

嵌入式软件功能路径测试用例自动生成研究

胡燕京^{1,2}, 张毅坤¹, 朱 伟¹, 于全喜¹

(1 西安理工大学 计算机科学与工程学院, 陕西 西安 710048; 2 武警工程学院 电子技术系, 陕西 西安 710086)

摘 要: 根据嵌入式软件的结构特点尝试了一种适合于嵌入式软件测试实际的功能路径测试方法. 通过分析功能路径上的结点谓词和功能约束条件, 用一种简洁、快速的算法建立和求解输入变量的约束系统, 生成功能路径测试用例. 实践证明, 该测试用例可同时测试嵌入式软件的功能和结构, 测试效率高、生成代价低且测试较为全面, 能有效降低嵌入式软件开发、测试和维护的成本.

关键词: 嵌入式软件; 功能路径测试用例; 功能点

中图分类号: TP311

文献标识码: A

文章编号: 1000-7180(2010)01-0081-05

Research on Automate Generation Method Embedded Software Function-path Test Cases

HU Yan-jing^{1,2}, ZHANG Yi-kun¹, ZHU Wei¹, YU Quan-xi¹

(1 School of Computer Science & Engineering, Xi'an University of Technology, Xi'an 710048, China;

2 Department of Electronics, Engineering College of the APF, Xi'an 710086, China)

Abstract: According to the structural characteristics of embedded software, this paper try to fit a practical function-path method of embedded software testing. By analyzing the node predicates and function constraints on the function paths, the author use a simple, fast algorithm to build up and solute a restraint system of input variables, generating function-path test cases. Practice has proved that the test cases can test the functions and structures of embedded software at the same time. The high-efficiency, low price and more comprehensive of the test cases can lower the costs of the development, testing and maintenance of embedded software.

Key words: embedded software; function-path test case; function point

1 引言

随着嵌入式软件规模和复杂度的日益增大, 为其快速有效生成测试用例的要求变得越来越迫切, 文献[1]中建立了嵌入式软件的类测试框架, 研究从 UML 状态图生成测试用例的方法, 但是尚未解决类状态迁移的冲突情况, 测试用例生成的自动化程度不高; 文献[2]中测试用例是根据运行剖面随机生成的, 然而运行剖面的构造比较繁琐, 而且随机生成的测试用例测试针对性不强. 文献[3]等一系列文章都对动态生成测试用例的方法进行了大量研究, 但

嵌入式软件的一个显著特点是运行环境和硬件紧密相关, 为了生成测试用例而不断运行被测程序的动态生成方法开销较大. 针对以上方法的不足, 文中根据 C 语言嵌入式软件的特点, 将被测软件划分成功能路径, 研究用静态手段生成功能路径测试用例的方法. 该方法将功能与结构测试同时进行, 以尽可能少的测试执行最大限度的发现问题, 增加了测试的透明度, 一方面可以尽早发现需求、设计与实现的不一致问题, 另一方面可以对每一个功能的内部逻辑实现清晰的把握, 使得测试效率显著提高的同时也降低了嵌入式软件测试用例生成的难度, 符合嵌入

式软件测试的实际,减少了测试人员的工作量。

2 功能路径测试用例生成过程

文中通过文档解析和源代码静态分析两方面信息的提取与对比,建立和求解功能路径约束系统,对同时满足功能需求和结构需求的功能路径测试用例生成进行了有益的探索。

定义 1 功能点:一个或一类功能的抽象描述,嵌入式软件从功能方面看,就是所有功能点的集合,它将被测软件划分成一个个独立但相互联系的测试区域。

定义 2 功能路径:功能点下包含的一条或若干条基本路径,这些路径只与实现特定的功能相联系,嵌入式软件从结构方面看,就是所有功能路径的集合。

测试的核心是为每一条功能路径生成测试用例。由于每一个功能点和它所涉及的路径规模都不太大,可以在提高测试用例生成效率的同时降低生成的难度。据此生成的测试用例兼顾外部功能与内部结构,实现了功能和基本路径的覆盖,适应了嵌入式软件高稳定性的要求。测试用例的生成总体描述如图 1 所示。

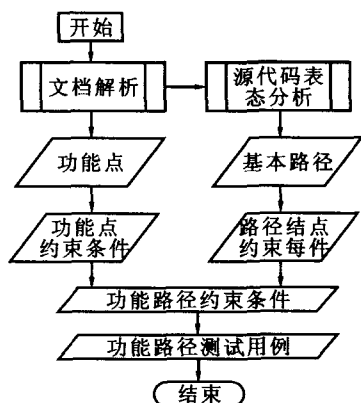


图 1 测试用例生成过程总体描述

测试用例生成的依据从文档解析和源代码静态分析结果中提取。文档解析提取出功能点、相关函数以及功能点约束等重要信息,一方面确定了对被测源代码进行功能划分的粒度,另一方面为测试做功能约束条件的准备。静态分析以功能点规定的范围对被测源代码进行扫描,通过关键字匹配划分出功能点包含的基本路径集,将这些信息与文档解析结果相结合,形成功能路径约束条件集。文中重点论述功能路径测试用例的生成方法,关于功能路径划分的详细内容将另文介绍^[4]。

3 功能路径测试用例生成方法

3.1 基本原理

我们之所以不采用路径测试方法,是因为即使一个不太复杂的程序,其路径数目都十分巨大,这就使得完全的路径测试不可能在有限时间内完成。通过研究 C 语言嵌入式软件的结构特点,我们发现功能路径是完成某个功能时程序必经的路径,这样的路径只和某个功能相关,因此路径数目会大量减少。为了弥补简化路径造成的不足,引入功能约束条件,将功能与路径约束相结合,可有效提高测试充分性。

定义 3 功能路径测试用例:设程序 Q 在测试用例 T 驱动下执行, $P = \langle b_0, b_1, \dots, b_n \rangle$ 是实现功能 F 所经过的路径,则称 T 是满足功能 F 和路径 P 的功能路径测试用例。

功能路径 P 上每一个结点 $b_i (0 \leq i \leq n)$ 存储着和其对应的源代码基本程序块中的信息,测试用例生成算法将读取每一个结点上的谓词表达式将其组合、转换,生成路径约束条件。结点的存储结构定义如下:

```

struct B_block
{
    int B_start_line; // 块起始行号
    int B_type; // 类型 0:顺序 1:分支 2:循环
    char B_expression; // 块中所有谓词表达式
    int BB_link; // 后继是否是分支结点:1 是 0 否
}
  
```

将每个功能点的约束条件和路径上每个结点的约束条件组成合取范式,降低了冗余测试用例的生成。设 $F = \{f_i \mid i = 1, 2, \dots, n\}$ 为功能点集合, $W_{f_i} = \langle b_{ij_0}, b_{ij_1}, \dots, b_{ij_m} \rangle$ 为功能点 f_i 下的路径结点约束条件集合 (f_i 下包含 $j \geq 1$ 条路径), 则 f_i 的功能路径约束条件集为 $f_i \cap b_{ij_0} \cap b_{ij_1} \cap \dots \cap b_{ij_m}$, 以此建立和求解约束系统,可获得既覆盖功能点 f_i 又覆盖路径 W_{f_i} 的功能路径测试用例。

测试用例以一定的结构存储,便于用例的管理、运行及测试执行后的结果分析。测试用例的存储结构定义如下:

```

Struct FPTTestData
{
    int FPTnum; // 用例编号
    char inpara; // 测试输入数据
    int inter_point; // 用例执行中断的断点位置,用于定位错误、异常或中断
    char real_fpath; // 用例执行实际经过的功能路径,用
  
```

于测试结果分析

测试用例生成算法分为约束条件收集、约束系统建立和约束系统求解三部分,流程如图2所示。

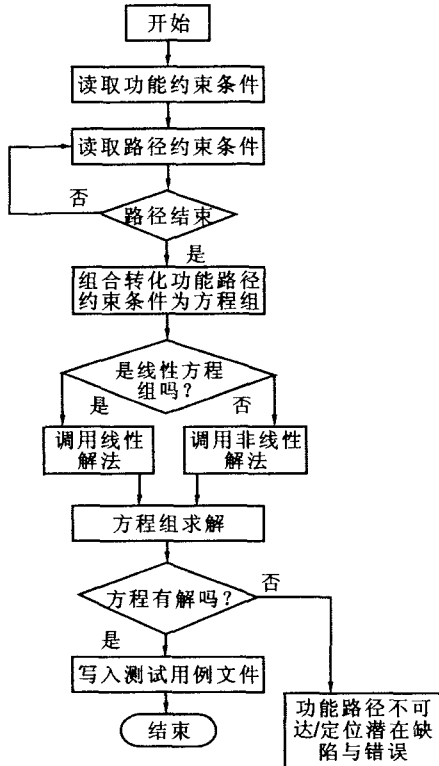


图2 测试用例生成算法流程

3.2 生成算法

输入:功能点 f_i 、功能路径 W_{f_i}

输出:功能路径测试用例集

$GenFPTData(f_i, W_{f_i})$

{

3.2.1 约束条件收集

(1) 读取文档解析结果中对功能点的约束描述 $function_test$ 数组中的元素;

(2) 读取静态分析结果中与功能点对应的基本路径集;

while(基本路径集不为空或路径未结束)

遍历 W_{f_i} 上第一条路径上的每一个路径结点, 读取 $B_expression$ 数组中的路径结点信息;

将 $function_test$ 和 $B_expression$ 数组中的元素组合成功能路径约束条件集 $FPTerm$ 数组;

访问下一条路径;

3.2.2 约束系统建立

(1) 将 $FPTerm$ 数组中的每一个功能条件根据输入输出关系及函数接口信息生成功能子域不等

式;

(2) 将 $FPTerm$ 数组中的每一个路径约束条件转化成一个路径约束不等式;

(3) 将所有不等式联立成一个不等式组;

(4) 根据文献[5]提供的方法将不等式组转化为方程组;

3.2.3 约束系统求解

(1) 若约束系统为线性方程组,则调用线性最小二乘问题的广义逆法^[6];

若为非线性方程组,则调用非线性方程组的 Monte Carlo 法^[6];

(2) 若有解,则将解 $X = (x_{ij0}, x_{ij1}, \dots, x_{ijn-1})^T$ 存入 $inpara$ 数组,此即为测试数据;

若无解,则打印出现异常的步骤;

(3) 将用例写入测试用例文件 $TData.xml$

}

测试用例的生成实际上转化为对约束系统求解的搜索优化问题. 约束条件收集是一个重要而基础的工作. 数组 $function_test = \{fexp_1, fexp_2, \dots, fexp_i\}$, 其中每一个元素都是对功能点某个子域的一个约束描述, 遍历数组中各个元素, 将它们组合成一个析取式 $fexp_1 \vee fexp_2 \vee \dots \vee fexp_i$, 生成一个面向功能的完整约束, 采用域分析策略确定合法取值和非法取值, 其中合法取值为最大值、最小值、正常值, 非法取值为无效值、越界值.

$B_expression$ 数组中存储着一条功能路径上所有结点的约束表达式, 设 $expr$ 为结点上具有明确取值要求的表达式, 则有 $expr_1 > expr_2, expr_1 \geq expr_2, expr_1 < expr_2, expr_1 \leq expr_2, expr_1 = expr_2, expr_1 \neq expr_2$ 以及由其组成的与、或、非等多种关系, 收集每一个结点上所有这类表达式, 将其分解, 转化为 $expr op 0$ 的形式, 其中 $op \in (<, \leq, >, \geq, =, \neq)$. 路径 i 上的功能子域描述 $fexp_i$ 为该条路径上的功能约束, 所有 $expr op 0$ 组成的运算表达式为路径约束, 将它们组成和取范式存入 $FPTerm$ 数组, 就得到一条功能路径约束条件.

约束通常指一个包含输入变量且符合特定条件的逻辑或算术表达式. 对于函数, 表示输入变量的取值范围; 对于表达式, 则表示对输入变量的限制条件. 约束集是指定功能路径上所有约束的集合. 约束系统的建立实质上是对收集到的约束条件实施一系列转换, 最终将约束系统转化为方程组. 求解约束系统则视方程组的类型而定. 若为线性方程组, 则利用文献[6]中提供的线性最小二问题的广义逆法经三

个步骤就可解得测试数据;对于非线性方程组,利用 Monte Carlo 法也只需六步就可获解.若方程组无解,则表明该条功能路径不可达,同时定位了可能的逻辑设计错误.

算法通常把功能点及其所包含的路径作为输入参数,当功能点 f_i 的输入为空时,设 f_i 下有 j 条路径,算法演变为对这 j 条路径生成测试用例.但这些路径是覆盖功能的“大路径”,与众多内部逻辑路径不同,能够覆盖这些“大路径”,也就完成了功能测试.由于嵌入式软件通常是以特定的目标硬件平台为基础、面向固定的任务进行设计的,因此功能是完全确定的,功能模块之间的逻辑结构相对独立,无函数的交叉调用,模块及其包含的基本路径规模都不太大,这一特性决定了可以以较小的代价收集功能路径约束条件,建立和求解约束系统,生成功能路径测试用例.

4 测试实例

4.1 算法实例验证

在一台 PC 机上,为我们自主开发的嵌入式软件智能水量计量仪(基于 MSP430 平台和 IAR Embedded Workbench for MSP430 Ver3. 41A 开发环境)的控制水阀主程序生成功能路径测试用例.控制水阀主程序部分代码及静态分析后的编号如下:

```
void main(void)
{
    001 WDTCTL=WDTPW+WDTHOLD;
    002 Symtem_Init();
    003 LED_Reset();
    004 BurstWriteTime(Time_Value);
    005 Parameter_Flag=1;
    006 while(1)
    {
        007 DINT();
        008 Compute_Water();//计算水量
        009 ReadSeq(0x0030,&Surplus_water[0],12);//读余水量
        .....
        010 PageWrite(0x0030,surplus,12);
        011 U=(X-Y)*2;
        012 if(X>Y)
        013 W=U
        014 else
            W=Y;
        015 if((W+Z)>10)
            {
                016 X=X-2;
                017 Y=Y+W;
                018 Cloze_Alarm();
            }
    }
}
```

```
019 Open_Valve();
020 else if(X*X+Z*Z)>10)
    {
        021 Y=X*Z+1;
        022 Cloze_Alarm();
        023 Open_Valve();
    }
024 if(U>0)
    {
        025 Open_Alarm();
        026 Open_Valve();
    }
027 else if(U==0)
    {
        028 Open_Alarm();
        029 Close_Valve();
    }
}
```

该段程序若按完全路径测试方法,有 11 条路径,需要 11 条测试用例才能满足路径覆盖.但若按照功能路径测试方法,只需要 4 条功能路径测试用例就能达到测试要求.图 3 是用自主开发的原型工具对上述代码进行功能路径分析的结果,左面板为源代码,右面板上方为分析得到的功能路径,下方为函数调用列表.根据功能路径分析方法,function_test 数组中只包含 4 个功能约束条件:

fexp₁:简易计算水量大于报警值, $W + Z > 10$;
fexp₂:精确计算水量大于报警值, $X * X + Z * Z > 10$;
fexp₃:水量大于 0, $U > 0$; fexp₄:水量为 0, $U = 0$;

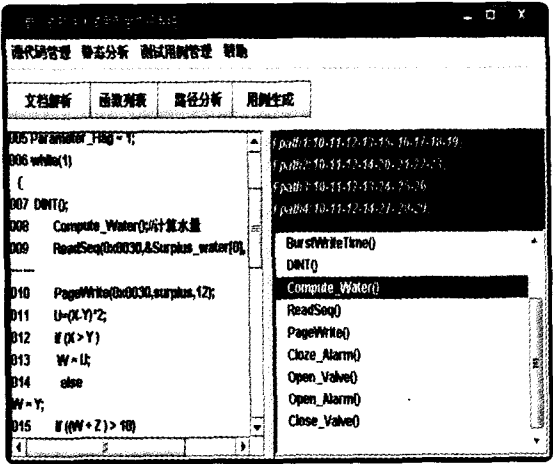


图 3 功能路径分析结果

上述 4 个功能约束条件分别对应以下 4 条功能

路径:

Path1:10-11-12-13-15-16-17-18-19;

Path2:10-11-12-14-20-21-22-23;

Path3:10-11-12-13-24-25-26;

Path4:10-11-12-14-27-28-29.

先为 Path1 生成功能路径测试用例. Path1 上 B_expression 数组中的路径约束均为线性约束条件,将其与功能约束条件结合建立功能路径约束系统: $X-Y>0, 2X-2Y+Z-10>0, 2X-2Y>0$. 将这三个不等式组成的不等式组转化成线性方程组,经线性最小二问题的广义逆法解得: $T_1(X, Y, Z) = (22, 1, 10.80)$, 此即为 Path1 的功能路径测试用例.

Path2 上存在非线性约束式,建立约束系统为: $X-Y\leq 0, X * X + Z * Z - 10 > 0, X * Z - Y + 1 = 0$. 利用 Monte Carlo 法解得 Path2 上的功能路径测试用例: $T_2(X, Y, Z) = (87.25, 87.75, -0.134)$. 同理,可解得 Path3 和 Path4 上的功能路径测试用例分别为: $T_3(X, Y, Z) = (3.725, 3.225, 46.156)$, $T_4(X, Y, Z) = (46.125, 46.375, 86.667)$.

4.2 实测结果分析

通过自主开发的插装工具对上述被测源代码插装后,运行测试用例进行目标机测试,读取从目标机返回的测试结果:程序涉及的 4 个功能子域全部覆盖,说明程序能够完成控制水阀的全部功能,同时测试用例记录的实际经过路径能够完全覆盖功能路径分析得到的预期关键路径结点,说明程序内部逻辑结构能够很好地支持功能的实现.

我们对比两种方法为智能水量计量仪的 24C64.c、alarm.c、IC_Card.c、1302.c、calculate.c、key.c、LED.c、mobile.c、sample.c、zlg7289.c 等 10 个功能模块进行目标机测试,运行测试用例并分析测试结果如表 1 所示.

表 1 测试效果对比

测试用例类型	路径数目	功能数目	功能覆盖率/%	用例个数	检错数/执行个数
完全路径测试用例	109	23	100	109	2/53
功能路径测试用例	23	23	100	23	2/5

上述模块共包含 23 个功能,功能路径分析法只关心和功能相关的大路径,因此路径和用例数均为 23 个;而完全路径分析则深入程序内部的具体逻辑

细节,因此路径数目较多,经过手动分析,不含循环结构共划分出 109 条路径,为此要设计 109 个测试用例.为了覆盖所有 23 个功能,前者需要执行 109 个测试用例而后者仅需 23 个,用例数量减少了近 80%. calculate.c 模块中有两处编码错误,前者执行了 53 次才发现,而后者执行 5 次就从 inter-point 字段中定位了用例执行中断在源代码中的位置.实践表明后者测试用例的生成效率和稳定性都比较高,生成的成本比较低,而测试效果良好.

5 结束语

文中提出了一种为嵌入式 C 语言程序静态生成测试用例的通用方法,从实测效果看,测试用例检错效率高、代价低、通用性强,能够从外部功能和内部结构两个方面对嵌入式软件进行全面测试,保证了嵌入式软件的质量和稳定性.下一步工作将研究测试时间性能和内存使用的测试用例,并对嵌入式操作系统^[7]的测试及测试路径的生成方法^[8]进行更深入地研究.

参考文献:

- [1] 马可,陈蜀宇,石振明.一种嵌入式软件的类状态测试框架设计[J].计算机系统应用,2008(1):23-26.
- [2] 陈小光,陈蔚薇,郭丽丽.嵌入式软件运行剖面建模及测试用例生成[J].嵌入式软件应用,2008(4):79-81.
- [3] 金虎,李志蜀,张磊,等.基于面向路径的遗传算法的测试用例自动生成[J].计算机工程,2007(2):21-23.
- [4] 万琳.路径测试用例自动生成中的搜索算法分析[J].计算机工程,2006(1):74-77.
- [5] Edvardsson J, Kamkar M. Analysis of the constraint solver in UNA based test data generation[J]. ACM SIGSOFT Software Engineering Notes, 2001,26(5):237-250.
- [6] 何渝.计算机常用数值算法与程序[M].北京:人民邮电出版社,2003.
- [7] 王红玲,吕强,褚亚铭.一个微内核操作系统的设计与实现[J].微电子学与计算机,2008,25(4):10-14.
- [8] 赵磊,伦立军,徐士华.基于软件体系结构的测试路径生成方法[J].微电子学与计算机,2008,25(1):177-180.

作者简介:

胡燕京 男,(1980-),硕士研究生,讲师.研究方向为嵌入式软件开发与测试、信息安全.

张毅坤 男,(1958-),博士,教授.研究方向为软件工程、软件测试.

朱伟 男,(1980-),硕士研究生.

于全喜 男,(1980-),硕士研究生.