doi:10.3969/j.issn.1001-2400.2012.06.012

程序流程图到代码的自动生成算法

王黎明，王帼钕，周明媛，褚艳利，陈 科，陈 平
(西安电子科技大学 软件工程研究所，陕西 西安 710071)

摘要：提出了一种从标准程序流程图到结构化代码生成的新算法. 该算法通过对程序流程图结构的分析与识别、循环结构的线性化以及对分支结构域的确定等过程，能够生成符合程序流程图语义的结构化 C 代码(包括 continue/break/return)；同时，能够识别出非结构化的程序流程图.

关键词：可视化编程；程序流程图；模型驱动；代码生成
中图分类号：TP311.5 **文献标识码**：A **文章编号**：1001-2400(2012)06-0070-08

Research on and implementation of the algorithm from the program flowchart to the code

WANG Liming, *WANG Guonü*, *ZHOU Mingyuan*,
CHU Yanli, *CHEN Ke*, *CHEN Ping*
(Research Inst. of Software Engineering, Xidian Univ., Xi'an 710071, China)

Abstract: A new method for generating the structural code from the standard program flowcharts is presented. By analyzing the flowchart, linearizing the cycle structure and delimiting the branch structure, the structual C code agreeing with the semanteme of the flowchart is generated. At the same time, the unstructural flowchart is recognized.

Key Words: visual programming; flowcharting; model-based; code generation

嵌入式领域中模型驱动开发技术日益成熟并广泛应用，国内外很多学者都对模型驱动技术进行了大量研究[1-6]，其中有很多是对程序流程图到代码的自动生成技术的研究[7-12]. 然而现有的程序流程图到代码的自动生成工具[7-9]对流程图有着各种限制，并不是基于标准的程序流程图，而是提供了结构化程序流程图中的多种基本控制结构组件，如不同类型的循环结构、分支结构等. 另外，这些工具[10-12]大多是针对不含 cotinue/break/return 语义的结构化的程序流程图，而针对含有 continue/break/return 语义的程序流程图的研究则较少. 而笔者在文中实现的程序流程图到代码的自动生成技术是针对标准的程序流程图设计的，并且对于含有 continue/break/return 语义的程序流程图也能够很好地识别并生成相应的代码.

1 算法中的定义

(1) 结构化程序设计：①这种设计是一种进行程序设计的原则和方法，它采用自顶向下逐步求精的方法和单入口单出口的控制结构，按照这种原则和方法设计出的程序具有结构清晰、容易阅读、容易修改、容易验证等特点. ②这种设计讨论了如何将任何大规模和复杂的流程图转换成一种标准形式，使用程序设计语言中的顺序、分支、循环等有限的控制结构，通过组合或嵌套来表示程序的控制逻辑. ③这种设计是避免使用

收稿日期：2011-07-26 网络出版时间：2012-09-12
基金项目：国家自然科学基金资助项目(60903198)
作者简介：王黎明(1982－)，男，讲师，西安电子科技大学博士研究生，E-mail：wanglm@mail.xidian.edu.cn.
网络出版地址：http://www.cnki.net/kcms/detail/61.1076.TN.20120912.1012.201206.83_012.html

GOTO 语句的一种程序设计.

(2) 结构化程序流程图:这种流程图是由顺序、分支和循环 3 种基本控制结构通过各种复杂的组合、嵌套关系构成的,并且各种基本控制结构是单入单出的,不会出现带有 GOTO 含义的元素的程序流程图.

(3) 半结构化程序流程图:这种流程图在结构化流程图的基础上引入了受限制的半结构化元素,如带有 continue/break/return 语义的控制流线.这些半结构化的元素在规范的限制条件下,扩展了基本控制结构单入单出的特性,既保留了结构化的清晰性和安全性,又不失灵活性.

2　算法描述

流程图代码自动生成算法包括:(1)程序流程图的结构分析与识别.(2)循环结构的线性化.(3)分支结构域的确定.(4)结构化 C 代码的生成.其中,在前 3 个阶段中嵌入了对程序流程图结构化的检测,整个过程首先根据一定的准则,识别出流程图确定的结构,对于无法初步识别的特殊结构采用回溯算法:初步假设一种结构,然后进行结构化验证,如果符合结构化准则,则表示识别正确;否则,进行回溯.

2.1　程序流程图的结构分析与识别

2.1.1　循环与分支结构的识别

标准的程序流程图中矩形框均为顺序结构不需要识别,只需要区分流程图中的菱形框所构成的是循环还是分支结构.从图论角度分析,流程图是有向图,其中的每个外层循环结构形成了有向图中的极大强连通子图,多个外层循环结构组合在一起形成非强连通有向图中的强连通分量(所有极大强连通子图).根据这个原则找出程序流程图中的所有最大强连通子图,就可以识别出流程图中最外层所有的循环结构.当识别出最外层的循环结构后,需要将构成循环的回边消掉,这样就可以继续判断该循环内层是否还存在嵌套的循环结构.找出所有的循环结构后,其余的菱形结点引领的结构即为分支结构.具体算法见算法 1.

2.1.2　半结构化程序图元素的识别

对于 continue/break/return 这种半结构化元素,则根据程序流程图的控制流流向信息来识别.标识 continue 的流程线把控制流引向了循环的判断结点,标识 break 的流程线把控制流引向了循环结束后的第 1 个结点,标识 return 的流程线把控制流引向了结束结点.在识别出半结构化的流程线之后,把流程线指向虚拟的标识此元素的结点,后继悬空,留待第 3 阶段处理.具体算法见算法 1.

算法 1　流程图结构的识别.

输入:最外层强连通子图即整幅程序流程图.

输出:true 为符合结构化准则,完成结构识别;false 为流程图非结构化.

方法:使用递归的方法,从最外层最大强连通子图开始判断流程图结构,并消去构成循环的回边,再递归地判断里层的结构.

```
bool RecognizeStructure(FlowChartSCG scg)
{
    计算当前层的最大强连通子图集合
    对于每个最大强连通子图进行如下操作
    {
        if (构成最大强连通子图的结点数为 1)
        {
            识别分支结构或顺序结构
            跳出此次循环
        }
        判断构成最大强连通子图的循环类型
        if (判断过程中出现异常)
            return 非结构化结论
```

```
        消去构成环路的流程线
        递归对该最大强连通子图 调用 RecognizeStructure 函数进行结构识别
        if (识别过程出现异常)
            return 非结构化结论
        识别最大强连通子图内部存在的半结构化元素
        if(识别过程出现异常)
            return 非结构化结论
        对最大强连通子图及其内部进行结构化验证
        if (验证过程出现异常)
            return 非结构化结论
    }
    return 结构化结论
},
```

2.2 循环结构的线性化

结构化程序流程图中,循环结构与分支结构相互嵌套,组合构成整个图. 为了生成代码及确定分支结构域更加方便,借鉴了标准流程图中循环界限的标识,将构成循环结构强连通子图中的环路去掉,并在循环的入口结点及出口结点处加入了循环上下界结点对整个循环的范围进行限定,这样就把循环结构变为了顺序结构.

2.3 分支结构域的确定

当把循环结构线性化之后,流程图中只存在顺序结构、分支结构与半结构化的无后继结点. 给每个结点增加一个标志变量 JointCount,初始值为 0,每当识别出一个汇点,就把该汇点的 JointCount 自加 1,根据表达式: 结点的入度−1=以该结点为汇点的分支结构的数量,来识别分支结构的作用域. 同时为第 1 阶段中后继悬空的半结构化元素确定后继结点. 该算法采用循环递归方式,从开始结点开始进行广度优先搜索,当搜索过程中遇到循环上界结点时,则调用处理循环结构的函数,同时该函数返回到循环下界结点,以便上层从循环下界结点开始继续搜索; 当搜索过程中遇到分支结构的菱形结点时,则调用处理分支结构的函数. 此时对于每一个搜索到的结点,先根据上述等式判断该结点是否为分支的汇点,如果是,则返回该汇点;否则,继续进行搜索,直到找到汇点,并返回. 当搜索过程中遇到矩形结点时,则继续搜索矩形结点的下一个结点. 具体算法如算法 2,算法 3,算法 4 所示.

算法 2 分支结构域的确定.

输入: 流程图开始、结束结点,或循环上、下界结点.

输出: true 为符合结构化准则; false 为流程图非结构化.

```
bool HandleLoop(FlowchartNode startNode, FlowChartNode endNode)
{
    从开始结点开始广度优先遍历,对于每一个结点进行如下操作
        if (当前结点为开始结点或矩形框结点)
            当前结点=当前结点的后继结点
        else if (当前结点为循环上界结点){
            递归调用 HandleLoop 函数处理此循环内部结构
            if (识别过程出现异常)
                return 非结构化结论
            当前结点=与当前结点对应的循环下界结点
        }
        else if (当前结点为 if 结点){
            调用 RecognizeIFEnd 函数处理此分支内部结构
```

```
            当前结点=RecognizeIFEnd 函数返回的汇点
        }
        else if (当前结点为 CBR 结点)
            确定半结构化元素的后继结点
        else return 非结构化结论
}.
```

算法 3　分支结构汇点的确定.

输入:引领分支结构的判断结点.

输出:分支结构的汇合点,NULL 表示非结构化.

方法:

```
FlowChartNode RecognizeIFEnd(FlowChartNode ifStartNode)
{
    调用 RecognizeIFPath 函数,获得 T 分支的汇点
    if (返回的汇点为 CBR 结点){
        半结构化结点连向 F 分支首结点,即把 F 分支首结点作为该 if 的汇点
        F 分支首结点的 JointCount 自加 1
        return F 分支首结点
    }
    调用 RecognizeIFPath 函数,获得 F 分支的汇点
    if (返回的汇点为 CBR 结点){
        交换 if 的 TF 两个分支
        半结构化结点连向 F 分支首结点,即把 F 分支首结点作为该 if 的汇点
        F 分支首结点的 JointCount 自加 1
        return F 分支首结点
    }
    if (T 分支返回的汇点=F 分支返回的汇点 并且该汇点不为 NULL) {
        该汇点 JointCount 自加 1,即为分支的汇点
        return 汇点
    }
    else
        return NULL
}.
```

算法 4　分支结构单分支汇合点的确定.

输入:分支结构单分支起始结点.

输出:分支汇合点或者半结构化结点,NULL 表示未找到汇点.

```
FlowChartNode RecognizeIFPath(FlowChartNode startNode)
{
    bool bFoundJoint=false;//标记是否找到汇合点
    从开始结点开始广度优先遍历
    while(未找到汇合点) {
        if (当前结点的入度-当前结点的 JointCount>=2) 找到汇合点
        else if (当前结点为矩形结点) 当前结点=当前结点的后继结点
        else if (当前结点为循环上界结点){
            调用 HandleLoop 函数处理循环内部结构
```

```
            当前结点=与当前结点对应的循环下界结点的后继结点
        }
        else if (当前结点为 if 结点){//当前结点引领了分支结构
            调用 RecognizeIFEnd 函数处理此分支内部结构
            当前结点=RecognizeIFEnd 函数返回的汇点
            if (当前结点为 NULL) 找到汇合点
        }
        else if (当前结点为 CBR 结点) 找到汇合点
    }
    return 当前结点
}.
```

2.4 结构化 C 代码的生成

进行了前面 3 步的处理之后,就可以以线性的方式从流程图的开始结点进行遍历,读取结构信息以及结点中携带的代码信息,生成每个结构对应的代码.

3 算法复杂度分析

设流程图中结点总数为 N,流程线总数为 M,循环结构的总数为 P.

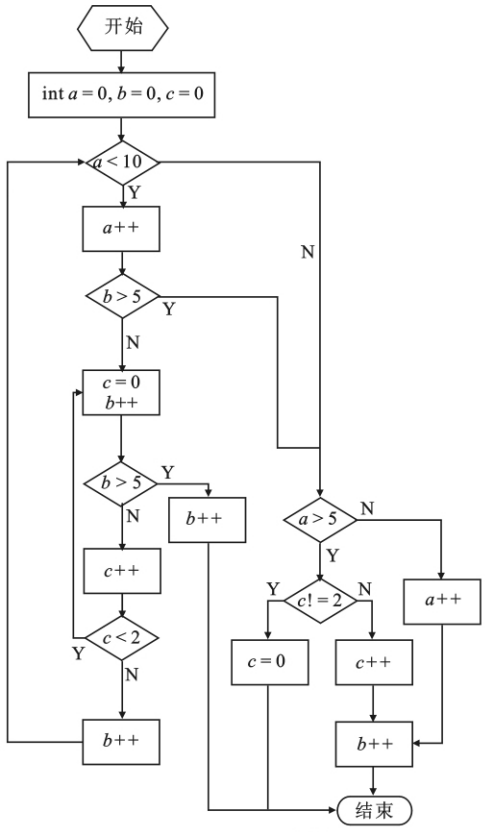
算法的主要开销集中于第 1 阶段中对流程图的遍历和递归迭代上. 每一层迭代处理一个循环结构,遍历此循环结构中的所有的结点和流程线. 整个流程图作为最外层的一个强连通图,因此算法共处理了 $(P+1)$ 个循环结构. 最坏的情况为每次迭代需要遍历的循环结构均由整幅图的所有结点和流程线组成,因此最坏的情况下算法的时间复杂度为 $O((P+1)(N+M))$.

4 实验结果

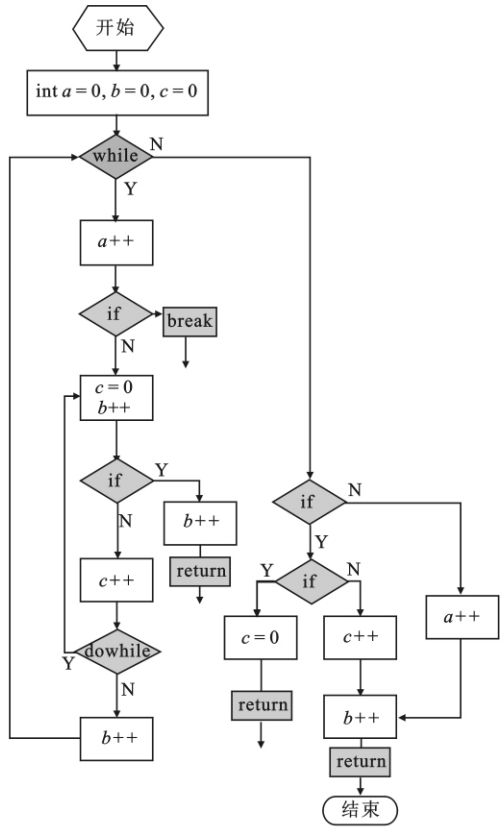
以图 1(a) 所示的程序流程图为例,具体介绍按照文中算法 4 个阶段生成其相应的 C 代码的过程. 第 1 阶段,识别程序流程图中结点的结构,通过强连通子图分析,可以得到最外层的“while”结构及内层的“dowhile”结构. 同时,能够识别出程序流程图中表示“break”和“return”的语义线. 具体结果见图 1(b). 第 2 阶段,把识别出来的循环结构(包括“while”和“dowhile”结构)转换为以上界和下界标识的顺序结构,并将表示“break”和“return”的语义线转换为相应的结点,如图 1(c) 所示. 第 3 阶段,确定每个分支结构(包括“if”结构)的范围,在此过程中,将上一阶段中获得的表示“break”和“return”的结点连接到相应的结构中,具体结果如图 1(d)所示. 第 4 个阶段,顺序遍历各个结点,获得结构化的 C 代码. 该代码中包含了“while”和“dowhile”的循环语句,也包含了“if”的分支语句,还包含了“break”和“return”的语句.

生成的 C 语言代码为:

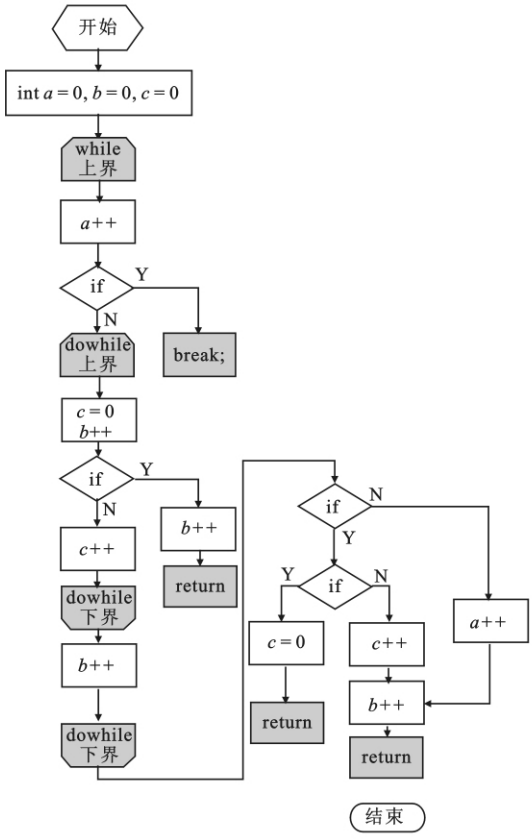
```
void function(){
    int a=0,b=0,c=0;
    while( a<10){
        a++;
        if( b>5){
            break;
        }else{
        }
        do{
            c=0;
```



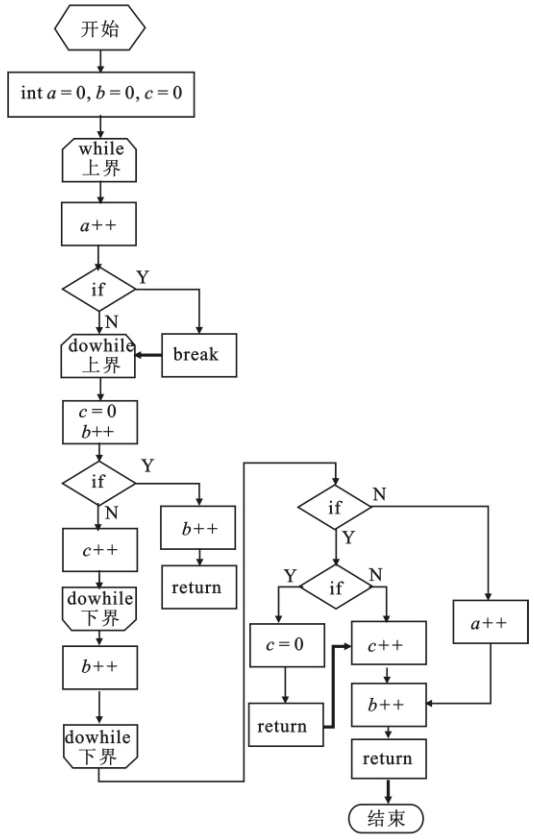
(a) 原图



(b) 第 1 阶段结果图



(c) 第 2 阶段结果图



(d) 第 3 阶段结果图

图 1　程序流程图实例

```
            b ++;
            if( b > 5){
                b ++;
                return;
            }else{
            }
            c ++;
        }while( c < 2);
        b ++;
    }
    if( a > 5){
        if( c ! =2){
            c =0;
            return;
        }else{
        }
        c ++;
    }else{
        a ++;
    }
    b ++;
    return;
}.
```

5 结 束 语

研究了程序流程图到代码的自动生成算法,该算法不对流程图加入任何限制,针对标准的程序流程图,能够识别出含有 continue/break/return 语义的流程线;对于结构化的程序流程图能够生成结构化的 C 代码;对于非结构化的程序流程图,能够识别出来,并给予说明.

参考文献:

[1] Ingo S, Mirko C, Ines F, et al. Experiences with Model and Autocode Reviews in Model-based Software Development [C]//Proc of Third International ICSE Workshop on Software Engineering for Automotive Systems (SEAS'06). Shanghai: ACM, 2006: 45-51.

[2] Hamid B, Kevin S. Monarch: Model-based Development of Software Architectures[C]//Proc of the 13th International Conference on Model Driven Engineering Languages and Systems: Part Ⅱ (MODELS'10). Antwerp: Springer, 2010: 376-390.

[3] Satya V, Johan C P. Automating UI Guidelines Verification by Leveraging Pattern Based UI and Model Based Development[C]//Proc of the 28th of the International Conference Extended Abstracts on Human Factors in Computing Systems(CHI EA'10). Atlanta: ACM, 2010: 4733-4742.

[4] Brown A W, Lyengar S, Johnston S. A Rational Approach to Model-Driven Development[J]. IBM Systems Journal, 2006, 44(4): 463-480.

[5] Wang L M, Wang G N, Wang W, et al. MBD-DSP: a Model Based Design Solution for DSP [C]//Proc of 2011 International Conference on Electrical and Control Engineering(ICECE 2011). Yichang: IEEE, 2011: 4561-4564.

[6] 李青山, 陈平, 褚华. 支持柔性机制的元数据驱动模型的研究与应用[J]. 西安电子科技大学学报, 2002, 29(3): 319-

324.
Li Qingshan, Chen Ping, Chu Hua. Research on and the Application of the Metadata-driven Model Supporting Flexibility [J]. Journal of Xidian University, 2002, 29(3): 319-324.
[7] Dy J B, Laureano P, Liu M D, et al. MDDT: Model Driven Development Tool as Aid for Learning Programming Fundamentals[R]. Davao: CSP, 2010.
[8] Martin C C, Terry A W, Jeffrey W H, et al. RAPTOR: Introducing Programming to Non-Majors with Flowcharts[J]. Journal of Computing Sciences in Colleges, 2004,19(4): 52-60.
[9] Tia W. The SFC Editor: a Graphical Tool for Algorithm Development[J]. Journal of Computing Sciences in Colleges, 2004, 4(1): 73-85.
[10] Kanis C, Somkiat W. Visual Programming Using Flowchart [C]//Communications and Information Technologies. Bangkok: ISCIT'06, 2006: 1062-1065.
[11] Drazen L, Ivan F. A Visual Programming Language for Drawing and Executing Flowcharts [C]//Proc of the 34th International Convention. Opatija: IEEE, 2011: 1679-1684.
[12] **钟志超，张志胜，戴敏，等. 流程图与类 C 语言实时转换方法**[J]. **东南大学学报**, 2009, 39(3): 502-506.
Zhong Zhichao, Zhang Zhisheng, Dai Min, et al. Real-time Conversion Method for Flowchart and Similar C Language [J]. Journal of Southeast Uinversity, 2009, 39(3): 502-506.

（编辑：齐淑娟）

（上接第 21 页）

[5] Al-Jamimi H A, Mahmoud S A. Arabic Character Recognition Using Gabor Filters [J/OL]. [2010-03-10]. http://china.springerlink.com/content/w6040hx428389w68/fulltext.pdf.
[6] Al-Emami S, Usher M. On-Line Recognition of Handwritten Arabic Characters [J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 1990, 12(7): 704-710.
[7] Harouni M, Mohamad D, Rasouli A. Deductive Method for Recognition of On-Line Handwritten Persian/Arabic Characters [C]//Proc of the 2nd International Conference on Computer and Automation Engineering (ICCAE). Singapore: IEEE, 2010: 791-795.
[8] Sternby J, Morwing J, Andersson J, et al. On-line Arabic Handwriting Recognition with Templates [J]. Pattern Recognition, 2009, 42(12): 3278-3286.
[9] **韩崇昭，朱洪艳，段战胜. 多源信息融合** [M]. 2 **版. 北京：清华大学出版社**, 2010: 82-105.
[10] Hady M F A, Schwenker F, Palm G. Semi-supervised Learning for Tree-structured Ensembles of RBF Networks with Co-Training [J]. Neural Networks, 2010, 23(4): 497-509.
[11] **苗壮，程咏梅，潘泉，等. 证据推理的近似计算研究** [J]. **西安电子科技大学学报**, 2011, 38(2): 187-193.
Miao Zhuang, Cheng Yongmei, Pan Quan, et al. Research on the Approximation Algorithm of the Evidential Theory [J]. Journal of Xidian University, 2011, 38(2): 187-193.
[12] ISO/IEC JTC 1. ISO/IEC 8859-6-1999, Information Technology-8-Bit Single-byte Coded Graphic Character Sets-Part 6: Latin/Arabic Alphabet [S]. London: BSI, 1999.
[13] Verma B, Blumenstein M, Ghosh M. A Novel Approach for Structural Feature Extraction: Contour vs. Direction [J]. Pattern Recognition Letters, 2004, 25(9): 975-988.
[14] Jin Lianwen, Wei Gang. Handwritten Chinese Character Recognition with Directional Decomposition Cellular Features [J]. Circuits, Systems and Computers, 1998, 8(4): 517-524.
[15] Kimura F, Takashina K, Tsuruoka S, et al. Modified Quadratic Discriminant Functions and Its Application to Chinese Character Recognition [J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 1987, 9(1): 149-153.

（编辑：高西全）