试。

郭东辉, 男, (1967-), 教授, 博士生导师, 研究方向: 人工智能、网络通讯、软硬件协同设计等方面。