第28卷 第11期 计 算 机 工 程 2002年11月
Vol.28 № 11 Computer Engineering November 2002
· 基金项目论文 · 文章编号：1000—3428(2002)11—0095—02 文献标识码：A 中图分类号：TP313.3

符号执行过程的DFA和CFA

陈凯明[1,2]，刘宗田[3]，王武荣[1]，叶 勇[1]
(1. 中国科学技术大学计算机科学技术系，合肥 230052；
2. 合肥工业大学微型机研究所，合肥 230009；3. 上海大学计算机学院，上海 200072)

摘 要：将含有库函数名的汇编代码程序翻译成中间语言程序，然后再转换成高级语言程序可降低逆编译的复杂性。文章介绍了从汇编程序到中间语言程序的转换过程中，采用基于数据流和控制流分析的符号执行方法。该方法已在基于知识的逆编译系统DECLER中实现。

关键词：逆编译；中间语言；数据流；控制流

DFA and CFA of Symbolic Execution

CHEN Kaiming[1,2], LIU Zongtian[3], WANG Wurong[1], YE Yong[1]
(1. Department of Computer Science & Technology, University of Science and Technology of China, Hefei 230052;
2. Institute of Microcomputer, Hefei University of Technology, Hefei 230009;
3. College of Computer Science of Shanghai University, Shanghai 200072)

【Abstract】Assembler code program containing library function name is firstly translated into intermediate language program and then into high level language program to reduce complexity of decompilation. Symbol execution based on control flow and data flow is described in the translating assembler program into intermediate program. The method has been implemented in the decompilation system based on knowledge named DECLER.

【Key words】Decompilation; Intermediate language; Data flow ; Control flow

逆编译的过程就是将高级语言对应的可执行程序翻译回高级语言程序的过程。它对于软件理解、构件复用、软件安全验证、计算机病毒清除等都具有一定的作用。对于一些含有符号信息的代码（Java语言），其逆编译的过程相对简单，技术也较成熟，但像C、C++等一些语言的反编译研究尽管开始很早，但其理论和技术仍有待完善，尤其是难以满足不断出新的语言和编译器版本发展的需要。

逆编译的过程主要有两个部分：一是可执行代码的控制结构转换；二是数据类型恢复。在将含有库函数名的汇编程序翻译成高级语言程序时，先将其翻译成中间语言程序然后再翻译成高级语言程序可以显著降低逆编译过程的复杂性，由于中间语言程序也有良好的结构和可读性，因此在对汇编语言程序进行符号执行时进行较为完备的控制流和数据流分析，尽可能多地保留可执行文件中的隐含信息以便数据类型和控制结构恢复时使用。

1 语法表示

(1) 汇编语言语法

<函数宣称>：：=<函数>｛<函数>｝
<函数> ：：=<入口地址>｛<汇编指令>｝
<入口地址>：：=<地址码> <OP> <OP1>
<汇编指令>：：=<地址码> <操作码> <操作数1> <操作数2>
<地址码> ：：=<段地址>：<段内偏移>

(2) 中间语言语法

<函数宣称>：：=<函数>｛<函数>｝
<函数> ：：=<函数名>｛<语句>｝
<语句> ：：=<标号><语句>｜<函数调用语句>｜
<赋值语句> ｜<条件转移语句>｜<跳转语句>｜
<返回语句> ｜<开关语句> ｜<表达式语句>

(3) 汇编语言程序规范化

1）除去程序中无效指令；2）统一操作数的格式为四元式；3）将隐式操作数转换为显式操作数；4）转换开关结构引导区，分离开关表；5）保留汇编语言的显式操作数格式以及寻址方式。

2 控制结构分析

(1) 汇编程序的基本块类型

基本块首行：1）函数的第一条有效指令；2）直接或条件转移指令直接转向的指令；3）开关表中所标识的地址处指令；4）紧接着直接或条件转移指令后的首指令。

基本块类型：1) 开始基本块：每个函数的第一条有效指令前面的指令块；2) 有效基本块：从一基本块首行到下一基本块首行之前的所有指令块；3) 开关基本块：转向地址表，由JMP间接寻址方式访问；4) 结束基本块：在函数或过程退出之前的所有无效指令组成的指令块。

无效指令：不与源程序的任何功能语句对应的指令，如PPSHBP，MOVBP, SP。

有效指令：子程序中除去无效指令后的所有指令。

(2) 程序流图表示

前驱块和后继块：如果基本块m有指令直接或条件转向n，或者按地址顺序n块紧接在m块之后，则称m为n的前驱块，n为m的后继块。

定义：程序流图定义为五元式的有向图，G=(Node, Edge,S,H) N为结点集，每个结点代表一个基本块，Edge为边集，每条边表示基本块间关系以及真假转移等信息，S为开始基本块，H为HALT集合。

基金项目：教育部博士点基金课题“面向Web的组件化CASE模型研究”（97035901）
作者简介：陈凯明(1964～)，男，副教授，研究方向：人工智能与软件工程；刘宗田，博导、研究员；王武荣、叶 勇，副教授
收稿日期：2001-11-26

(3) 控制结构提取

1) if_goto语句 (图1)

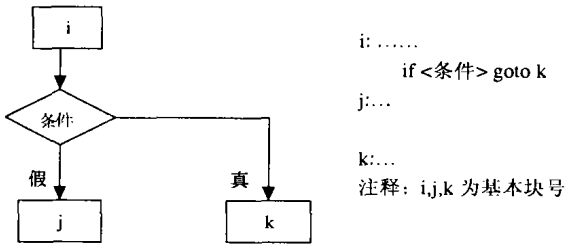

图1 if…goto 语句流图

2) switch结构 (图2)

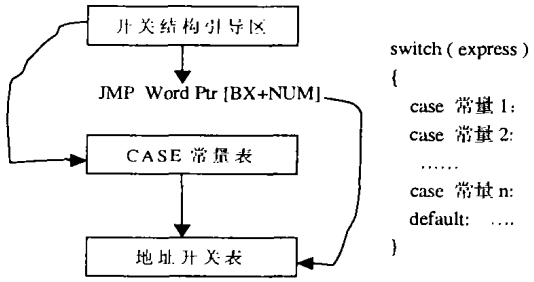

图2 switch 流程图

3 数据流分析(DFA)

(1) 寄存器的定值链

在符号执行程序之前，必须对有效基本块中所使用的寄存器进行检查，以确定寄存器的出现情况。定义性出现是指寄存器在某基本块中第一次出现时被赋以新值，而没有引用其原值的出现，使用性出现是指寄存器在某基本块中第一次出现时除去定义性出现的出现，如MOV AX,BX。 AX：定义性出现；BX：使用性出现。

反向传播，是指在某个基本块中为使用性出现而在前驱块中没有明确指出为使用性出现时要把其前驱块中关于该寄存器的出现标记为使用性出现。

(2) 寄存器变量

优化的结果，高级语言程序中使用频率较高的变量在实现时经常被分配为寄存器，因此在翻译时要确定哪几个寄存器对应为高级语言中的变量。

判定的准则是在起始块中若有PUSH SI或者PUSH DI指令，则SI、DI表示变量。

(3) 重复块处理

定义：对于某个有效基本块，其中存在寄存器的使用性出现和至少有两个前驱或两个以上的前驱，则称其为重复基本块。

处理方法：

1) 重复块分裂 (如图3、4)

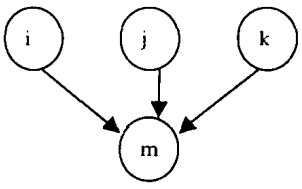

图3 重复块

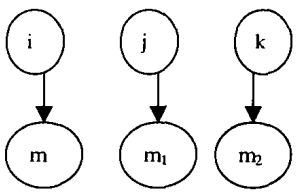

图4 分裂后结果

优点：不改变源程序的数据空间。

缺点：使得符号执行时间延长、复杂度增加。

2) 引进临时变量 (如图5)

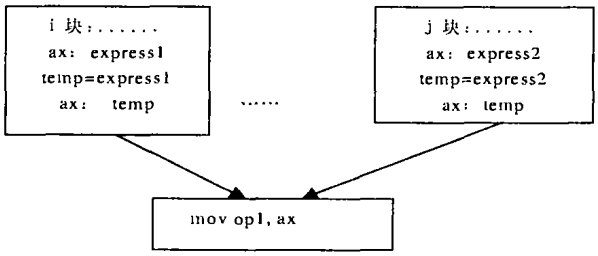

图5 引进临时变量示意图

对于同一个重复块可以在不同的前驱中引进相同的临时变量，然后将使用性出现的寄存器符号内容改变为临时变量，使得所有前驱关于该寄存器的符号内容相同。

优点：翻译后的结果易于阅读。

(4) 函数的返回值

定义HALT集合：

1) 结束基本块属于HALT集合，结束基本块的直接前驱属于HALT集合；

2) 如果基本块 i 仅有一条转移指令转向HALT集的基本块，则 i 也属于HALT集合；

3) 重复1)、2)，直至该集合不再改变为止。

若寄存器AX在HALT集的基本块中为定义性出现，则该函数有返回值，否则不返回值。

(5) 数据类型分析

汇编语言程序中没有明确的变量类型，高级语言中数据类型隐式地包含在操作码和操作类型中，符号执行时主要提取操作数的单元长度信息，如Word(2字节)、Byte(1字节)、Dword(4字节)、Qword(8字节)以及与其他操作数相互作用的信息[5]，长整型数要结合指令一起判断，浮点数的操作指令与整形数操作指令有明显区别再结合操作数较容易判定。

由于复合数据类型恢复非常困难，使得在中间语言程序中不显式给出类型信息，而只给出对应的地址计算表达式以及操作长度等信息，用户可以根据上下文理解。

4 符号执行过程

具体过程如图6所示。

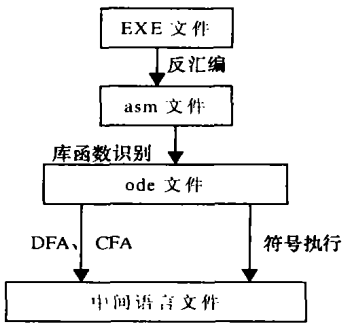

图6 符号执行过程

5 结论

文中所描述的工作已在PC80X86机器的"基于知识的逆编译系统DECLER[6]"中实现，通过大量的实例测试表明，基于CFA和DFA的符号执行方法具有良好的效果。

(下转第122页)

表1 一张空间数据表

ID	类型	笔	填充颜色或图案	外接矩形	坐标
101	1(点)	空	背景颜色	CRect	位图编号
102	2(直线)	(线宽、线型、颜色)	空	同上	起、终点坐标
103	3(折线)	同上	空	同上	折线数、每条折线有顶点数、顶点坐标
104	4(弧)	同上	空	同上	外接椭圆坐标、弧度
105	5(椭圆)	同上	整数 (0表示不填充)	同上	左上角和右下角坐标
106	6(矩形)	同上	同上	同上	左上角和右下角坐标
107	7(圆角矩形)	同上	同上	同上	左上角和右下角坐标、圆角弧度
108	8(封闭多边形)	同上	同上	同上	多边形数、每个多边形有顶点数、定点坐标
109	9(标注)	字体	空	同上	文本内容

3 系统实现

整个系统在VC++6.0平台上开发，相当于一个小型的MapInfo系统，具备基本的矢量图形系统，但它改进了MapInfo对空间数据与属性数据的分开存储问题。我们设计开发的空间数据库系统完全采用面向对象的技术，采用了面向对象数据模型和自己设计的面向对象数据库模型。它借鉴了SDE的存储结构，建立了自己的小型空间数据库系统。

此系统采用Client Server结构，二维地图显示及处理作为前端Client，通过应用接口与空间数据库(Server)进行数据交换。应用接口作为中间层将前端应用于空间数据库相隔离，增加了系统的灵活性和可扩展性。应用接口的功能在于使上层应用可以处理空间数据库系统的数据，包括：访问空间数据库中的元数据；访问空间数据库中的空间数据；设定查询条件进行数据查询；对空间数据库进行空间数据和属性数据的刷新。

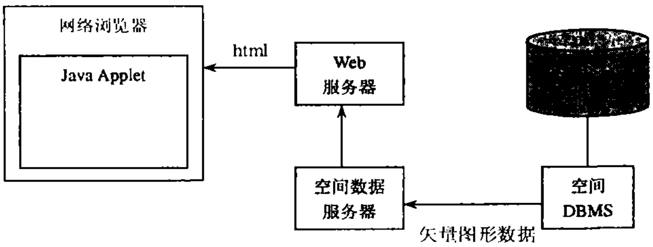

图3 网络数据访问模型

采用集成式地理信息系统的内嵌式数据库，其数据库引擎被GIS系统平台完全封装，并且可以在同一进程中对数据库进行操作，虽效率较高，但内嵌数据库缺乏商业数据库所拥有的开放性、一致性、安全性和标准化等一系列重要特性，所以我们也考虑到与工业标准的大型关系数据库进行集成，采用ODBC使这个地理信息系统能够访问SQL Server中的数据，便于利用已有的空间数据。

此系统与Internet相结合，接收浏览器发出的查询请求，进行相应的空间数据库查询，并将结果以图形的方式返回给浏览器。这样在任何一个Internet上的浏览器中可以访问空间数据库的内容。网络数据访问模型[4]见图3。

4 结束语

我们设计和实现的空间数据库系统采用基于对象的空间数据模型，具有一体化的空间数据组织方法，目前该数据库已在一个小型的GIS平台上得到实际应用，效果良好。在局域网上运行，利用RDBMS成功的互操作性，可以实现真正的Client/Server计算，并在系统级、数据库级实现信息共享。各终端只需发出请求，中心数据库就会快速地返回结果，数据检索效率较高、速度较快，能够满足基本的需求。

基于RDBMS的空间数据库技术，对于多用户、大数据量的GIS应用有着较大优势。基于RDBMS的空间数据库技术在部门GIS和企业级GIS的建设中，可以降低系统的复杂程度，提高开发效率，提高系统的性能。基于RDBMS的空间数据库技术还为WebGIS的建设提供数据支持，使其远程访问数据的能力得到提高。实现WebGIS时，制图服务器和数据管理服务器可以保持相对的独立以优化系统结构。

参考文献

1 ESRI White Paper. Spatial Database Engine. 1998
2 陈 晟, 景 宁, 孙茂印等. X-2000空间数据库系统的设计和实现. 国防科技大学学报, 1998, 20(3)
3 Hilbert K I. R-tree: An Improved R-tree Using Fractals. Proceedings of the 20th VLDB Conf., 1994
4 Brodie R S. Integrating GIS and RDBMS Technologies During Construction of a Regional Groundwater Model. Environmental Modeling & Software, 1999,14: 119-128

☆☆

(上接第96页)

参考文献

1 赵 磊, 王开铸. C反编译控制流恢复的形式化描述及算法. 计算机学报, 1998, 21(1):87-91
2 申例民. 基于CFA和DTA的逆编译方法. 小型微型计算机系统, 1998, 19(3):19-23
3 陈凯明, 刘宗田. 逆编译中开关结构的识别与恢复. 小型微型计算机系统, 2000, 21(12):1279-1281
4 Cifuentes C. Inter-procedural Data Flow Decompilation.Journal of Programming Languages, 1996, 4(2): 77-99
5 Breuer P T, Bowen P. Decompilation: The Enumeration of Types and Grammars. ACM Transaction on Programming Languages and Systems, 1994, 16(5): 1613-1647