第35卷 第22期 计 算 机 工 程 2009年11月
Vol.35 No.22 Computer Engineering November 2009
·软件技术与数据库· 文章编号：1000—3428(2009)22—0062—03 文献标识码：A 中图分类号：TP311.12

基于数据流分析的寄存器参数恢复方法

方 霞，尹 青，蒋烈辉，黄 海，何红旗
(解放军信息工程大学信息工程学院，郑州 450002)

摘 要：针对在反编译IA32体系结构可执行程序过程中涉及的寄存器参数恢复问题，基于IA32适用的ABI约定，分析寄存器参数的特征，借鉴数据流分析中到达-定义分析及使用-定义链等经典方法，利用寄存器定义和使用等信息，提出一种寄存器参数的识别、处理及实参恢复的方法。

关键词：反编译；寄存器参数；参数恢复；数据流分析

Register Parameter Recovery Method Based on Dataflow Analysis

FANG Xia, YIN Qing, JIANG Lie-hui, HUANG Hai, HE Hong-qi
(Institute of Information Engineering, PLA Information Engineering University, Zhengzhou 450002)

【Abstract】Aiming at the register parameter recovery problem in the process of decompiling the executable programs on the IA32 architectures and based on Application Binary Interface(ABI) conventions for IA32, this paper analyzes the characteristics of register parameters, and uses register definition and usage information to propose a method for register parameter recognition and recovery information of registers, which is gained by reaching definition analysis and ud-chain.

【Key words】decompilation; register parameter; parameter recovery; dataflow analysis

1 概述

反编译是将机器语言程序转换成与之功能等价的高级语言程序的过程，是编译的逆过程[1]。如图1所示，反编译一般包括5个主要部分，其中的过程恢复有助于更好地理解各过程之间的调用关系、数据传递等信息。过程恢复分为参数恢复、过程定义和引用恢复、返回值恢复3个方面[2]。过程的作用很大程度上体现在对参数的操作和处理上，因此，参数恢复对理解过程间的数据传递信息有重要意义。

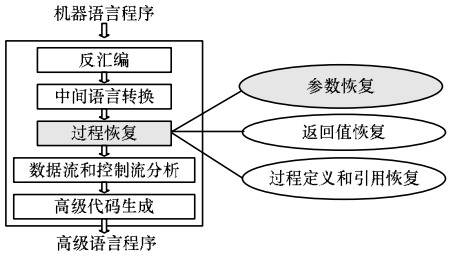

图1 反编译过程

参数恢复是通过分析得到过程的参数及该过程被调用处相应的实参。根据传递方式的不同，参数可以分为栈参数和寄存器参数2种。栈参数是指调用者通过栈传递参数给被调用者，寄存器参数是指调用者通过寄存器传递参数给被调用者，其中，传递方式、传递顺序等根据不同的调用约定而有所不同。

澳大利亚昆士兰大学在先后开发的反编译器Dcc、二进制翻译系统UQBT以及反编译器Boomerang中，设计实现了过程抽象语言(PAL)[2]，对一个过程中与机器体系结构相关的部分进行抽象，其中也包括参数等信息的描述，从而可以进行参数恢复。PAL的设计目的在于建立一个多元多目标的通用框架结构，无形中增加了问题的难度，并且其中没有关于寄存器参数的处理。信息工程大学在开发IA64/Linux到Alpha/Linux的静态二进制翻译器过程中将PAL扩展到IA64框架下[3]，并且提出一种基于IA64 ABI的栈参数和寄存器参数的处理方法，但由于ABI规定不同，不适用于IA32体系结构。中科院计算技术研究所在设计开发二进制翻译系统Bitran的过程中，也涉及到过程恢复的问题，开发者的做法是为每一个过程建立一个虚拟栈[4]，再考虑被调用过程对栈的使用情况，该方法只能解决栈参数，不能处理寄存器参数。

相对于反编译中的其他问题，针对寄存器参数的理论研究较少，但寄存器参数作为参数的重要组成部分，对于理解过程的行为以及与其调用者的关系至关重要，是一个不能忽视的问题。本文根据在反编译IA32体系结构可执行程序过程中涉及到的寄存器参数恢复问题以及IA32适用的ABI(Application Binary Interface)约定，提出一种利用数据流信息寻找形参和实参并恢复过程引用和定义的方法。

2 数据流分析技术

数据流分析是沿着程序可能的执行路径收集数据的使用、定义及相互关系信息的过程。数据流分析一般用于编译优化，它能从程序代码中收集程序的语义信息，通过代数的方法确定变量的定义和使用，从而更好地进行代码优化。本文将数据流分析应用于反编译中的寄存器参数恢复，利用寄存器的定义和使用信息，确定过程间传递的信息。下面就本文用到的数据流分析中的经典应用到达-定义分析及使用-定

基金项目：国家“863”计划基金资助项目“软件特征分析及基于水印的可控性技术研究”(2006AA01Z409)

作者简介：方 霞(1984－)，女，硕士研究生，主研方向：软件逆向工程；尹 青，副教授、博士；蒋烈辉，教授、博士生导师；黄 海，硕士研究生；何红旗，讲师、硕士

收稿日期：2009-03-03 **E-mail：**fangxia_private@163.com

义链(ud 链)[5]进行讨论。

设 Q 是程序中变量 A 的定义点(即在 Q 点上的语句对 A 赋值)，则对于程序中的某点 P，若流程图中存在从 Q 到 P 的通路，且在此通路上不再含有 A 的其他定义点，则称 A 在点 Q 的定义能到达点 P。建立 ud 链所需要的信息是通过到达-定义分析收集得到的。若在程序的某点 u 引用了变量 A 的值(即以 A 为操作对象)，则将流程图中能到达 u 的所有 A 的定义点所组成的集合称为 A 在引用点 u 的 ud 链。在进行到达-定义分析之前先定义 4 个集合：

(1)In[B]是能到达 B 的开始点的所有变量定义点的集合。

(2)Out[B]是能到达 B 的结束点的所有变量定义点的集合。

(3)Gen[B]是在 B 中定义并且能够到达 B 的结束点的定义点的集合。

(4)Kill[B]对于 B 内定义的每个变量 v 而言，包含除了出现在 Gen[B]集中的定义点以外其他所有 v 的定义点。

假定每个块的 Gen 和 Kill 都已计算，可以建立如下 2 组方程，它们将 In 和 Out 联系起来：

$$In[B] = \bigcup_{P是B的前驱} Out[P] \quad (1)$$

$$Out[B] = Gen[B] \cup (In[B] - Kill[B]) \quad (2)$$

如果流图有 n 个基本块，则可以得到 $2n$ 个方程。对这 $2n$ 个方程，从第 1 个基本块 B 开始计算，把 In[B]是空集作为初始条件，从前往后迭代求解，即可得到所有基本块的 In 集合和 Out 集合。

若所有基本块 B 的 In[B]已求出，则可根据如下 2 条规则，求出能到达 B 内任一变量 A 的引用点 P 的全部定义点，建立起每个变量引用的 ud 链。

规则 1 若在 B 中位于 P 之前有 A 的定义点，则离 P 最近的定义点就是能到达 P 的唯一定义点。

规则 2 若在 B 中位于 P 之前无 A 的定义点，则能到达 P 的 A 的定义点只能是包含在集合 In[B]中的那些 A 的定义点。

通常意义上的数据流分析是针对变量进行分析，本文的工作是得到寄存器的定义和使用信息，从而进行后续的分析，因此，在实际工作中数据流分析的对象是寄存器。根据上述方法对语句中的寄存器进行到达-定义并建立 ud 链，是进行寄存器恢复各项分析的基础。

3 寄存器参数恢复

寄存器参数即被调用者在使用某一个寄存器之前没有对其赋初值，而是使用了该寄存器在调用者中的内容，可以看作是由调用者传递给被调用者的参数。图 2 所示的代码是从某程序的 IDA Pro 反汇编结果界面中截取的片断。可见，寄存器 eax, edx, ecx 在被使用之前都没有进行定义，因此，是该过程的寄存器参数。

```
CODE:0040148C        sub_40148C  proc near
CODE:0040148C
CODE:0040148C                            push    ebx
CODE:0040148D                            push    esi
CODE:0040148E                            push    edi
CODE:0040148F                            push    ebp
CODE:00401490                            mov     ebx, ecx
CODE:00401492                            mov     esi, edx
CODE:00401494                            mov     ebp, eax
CODE:00401496                            mov     dword ptr [ebx+4]
                                  ⋮
```

图 2 某可执行程序的 IDA 反汇编结果片断

寄存器参数恢复可以分为 3 个方面：寄存器参数的识别；过程定义和引用恢复；寄存器实参恢复，其中，寄存器参数的识别相对较难也较复杂。本文在描述中假设过程的栈参数已经过识别并处理。

3.1 寄存器参数识别

寄存器参数的识别可以分为 2 步：首先识别过程中所使用的没有赋初值的寄存器，然后将所识别的寄存器与可能的寄存器参数列表匹配，最终确定真正的寄存器参数。

寄存器参数是过程中使用的没有赋初值的寄存器。根据 ud 链的含义，如果某个被使用的寄存器 ud 链为空，即找不到该寄存器的定义，也即没有赋初值的寄存器被使用，可以根据每条指令中寄存器的 ud 链，查找过程中 ud 链为空的寄存器，该寄存器可能是没有赋初值即被使用的寄存器。之所以说可能是没有赋初值而被使用，是因为需要根据汇编代码中的语句类型分别处理判断。使用寄存器的有运算类语句、调用语句、条件跳转语句、有返回值的返回语句以及入栈语句。运算类语句、条件跳转语句、调用语句和有返回值的返回语句可以直接判断语句中 ud 链为空的寄存器并标记为可能的寄存器参数，对于入栈语句则需要特殊考虑。代码中入栈指令有 3 种类型：(1)将参数入栈，该类 push 没有相应的 pop 语句。(2)保存寄存器，该类 push 有相应的 pop 语句，操作数是相同的寄存器。(3)赋值，该类 push 有相应的 pop 操作对应，但操作数不同。其中，第(1)类 push 操作在前端完成栈参数的识别后，将其操作数加入相应的调用语句并且将 push 语句删除。对于其他 push 语句，如果操作数是寄存器并且 ud 链为空，首先找到其相应的 pop 语句，并判断操作数是否相同：如果相同，属于第(2)类入栈语句，该寄存器并没有被真正使用，因此，不能作为寄存器参数；如果不同，属于第(3)类 push 操作，其实质是 push 的操作数对 pop 的操作数的赋值，因此，push 中的该寄存器被使用。

根据上述策略可以识别过程中没有赋初值而直接使用的寄存器，但并不是所有识别出的寄存器都是过程的寄存器参数。IA32 ABI 规定，虽然 8 个通用寄存器对调用过程和被调用过程都是可见的，但是 ebx, esi, edi, esp, ebp 是属于调用者的寄存器，如果被调用者要使用这 3 个寄存器，必须入栈保存其内容，eax, ecx, edx 则可以不保存内容直接使用，因此，寄存器参数只能是 eax, ecx 和 edx 中的一个或者几个。在进行上述识别之后，将所识别的寄存器与 eax, ecx, edx 相匹配，如果是 3 个之中的寄存器，则确定为过程的寄存器参数。

3.2 过程定义和引用恢复

完成寄存器参数识别并确定了存储参数的寄存器后，即可修改过程的定义以及在调用者中对过程的引用。过程的定义是针对被调用者的，将识别的寄存器参数加入到过程的参数列表，修改过程的定义；过程的引用是针对调用者的，如果某个过程识别出寄存器参数，那么就需要修改所有调用该过程调用处的实参列表。

图 3 表示了 2 个过程的调用关系，其中，过程 Sub_S 调用过程 Sub_A 且调用过程中不存在栈参数。根据 3.1 节中描述的方法可以得到寄存器 eax 和 ecx 是过程 Sub_A 的寄存器参数。在 Sub_A 中，假设 2 个参数分别命名为 arg1 和 arg2，并且将相应的寄存器参数名字改为 arg1 和 arg2，由此完成过程的定义。在 Sub_S 中，根据被调用过程 Sub_A 的分析，已知 eax, ecx 为寄存器参数，将这 2 个寄存器作为实参传递给被调用者，完成对过程 Sub_A 的引用。

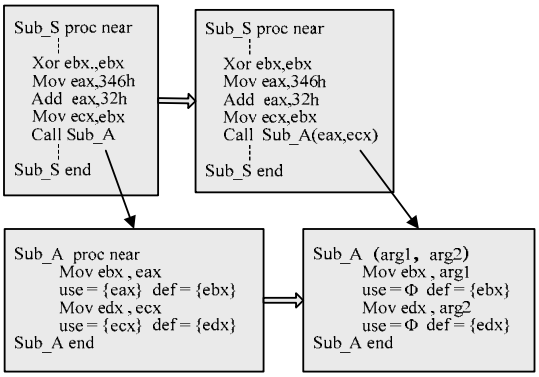

图 3　过程寄存器参数示例

3.3　实参的恢复

对寄存器参数进行识别后即可确定被调用过程中被作为参数传递的寄存器。但由于寄存器是程序代码中用于临时存储和转移数据的临时存储位置，其中存储的真正数值可能来自于某个存储单元或者常量，因此实参恢复的任务就是追踪代码中寄存器的真正值，并在过程调用处进行替换。

继续 3.2 节的例子，恢复实参的目的是得到 call 语句中 eax 和 ecx 的真正值。利用前面数据流分析技术建立过程中所有寄存器的 ud 链，回溯到寄存器的真正值。回溯需要设定一个合理的停止点，寄存器是依赖于机器的底层概念，而存储单元和变量不依赖于机器，符合高级语言的特性，因此，回溯遇到存储单元或常量即停止，此时得到的可能是由存储单元和常量组成的表达式。如图 4 所示，根据 call 中参数 eax 和 ecx 的 ud 链，可以得到如下回溯过程(尖括号内为此次回溯对应的定义的指令序号)：

eax = eax + 32h<3> = 346h<2> +32h

ecx = ebx<3>= 0h<1>

得到 2 个寄存器参数的实际值，分别为(346h + 32h)和 0h。对 Sub_S 调用 Sub_A 的调用指令中的参数进行替换，即可得到高级语言形式的调用方式。

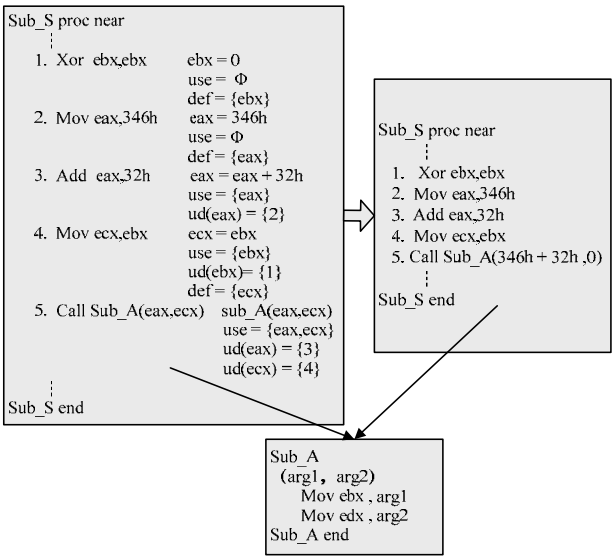

图 4　寄存器参数实参恢复示例

4　寄存器参数恢复实现算法

寄存器参数识别和过程定义恢复是对被调用者而言的，参数引用恢复和实参恢复是对调用者而言的，整个过程分为 2 个阶段，分别在调用过程和被调用过程中完成。寄存器参数的识别要在实参恢复之前完成，因此，被调用过程必须在调用过程之前进行寄存器参数的识别和参数引用的恢复。为了满足此条件，按照过程调用深度优先序列的逆序依次对每个过程进行处理，假设程序中有 *n* 个过程，入口过程为 main。以 main 为开始按照深度优先遍历程序中的过程，得到深度优先序列 L，然后将 L 逆序得到 L’，对 L’中过程用以下算法进行处理，完成寄存器参数的恢复：

输入 过程的控制流图以及所有调用本过程的调用点

(1)按照方程 1 和方程 2 建立每个基本块的 IN 和 OUT 集合

(2)按照规则 1 和规则 2 建立基本块中所使用寄存器的 ud 链

(3)寄存器参数模版列表 PL = { eax, ecx, edx}

(4)寄存器参数列表 PL’ = ∅

(5)For 过程中的每条语句

(6)If(寄存器 ud 链为空)

(7)If(语句类型为运算类||调用||条件跳转||返回语句)

(8)将该寄存器加入到 PL’中

(9)Else if(语句类型为入栈 PUSH&&操作数类型为寄存器)

(10)根据相应算法查找与之对应的出栈 POP

(11)If(PUSH 与 POP 操作数不相同)

(12)将该寄存器加入到 PL’中

(13)For (PL’中的每个元素)

(14) If(该元素不属于 PL)

(15)将该元素从 PL’中删除

(16)将 PL’中的寄存器加入到过程的形参列表中

(17)For(每个调用本过程的调用点处)

(18)将 PL’中的寄存器加入到调用点处调用语句的实参列表中并特殊标记该类参数

(19)For(过程中每条语句)

(20)If(语句类型为调用语句)

(21)For(调用语句的每个参数)

(22)If(该参数是被特殊标记的寄存器参数)

(23)根据 ud 链回溯直到得到由常量和存储单元组成的表达式

5　结束语

参数恢复是反编译中的关键步骤，能否有效地解决该问题，直接关系到过程间调用信息传递的正确性。本文根据二进制代码反编译过程中的寄存器参数恢复问题，提出了一种基于数据流分析的寄存器参数恢复方法，对寄存器参数恢复具有一定的参考价值。目前该方法已经应用于笔者正在开发的 IA32 体系架构反编译系统，并且能够正确地完成寄存器参数的识别和恢复，对 IA32 反编译系统的实现起到了重要的作用。

参考文献

[1] 陈福安, 刘宗田, 李　力. 8086C 语言反编译系统的设计及实现技术[J]. 小型微型计算机系统, 1993, 14(4): 10-18.

[2] Cifuentes C, Simon D. Procedure Abstraction Recovery from Binary Code[C]//Proceedings of the Conference on Software Maintenance and Reengineering. Washington, USA: IEEE Computer Society, 2000.

[3] 付　文, 魏　博, 张天雷. 过程恢复技术在 IA64 二进制翻译中的应用与实现[J]. 计算机工程与应用, 2006, 42(21): 81-89.

[4] 马湘宁, 张兆庆, 冯晓兵. 二进制翻译中的过程恢复技术[J]. 计算机工程与应用, 2002, 38(19): 1-5.

[5] 陈火旺, 钱家骅, 孙永强. 程序设计语言编译原理[M]. 北京: 国防工业出版社, 1984.

编辑　张　帆