ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.3, March 2006, pp.628−637
DOI: 10.1360/jos170628

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

包含依赖输入分支程序的符号化 WCET 分析*

姬孟洛+, 齐治昌, 王怀民

(国防科学技术大学 计算机学院,湖南 长沙 410073)

Symbolic WCET Analysis of Programs Containing Input-Dependent Branches

JI Meng-Luo+, QI Zhi-Chang, WANG Huai-Min

(School of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: Phn: +86-731-4573650, E-mail: jimengluo@yahoo.com.cn

Ji ML, Qi ZC, Wang HM. Symbolic WCET analysis of programs containing input-dependent branches. *Journal of Software*, 2006,17(3):628−637. http://www.jos.org.cn/1000-9825/17/628.htm

Abstract: Symbolic WCET (worst-case execution time) analysis yields symbolic upper bound expressions for tasks that contain parameters. Quickly evaluated at run-time, such expressions can improve the accuracy of WCET estimate. This paper proposes a symbolic WCET analysis method that is especially for the input-data dependent branches. First, the formulas described by Blieberger is expanded, so that they can express the input-data dependent branches. Simplification of the formulas using control-dependent graph yields conditional symbolic expressions that have different forms corresponding to different input value ranges. Different from the existing methods, the symbolic formulas are directly dependent on input-data, so WCET estimate evaluation at run-time is more simple and straightforward.

Key words: parametric worst-case execution time (WCET) analysis; WCET analysis; program analysis; real-time system; software engineering

摘 要: 符号化 WCET(worst-case execution time)分析是用符号表达式表示任务的最大执行时间:表达式中包含了参数.通过在运行时刻快速确定表达式值,符号化 WCET 分析可以更精确地估算 WCET.提出了一种针对其分支直接依赖于输入数据的程序的符号化 WCET 分析方法.首先对 Blieberger 方法进行扩充,使得 WCET 符号表达式能够表达依赖输入分支,然后利用程序的控制依赖图对符号表达式进行化简,从而产生带条件的 WCET 符号表达式,即不同的条件对应不同的符号表达式.与已有方法不同,符号化 WCET 公式直接依赖于输入参数,使得运行时的 WCET 估算更加简单直接.

关键词: 参数化最大执行时间 WCET 分析;WCET 分析;程序分析;实时系统;软件工程
中图法分类号: TP311 文献标识码: A

事先获取系统中每个任务的最大执行时间(worst-case execution time,简称 WCET)对实时系统的时间分析具有非常重要的意义.事实上,获知系统中任务的 WCET 是任务调度及可调度性检测的前提,是系统设计中软硬件界限划分的重要依据,也是系统设计阶段处理器特性选择的基础.

* Supported by the National Natural Science Foundation of China under Grant No.60303013 (国家自然科学基金)
Received 2004-07-01; Accepted 2005-08-29

获取 WCET 的主要方法是静态分析,即静态确定每个任务的最大执行时间.传统的 WCET 分析方法的结果是一个数值,这样不仅不够精确,而且限制了应用[1].符号化 WCET 分析获得 WCET 的符号化表示:其结果表达式是包含了参数的公式.这些参数或者是程序输入参数[2−4],或者是循环不变量[4],或者是循环的最大迭代次数[1].这样的公式可以在运行时刻快速确定公式值,以便在调度一个任务或确定一个任务中的算法时,做出动态调度决策[1].

针对包含直接依赖输入数据(简称依赖输入(input-dependent))分支的程序,本文提出一种符号化 WCET 分析方法.首先,对 Blieberger 方法[2]进行扩充,使得 WCET 符号表达式能够表达依赖输入分支节点;然后,利用控制依赖图生成扩展输入控制依赖图,再利用此图对包含依赖输入分支程序的符号化 WCET 表达式进行化简,从而产生带条件的 WCET 符号表达式,即不同的条件对应不同的符号表达式.与文献[4]相比,我们的方法能够自动确定模式以及模式和输入数据的对应关系.与以前的其他符号化分析方法[1−4]不同,我们的符号化 WCET 公式直接依赖于输入参数,这样,在运行时,WCET 估算更加简单直接.

本文第 1 节主要描述与程序的控制流图(CFG)相关的概念.第 2 节介绍基于分支执行频率的 WCET 计算公式以及对包含依赖输入分支进行的扩充.第 3 节描述使用控制依赖图对包含依赖输入分支程序的 WCET 符号表达式的化简方法,化简包括获得简化 WCET 符号表达式以及获得相应输入条件的方法.第 4 节以具体例子说明如何对一个程序框架计算其带条件的 WCET 符号表达式.第 5 节将本文方法与其他符号化 WCET 方法进行比较,并对全文进行总结.

1 基本概念

任务(task)是一个程序,每个程序都有唯一的控制流图 CFG(control-flow graph)[5].一个程序的 CFG 是一个有向流图 $G=(N,E,entry,exit)$.其中,N 是节点集合;E 是边集合;$entry$ 是唯一的入口节点;$exit$ 是唯一的出口节点.

CFG 中一个节点 $n\in N$ 表示一组或者一条语句,也就是程序的一个基本块[5].边 $(m,n)\in E$ 表示两个节点 $m,n\in N$ 之间的控制转移,其中 m 是 n 的前驱节点,n 是 m 的后继节点.入口节点 $entry$ 没有前驱节点.出口节点 $exit$ 没有后继节点.$Preds(n)$是节点 n 的前驱节点集合,即 $\forall m\in Preds(n)$,都有 $(m,n)\in E$.$Succs(n)$是节点 n 的后继节点集合,即 $\forall m\in Succs(n)$,都有 $(n,m)\in E$.后继节点多于一个的节点称为分支节点.对于分支节点 m,如果 n 是其后继节点,则边 (m,n)称为节点 m 的一个分支.G 中的一条路径是一个节点序列 $[n_1,n_2,\ldots,n_k]$,且对 $1\le i\le k-1$,都有 $n_{i+1}\in succs(n_i)$.这里,假定所有的节点都位于从 $entry$ 到 $exit$ 的一条路径之中.

与文献[2]类似,这里假定每个基本块的 WCET 是固定的,并可以在编译时确定.也就是说,我们针对的是指令执行时间不变的处理器,或者是指令执行时间依赖于操作数的处理器(例如某些微指令处理器,虽然其乘法和除法指令的执行时间会因操作数的不同而发生变化,但其 WCET 是固定的).因此,G 中每一个节点对应一个 WCET,此执行时间为常量.为简化表示,用 τ_n 表示节点 n 的 WCET,用 $\tau_{n1,\ldots,nk}$ 表示 k 个节点 $n_1,\ldots,n_k$ 的 WCET 之和.

定义 **1**. 对分支节点 m,$(m,n)\in E$,$Cond(m,n)$表示从节点 m 迁移到节点 n 的条件,当 $Cond(m,n)$=TRUE 时,m 执行分支(m,n).对于无条件迁移,$Cond(m,n)$总为 TRUE.

举例来说,当节点 n 位于 if-then-else 语句的 then 分支时,当该语句的判断表达式(谓词)为 TRUE 时,$Cond(m,n)$=TRUE;否则,$Cond(m,n)$=FALSE.而当节点 n 位于该语句的 else 分支时,在该谓词为 FALSE 时,$Cond(m,n)$=TRUE;否则,$Cond(m,n)$=FALSE.这里,m 是指 if 节点.

为了简化分析,假定分支节点只有两个分支,即分支语句的判断表达式为 TRUE 或者 FALSE 时执行的两个分支.对于多个分支的情况,可以类似处理.

定义 **2**. 一个分支节点 m 称为依赖输入节点,如果其对执行分支的选择由输入参数值确定.依赖输入分支节点的分支称为依赖输入分支.

在 CFG 中,节点 m 支配(dominate)另一个节点 n,当且仅当从入口节点 $entry$ 到 n 的所有路径都经过 m.节点 m 后支配(post-dominate)另一个节点 n,当且仅当从节点 n 到出口节点 $exit$ 的所有路径都经过 m.如果 $m\ne n$,则 m 严格(后)支配 n.

2 符号化 WCET 分析方法

2.1 基于分支执行频率的WCET计算方法

本节简要介绍通过静态分析获得符号化分支执行频率以及基于分支执行频率的 WCET 计算方法[2].

预测程序某个分支的执行频率通常是为编译期间的优化提供有用的信息,包括全局指令调度、代码分层、函数嵌入(inline)、过程间的寄存器分配以及许多高层优化.获取分支的执行频率一般有启发式、实际执行采样和静态分析 3 种方法.其中静态分析方法就是通过对源程序分析得到期望分支的执行频率[6,7].

定义 **3**. CFG 的一条边(m,n)的执行频率 $p(m,n)$是一个实数值,它满足:

$$0\le p(m,n)\le 1 \tag{1-1}$$

$$\sum_{j\in Succs(m)} p(m,j)=1 \tag{1-2}$$

其中,$p(m,n)=0$ 表示死路径(dead path),即在任何输入下都不可能执行的路径.

分支的执行频率通常用数值表示,分支执行频率的符号化表示就是用代数表达式表示分支的执行频率.在已知程序条件递归关系的前提下,Blieberger 通过符号化插装,能够分析得到基本块执行频率的符号化公式.

对每个节点 i 定义一个符号化整数值变量 b_i.b_i 的初始值为 0,每进入一次节点 i,变量 b_i 值增加 1.计算出的 b_i 的符号值称为节点 i 的执行计数.b_i 称为插装变量(instrumentation variable).

如果能够得到所有插装变量的符号表达式,那么可以建立公式:

$$b_i=\sum_{j\in Succs(m)} c_{ij} \tag{2-1}$$

$$b_i=\sum_{k\in Preds(i)} c_{ki} \tag{2-2}$$

这里,c_{ij} 是赋给边(i,j)的符号计数,$Preds(i)$和 $Succs(i)$分别是节点 i 的前驱和后继节点集合.

根据公式(2-1)、公式(2-2),通过代入消减能够得到 c_{ij} 的变量表达式,然后通过回代,可以求得对应各个边的符号执行计数.得到边的符号执行计数之后,根据公式(1-1)、公式(1-2),显然有

$$p(i,j)=c_{ij}\Big/\sum_{k\in Succs(i)} c_{ik} \tag{3}$$

根据每条边的执行频率,Blieberger 给出了节点 m 的母函数 $G_m(z)$的公式.

定义 **4**. 节点 m 的母函数 $G_m(z)$定义如下:

$$\begin{cases} G_m(z)=z^{\tau_m}\sum_{j\in Preds(m)} p(j,m)G_j(z) \\ G_{entry}(z)=z^{\tau_{entry}} \end{cases} \tag{4}$$

根据母函数已有的结论[8]可知,一个任务 $Task$ 的最长执行时间 $WCET_{Task}$ 为

$$WCET_{Task}=\frac{\mathrm{d}}{\mathrm{d}z}G_{exit}(z)\Big|_{z=1} \tag{5}$$

式(5)右边的含义是,对出口节点 $exit$ 的母函数 $G(z)$针对 z 微分,然后令 z=1.

2.2 包含依赖输入分支程序的WCET分析方法

循环控制分支通过简单的嵌套启发式方法[9]或者静态分析方法[1,6,7]能够确定其执行频率,对于非循环控制分支,其执行频率则难以确定.Blieberger 在文献[2]中只是给出了循环控制分支执行频率的计算方法,本节对其扩充,使之能够应用于包含依赖输入分支程序.这里假定每个循环控制分支执行频率已知,它们是数值,或者是符号表达式,并且满足公式(1-1)~(2-2).

通过符号计算[10]等方法,最终能够确定依赖输入分支节点的条件表达式,此表达式是由输入参数确定的符号表达式.

对于依赖输入的分支节点,其选择分支的行为与其所在的循环无关.因此在循环执行过程中,这样的分支要

么执行要么不执行,也就是说,其执行频率要么为 1 要么为 0.因此,包含依赖输入分支程序的 WCET 也由输入参数值确定.

定理 **1**. 假定节点 m 是依赖输入分支节点,其执行计数为 Cnt,m 有 2 个依赖输入分支(m,n_1)和(m,n_2),令

$$p(m,n_i)=\begin{cases}1, & \text{if Cond}(m,n_i)=\text{True}\\ 0, & \text{otherwise}\end{cases},$$

则节点 n_i 的执行计数为 $p(m,n_i)*Cnt$.此处,$p(m,n_i)$为分支(m,n_i)的执行频率,i=1,2.

定理 1 的证明要用到如下定义:

定义 **5**. 令 v 表示变量,则称 $v(k)$为该变量的递归副本(recursive counterpart).这里,k 表示变量 v 位于循环执行中的第 k 次循环.

定理 1 证明:假定对节点 m 的插装变量 b_m 有递归关系[2]:

$$b_m(k+1)=e(k) \text{ if } C_m(k)\text{计值为真},$$

其中 $e(k)$和 $C_m(k)$是指 e 和 C 中包含的所有变量都用它们的递归副本代替.这里,e 是 b_m 的符号表达式;C_m 是条件.

那么显然有

$$b_{n_i}(k+1)=e(k) \quad \text{if } C_m(k)\text{计值为真,且 } Cond(m,n_i)=\text{TRUE},$$

即

$$b_{n_i}(k+1)=p(m,n_i)\times e(k) \quad \text{if } C_m(k)\text{计值为真}.$$

因为 m 节点执行计数为 Cnt,所以 n_i 节点的执行计数为 $p(m,n_i)\times Cnt$.

注意到

$$\begin{cases}0\le p(m,n_i)\le 1\\ \sum_{i=1}^{2} p(m,n_i)\end{cases}, \ i=1,2,$$

可知 $p(m,n_i)$为分支(m,n_i)的执行频率.

当对依赖输入分支指定了执行频率 $p(m,n_i)$之后,按照第 2.1 节的方法仍然可以得到式(5),此时,式(5)为包含依赖输入分支节点执行频率的 WCET 计算公式,其中 $p(m,n_i)$是一个符号,其取值为 1 或 0,分别对应于分支节点 m 谓词的不同取值.为了表述方便,我们对所有的依赖输入分支的执行频率 $p(m,n_i)$按顺序进行编号,并约定:假如 $p(m,n_1)$位于分支节点 m 的 TRUE 分支,且标记为 p_j 时,则位于 FALSE 分支的 $p(m,n_2)$标记为 $1-p_j$.同时,为叙述方便,我们把所有的 p_j 和 $1-p_j$ 统称为各个 p.所以说,式(5)是一个包含各个 p 的符号化公式.

如果运行时使用此公式进行动态决策,则需要根据当时的输入参数值计算各个 p 的值,然后和其他参数值一起代入公式(5).这显然是一个繁琐的过程.同时注意到,p 对应于输入参数值的一个组合.结合这两点,我们希望直接得到针对输入参数的符号表达式.

3 带条件的 WCET 符号表达式产生方法

需要注意的是,直接列出各个 p 的组合,然后计算对应的条件,这样求出的条件有可能是错误的.举例来说,假定有如下 C 程序段:

$$\text{if } (x<1)/*p_1*/S_1 \text{ else if } (x<0)/*p_2*/S_2 \text{ else } S_3.$$

其用各个 p 作标记的 CFG 如图 1 所示.

如果列出各个 p 的组合(即 p_1 和 p_2,p_1 和 $1-p_2$,$1-p_1$ 和 p_2,$1-p_1$ 和 $1-p_2$),然后计算对应条件,则得到的条件列表为 $x<0$,$0\le x<1$,$x\ge 1$.而实际上,该条件列表为 $x<1$,$x\ge 1$.显然,p 的组合应当考虑控制流确定的执行次序.

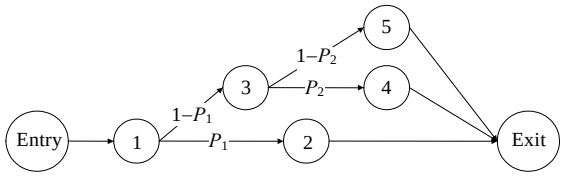

Fig.1 The CFG annotated with p

图1 用p作标记的CFG

本节给出包含直接依赖输入节点的式(5)的条件符号表达式的产生方法.该方法基于控制依赖图,主要包括如下几个步骤:

(1) 根据控制依赖图生成扩展输入控制依赖图(由依赖输入节点组成的控制依赖图称为输入控制依赖图);

(2) 根据扩展输入控制依赖图产生各个 p 的组合;

(3) 计算 p 的组合,求解相应的条件以及该条件下的符号化公式.

3.1 控制依赖图和扩展输入控制依赖图

控制依赖(control dependences)[11]捕获的是程序某些组件的执行(或者执行多少次)是否依赖于某些谓词的值.举例来说,if-then-else 语句的 true 分支只有在条件表达式计值为真时才执行;同样,其 false 分支只有在条件计值为假时才执行.从而,true 分支语句称为真值控制依赖于(*true-control-dependent* on)if 谓词,false 分支语句称为假值控制依赖于(*false -control-dependent* on)if 谓词.

定义 **6**. 程序 CFG 的一个节点 n 是以标记 v 控制依赖于节点 m,当且仅当 n 后支配 m 的 v 后继节点,但没有后支配 m.如果没有其他节点 r,$r\neq m$ 且 $r\neq n$,使得 n 控制依赖于 r,而 r 控制依赖于节点 m,则 n 是 v 直接控制依赖于节点 m(这里,标记 v 为 TRUE 或 FALSE).

由控制依赖关系组成的图称为控制依赖图.控制依赖图是有向图,节点 m 和 n 之间有一条边(m,n),当且仅当 n 以标志 v 直接控制依赖于 m,并且把 v 作为控制依赖边的标记.根据程序 CFG 生成控制依赖关系的算法见文献[11].

因为输入条件只和依赖输入节点有关,因此仅需要产生输入控制依赖图.产生依赖输入分支节点组成的控制依赖图的算法很简单,只要把非依赖输入节点去掉,同时保持原有的控制依赖关系即可.其算法如图 2 所示.

```
foreach vertex v in topological order {
    if v is not a input control vertex {
        foreach edge (m,v) {
            foreach edge(v,n) {
                generate edge (m,n);
                p(m,n)=p(m,v); }      }
        delete each edge (m,v) and (m,n);
        delete vertex v; }   }
```

Fig.2 The algorithm of generating input control dependent graph

图 2 输入控制依赖图生成算法

显然,图 2 所示生成算法的计算复杂度是 $O(N)$,N 是控制流图节点的个数.

在上述算法中,由剩余下来的节点和边组成了输入控制依赖图.因为依赖输入分支节点有可能处于 CFG 控制依赖图的不同分支上,因此一个控制依赖图有可能产生几个独立的输入控制依赖图.

为了后面获取输入条件方便,需要对输入控制依赖图进行扩展.首先,为输入控制依赖图的每一条边更换/

添加标记 p_i,p_i 为该边对应的执行频率,如前所述,它是一个按顺序编号的符号;然后,在每个输入控制依赖图中添加 *exit* 节点,并为每一个叶节点(即没有后继节点的节点)添加两条从叶节点到 *exit* 节点的边,并为这两条边添加对应的执行频率标记 p.

扩展输入控制依赖图的生成算法容易实现,这里不再赘述.

3.2 由输入控制依赖图产生p的组合

对于扩展的输入控制依赖图,可以使用有向图遍历的方法[12]进行遍历,从而获得其中的每一条路径.对于这样的一条路径,可以用序列$[(n_1,n_2,p_1),(n_2,n_3,p_2),\ldots,(n_k,n_{k+1},p_k)]$表示,其中$(n_i,n_{i+1},p_i)$表示标记为 p_i 的边(n_i,n_{i+1}),$i\in[1,\ldots,k]$.这里,(n_i,n_{i+1})是扩展输入控制依赖图中的边.因为标记 p_i 唯一地确定了边(n_i,n_{i+1},p_i),所以,这样的一条边也可以直接用 p_i 表示,而相应的路径则用$[p_1,p_2,\ldots,p_k]$表示,此路径称为标记路径,并称 p_i 为该标记路径的节点.

假定 CFG 有 s 个扩展输入控制依赖图,其中第 i 个图有 l_i 条标记路径,$i\in[1,\ldots,s]$.令 Y_i 为第 i 个扩展输入控制依赖图的标记路径集合,即 $Y_i=\{y_{i1},y_{i2},\ldots,y_{il_i}\}$.这里,$y_{ij}$ 表示第 i 个扩展输入控制依赖图的第 j 条路径.

令 $YSet=Y_1\times Y_2\times\ldots\times Y_s$ 表示 s 个标记路径集合的笛卡尔乘积,$a=(y_{i1},y_{i2},\ldots,y_s)\in Yset$.定义 $Proj(a)(j)=\{Y_{ij}\}$为 $YSet$ 第 j 个分量所对应的标记路径的节点集合,则输入条件集合 *ConditionSet* 为

$$ConditionSet=\left\{\bigcup_{i=1}^{s}proj(a)(i)\mid a\in YSet\right\}.$$

简单地说,*ConditionSet* 表示各个扩展输入控制依赖图中标记路径对应的 p 值集合组成的所有组合.

$\forall a\in ConditionSet$,根据 a 所对应的各个 p 的取值,既可以计值相应的条件,也可以化简公式(5).通过对所有的 $a\in ConditionSet$ 进行计值和化简操作,就可以得到整个 CFG 的条件符号表达式.

注意,一个 a 所对应的条件组合有可能是矛盾的,此时是指 a 对应的路径为不可行路径.

下面定理 2 说明,只有 *ConditionSet* 生成的 p 组合才是可行的.

定理 2. 任给 p 的组合 b,如果 $b\notin ConditionSet$,则 b 是不可行的.即不存在一个输入 x,使得在 x 下能够遍历 b 所包含的所有 p.

证明:令 YP_i 表示第 i 个扩展输入控制依赖图所包含的 p 的集合,$i\in[1,\ldots,s]$.根据扩展输入控制依赖图的定义可知,$\forall p$ 都有一个 YP_i,使得 $p\in YP_i$.也就是说, YP_i 是对符号 p 的一种划分.

根据 YP_i 的划分,假定 $b=Yb_1\cup\ldots\cup Yb_s$,$Yb_i\subseteq YP_i$,因为 $b\notin ConditionSet$,则必有一个 Yb_i,使得对所有的 $ys_j\in Y_i$,都有 $ys_j\neq Yb_i$,$j\in[1,\ldots,l_i]$.

假定 $Yb_i=\{p_{i1},p_{i2},\ldots,p_{il}\}$,满足$\{p_{i1},p_{i2},\ldots,p_{im}\}\in ys_j$,$m<l$.根据标记路径的定义可知,必有一个 $p_{in}\in YP_i$,而 p_{in} 不在 ys_j 对应的路径上.同样,根据标记路径的定义可知,在 ys_j 对应的输入条件下,p_{in} 是不可能执行的,从而 p_{in} 是多余的.从而说明 b 是不可行的. □

因为标记路径数量不会超过程序的路径,而每个扩展输入控制依赖图至少有两条路径,因此生成 *ConditionSet* 的复杂度为 $O(P^s/s^s)$.这里,s 为扩展输入控制依赖图的个数,P 为程序的路径数.

按照上述方法,本节开始的 C 程序段的 p 组合为 p_1,$1-p_1$ 和 p_2,$1-p_1$ 和 $1-p_2$,从而得到的条件为 $x<1$,$x\geq 1$.

4 举 例

1) CFG 和控制依赖图

图 3 是一个示意程序框架及其控制流图.示意程序右边的注释说明了源程序语句和控制流图的对应关系.图中入口节点用双层空圈表示,出口节点用双层实圈表示.

示意程序的控制依赖图如图 4 所示.

2) 计算节点的执行计数和边的执行频率

此程序各个节点的执行计数和各个边的执行频率见表 1.如前所述,在表 1 中,我们把依赖输入节点对应于 TRUE 分支的执行频率用 p_i 表示,另一个分支用 $1-p_i$ 表示.

```
bool example(integer n,x,cnd1,cnd2) {
  i=0;                          ----Node 1
  while (i<n) {                 ----Node 2
    i++;                        ----Node 3
    if (cnd1>4) {               ----Node 3
      if (i<n–x)                ----Node 4
        S1;                     ----Node 5
      else if (cnd2=1)          ----Node 6
        S2;                     ----Node 7
      else S3;                  ----Node 8
    }
    if (cnd1<3)                 ----Node 9
      S6;                       ----Node 10
      else S7;                  ----Node 11
    S8;                         -----Node 12
  } }
```

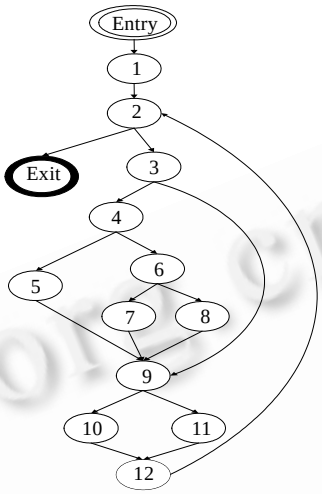

Fig.3 An example and its control flow graph

图 3 示意程序框架及其控制流图

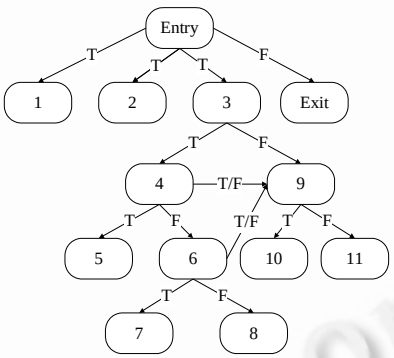

Fig.4 Control dependent graph of the example program

图 4 示意程序控制依赖图

Table 1 The executing count of nodes and the executing frequency of edges

表 1 节点的执行计数和边的执行频率

Node	Executing count	Edge	Executing frequency
entry	1	(*entry*,1)	1
1	1	(1,2)	1
2	n+1	(2,3)	$n/(n+1)$
3	n	(2,y)	$1/(n+1)$
4	$p_1 \times n$	(3,4)	p_1
5	$p_1 \times (n-x)$	(3,9)	$1-p_1$
6	$p_1 \times x$	(4,5)	$(n-x)/n$
7	$p_1 \times p_2 \times x$	(4,6)	x/n
8	$p_1 \times (1-p_2) \times x$	(6,7)	p_2
9	n	(6,8)	$1-p_2$
10	$p_1 \times n$	(9,10)	p_3
11	$(1-p_3) \times n$	(9,11)	$1-p_3$
12	n	(12,2)	1
exit	1		

3) 计算各节点的母函数及程序的 WCET 符号表达式

包含依赖输入分支的母函数公式由下面公式给出:

$G_{entry}(z)=z^{\tau_{entry}}$	$G_7(z)=z^{\tau_7}p_2G_6(z)$
$G_1(z)=z^{\tau_1}G_{extry}(z)$	$G_8(z)=z^{\tau_8}(1-p_2)G_6(z)$
$G_2(z)=z^{\tau_2}(G_1(z)+G_{12}(z))$	$G_9(z)=z^{\tau_9}((1-p_1)G_3(z)+G_5(z)+G_7(z)+G_8(z))$
$G_3(z)=z^{\tau_3}\left(n/(n+1)\right)G_2(z)$	$G_{10}(z)=z^{\tau_{10}}p_3G_9(z)$
$G_4(z)=z^{\tau_4}p_1G_3(z)$	$G_{11}(z)=z^{\tau_{11}}(1-p_3)G_9(z)$
$G_5(z)=z^{\tau_5}((n-x)/n)G_4(z)$	$G_{12}(z)=z^{\tau_{12}}(G_{10}(z)+G_{11}(z))$
$G_6(z)=z^{\tau_6}\left(x/n\right)G_4(z)$	$G_{exit}(z)=z^{\tau_{exit}}\left(1/(n+1)\right)G_2(z)$

因为控制流图CFG最多是一个强连通图,因此计算控制流图各节点母函数的复杂度为 $O(N^2)$,N 是控制流图的节点个数.

对上述公式,通过代入消减后,得:

$$G_2=z^{\tau_{entry,1,2}}+z^{\tau_{3,9,12}}\left[p_3z^{\tau_{10}}+(1-p_3)z^{\tau_{11}}\right]\left[(1-p_1)n+(n-x)p_1z^{\tau_{4,5}}+xp_1p_2z^{\tau_{4,6,7}}+xp_1(1-p_2)z^{\tau_{4,6,8}}\right]G_2/(n-1).$$

从中消除循环后可得

$$G_2=\frac{z^{\tau_{entry,1,2}}(n+1)}{n+1-z^{\tau_{3,9,12}}\left[p_3z^{\tau_{10}}+(1-p_3)z^{\tau_{11}}\right]\left[(1-p_1)n+(n-x)p_1z^{\tau_{4,5}}+xp_1p_2z^{\tau_{4,6,7}}+xp_1(1-p_2)z^{\tau_{4,6,8}}\right]}.$$

从而可得

$$G_{exit}=\frac{z^{\tau_{entry,1,2,exit}}}{n+1-z^{\tau_{3,9,12}}\left[p_3z^{\tau_{10}}+(1-p_3)z^{\tau_{11}}\right]\left[(1-p_1)n+(n-x)p_1z^{\tau_{4,5}}+xp_1p_2z^{\tau_{4,6,7}}+xp_1(1-p_2)z^{\tau_{4,6,8}}\right]}.$$

对 z 求微分后,将 z=1 代入后得:

$$\frac{\mathrm{d}}{\mathrm{d}z}G_{exit}\mid_{z=1}=\Delta+(p_3\tau_{10}+(1-p_3)\tau_{11})n+(n-x)p_1\tau_{4,5}+xp_1p_2\tau_{4,6,7}+xp_1(1-p_2)\tau_{4,6,8}.$$

在上式中,Δ表示 $\tau_{entry,1,2,3,9,12,exit}$.从这个例子可以看出,此表达式还是比较复杂的.

4) 根据控制依赖图产生扩展输入控制依赖图

根据图 2 的算法产生的依赖输入分支节点构成的扩展输入控制依赖图如图 5 所示.

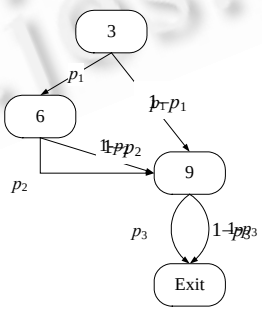

Fig.5 Input control dependent graph of he input dependent branch nodes

图 5 依赖输入分支节点构成的输入控制依赖图

5) 标记路径集合和各个 p 组合

根据上述算法产生的各个 p 组合见表 2.

Table 2 The annotated path set

表 2 标记路径集合

No.	Annotated path	No.	Annotated path	No.	Annotated path
1	p_1,p_2,p_3	3	$p_1,1-p_2,p_3$	5	$1-p_1,p_3$
2	$p_1,p_2,1-p_3$	4	$p_1,1-p_2,1-p_3$	6	$1-p_1,1-p_3$

6) 产生相应的条件表达式

上述例子的计算,其结果是共有 6 条路径(见表 2).其中,第 1 和第 3 条路径对应的条件组合分别为:$(cnd1>4)\wedge(cnd2=1)\wedge(cnd1<4)$和$(cnd1>4)\wedge(cnd2\neq 1)\wedge(cnd1<4)$,容易判断它们是矛盾的,从而为不可行路径.

对于剩余的 4 条可行路径,其对应的符号表达式以及相应的条件如下所示:

$$\begin{cases} \Delta+n\tau_{4,11}+(n-x)\tau_5+x\tau_{6,7} & \Leftarrow (cnd1>4) \text{ and } (cnd2=1)\\ \Delta+n\ \tau_{4,11}+(n-x)\tau_5+x\tau_{6,8} & \Leftarrow (cnd1>4) \text{ and } (cnd2\neq 1)\\ \Delta+n\ \tau_{10} & \Leftarrow (cnd1<3)\\ \Delta+n\ \tau_{11} & \Leftarrow (cnd1\leq 4) \text{ and } (cnd1\geq 3) \end{cases}.$$

能够看出,每个符号表达式都得以简化,且它们所对应的条件是正交的,即它们的定义域没有重叠.

5 相关工作和结论

WCET 分析是实时系统的重要研究领域之一,也是最近 10 多年的研究热点[13].本文提出了一种针对包含直接依赖输入分支程序的参数化 WCET 分析方法,其分析结果是该程序带条件的 WCET 符号表达式,不同的条件可能对应不同的符号表达式.

文献[1]提出一种计算循环的 WCET 的迭代方法,循环的时序行为迭代到收敛为止.这里的符号表达式只是针对循环的,同时,它是以循环迭代次数为参数的符号表达式.

文献[3]描述了一种自动参数化 WCET 分析技术,它使用抽象解释(abstract interpretation)导出流图节点的执行计数以及路径的约束,使用符号化 ILP 技术解决 WCET 界限的 IPET 计算问题.文献[2]使用数据流框架估算实时程序的符号化 WCET.该方法在已知程序条件递归关系的情况下,通过符号插装技术,能够计值 CFG 节点的符号化执行频率,最后,利用母函数获取程序精确的 WCET 符号表达式.需要注意的是,该方法只考虑循环控制分支的执行频率,而循环控制分支的执行频率比较容易分析,非循环控制分支的执行频率则难以确定[14].依赖输入分支正是非循环控制分支.以上这两种方法都有可能产生依赖于输入数据的参数表达式,但并不总是产生依赖输入的带条件的符号表达式,同时,他们没有考虑依赖输入的分支.

文献[4]通过使用标记把程序分成不同的基本路径,称为模式(mode),其模式既可能由输入数据确定,也可能由循环不变量确定.通过对每一种模式进行符号计值得到该模式的参数表达式.因为循环不变量通常也由输入数据确定,因此,文献[4]的模式通常对应于本文的一个条件.所以一般说来,我们的方法能够自动确定文献[4]中的模式以及模式和输入数据的对应关系.

与上述这些符号化 WCET 分析方法不同,我们的方法直接产生针对输入数据的符号表达式,这样的条件符号表达式简洁、直接,易于计算,能够在运行时快速产生实际的 WCET.

今后的工作包括 3 个方面:首先,扩展本分析方法,使之能够分析包含间接输入数据分支的程序;其次,把本分析方法集成到 WCET 分析工具中;最后,进一步把本文方法应用到具有流水线和高速缓存的处理器上.

致谢 感谢李梦君、王乐春、秦杰博士提出了很多宝贵意见.

References:

[1] Vivancos E, Healy C, Mueller F, Whalley D. Parametric timing analysis. In: Workshop on Languages, Compilers, and Tools for Embedded Systems. Utah: ACM Press, 2001. 88−93.

[2] Blieberger J. Data-Flow frameworks for worst-case execution time analysis. Real-Time Systems, 2002,22(3):183−227.

[3] Lisper B. Fully automatic, parametric worst-case execution time analysis. In: Proc. of the 3rd int'l Workshop on Worst-Case Execution Time (WCET) Analysis. Porto: IEEE Computer Society Press, 2003. 85−88.

[4] Chapman R, Burns A, Wellings A. Integrated program proof and worst-case timing analysis of SPARK Ada. In: Proc. of the ACM Workshop on Language, Compiler and Tool Support for Real-time Systems. New York: ACM Press, 1994.

[5] Aho AV, Sethi R, Ullman JD. Compilers, Principles, Techniques, and Tools. New York: Addison-Wesley, 1997.

[6] Wu Y, Larus JR. Static branch frequency and program profile analysis. In: Proc. of the 27th Int'l Symp. on Microarchitecture. San Jose: ACM Press, 1994. 1−11.

[7] Patterson JRC. Accurate static branch prediction by value range propagation. In: Proc. of the ACM SIGPLAN'95 Conf. on Programming Language Design and Implementation. La Jolla: ACM Press, 1995. 67−78.

[8] Graham RL, Knuth DE, Patashnik O. Concrete Mathematics: A Foundation for Computer Science. 2nd. New York: Addison-Wesley, 2002.

[9] Smith JE. A study of branch prediction strategies. In: Proc. of the 8th Annual Symp. on Computer Architecture. Minneapolis: IEEE Computer Society, 1981. 135−148.

[10] Cheatham TE, Holloway GH, Townley JA. Symbolic evaluation and the analysis of programs. IEEE Trans. on Software Engineering, 1979,5(4):403−417.

[11] Cytron R, Ferrante J, Rosen B, Wegman KMN, Zadeck FK. Efficiently computing static single assignment form and the control dependence graph. ACM Trans. on Programming Languages and Systems, 1991,13(4):451−490.

[12] Baase S, Gelder AV. Computer Algorithms: Introduction to Design and Analysis. 3rd. New York: Addison-Wesley, 2001.

[13] Puschner P, Burns A. Guest editorial: A review of worst-case execution-time analysis. Real-Time Systems, 2000,18(2−3):115−128.

[14] Ball T, Larus J. Efficient path profiling. In: Proc. of the IEEE/ACM Int'l Symp. on Microarchitecture. Paris: IEEE Computer Society Press, 1996. 46−57.

姬孟洛(1963－),男,河南洛阳人,博士生,高级工程师,主要研究领域为实时系统分析,面向对象设计.

齐治昌(1942－),男,教授,博士生导师,主要研究领域为软件工程,计算机教育.

王怀民(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为分布计算系统,实时系统.