C 逆编译系统的中间语言的一种优化表示方法①

胡 荣 范明钰 王光卫 宫亚峰 (电子科技大学 软件学院 四川 成都 610059)

摘 要: 数据类型的重新定义、控制流恢复和自定义函数或结构体的识别是结构化语言逆编译过程中的难点,以往研究多采用改进汇编语言的数据类型表示和增加自定义函数或结构体的特征量的方式来优化逆编译结果。使用中间语言抽象表示逆编译后得到的汇编语言的语言形式,并设计了一种新的逆编译模式,一定程度上提升了逆编译结果的识别率、可读性和完整性。

关键词: 反编译; 结构化语言; 中间语言; 自定义函数; 控制流

Reverse C Compiler Optimization of the Intermediate Language Representation of One

HU Rong, FAN Ming-Yu,WANG Guang-Wei, GONG Ya-Feng (Department of Software College, University of Electronic Science and Technology of China, Chendu 610059, China)

Abstract: To redefine the data types, control flow restoration and custom function, or structure of recognition, are structured languages. In the process of reverse compilation difficulties, previous studies use more data types in assembly language to improve representation and increase the custom function or structure of the characteristic quantities as a way of optimizing the reverse compile the results. This article uses the intermediate language to be compiled against the abstract representation of the assembly language of the linguistic forms and designs a new reverse compile mode to reverse compilation. The results are an improved recognition rate, readability, and integrity.

Keywords: de-compilation; structured language; intermediate language; self-defined functions; control flow

逆编译是把二进制可执行代码转换为等价的中级或者高级语言程序的过程,是软件逆向工程的一个最重要组成部分。它在软件重用、程序跨平台移植、软件漏洞分析、系统安全性检测等方面有着广泛的应用领域[1]。国内外对可执行程序逆编译后得到的汇编语言的语言结构形式的抽象表示一直没有可以实际应用的研究成果[2],而这样的抽象表示,正是由汇编语言向高级语言转换的一种重要转换方法。

1 中间语言定义

1.1 中间语言的由来

随着计算机科学理论和技术的发展,逆编译的组织过程和结构内容也在不断变化,基于前人的研究成果,同时结合实际项目,本文针对逆编译系统中的重现完整性和识别率两大关键指标,通过使用结构化分析、模板识别等技术,建构了一种结构化的中间语言并使用这种语言来抽象表示逆编译后的汇编语言,这种中间语言对数据类型的重新定义、控制流的表示基本等同于高级语言,从而优化了复杂数据结构的识别和控制流的重现;这种中间语言的结构化特征则提升了自定义函数或结构体的识别效率。这样就脱离了过去根据跳转地址直接生成临时数据类型[3]、修改库函数的特征点、改进库函数特征点的存储结构来加快搜索匹配等单纯的逆向算法进行反编译的模式[4],从而在语言系统层面来改进整个逆编译系统。

① 基金项目:国家高技术研究发展计划(863)(2009AA01Z403,2009AA01Z435);北京电子科技学院开放基金(KFHT200704);国家自然科学基金(60373109,60673142)
收稿时间:2010-01-12;收到修改稿时间:2010-03-02

1.2 中间语言的定义

1.2.1 中间语言各层定义

本文对众多不同的反编译器的汇编语言结果采用自顶向下的结构化分析方法——将汇编语言的指令序列通过分析逐渐拆分成 C 语言语法上的子程序或语句块，然后参考 C 语言程序的结构分层，建立起一个中间语言的层次结构[5]，进而分析出一种数据类型与地址无关的中间语言，并将其自定义为一种自适应程序数据类型的中间语言 AIL(adaptive intermediate language)，用其来优化表示基于可执行程序的逆编译过程。AIL 的层次结构如图 1：

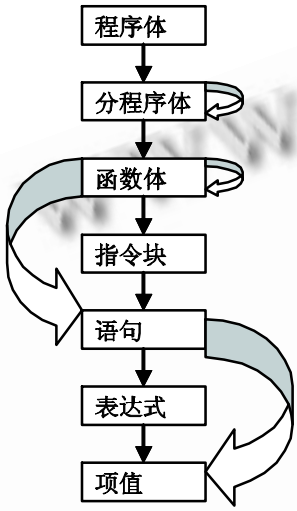

图 1　AIL 的层次结构图

由图 1 可以看出，AIL 语言的结构层次由于有自跨层，所以层与层之间形成偏序的关系；在层次分解的过程中，某些同层的元素会有一定的结构改变；某些下层元素在一定条件可以组成上层元素。整个层次对应类 C 语言的结构层次，但每一层都是根据转换汇编语言的逻辑顺序而具体设定的。

使用了高级语言程序的类型定义、变量说明、结构化控制语句和表达式是中间语言与汇编语言的区别，具体的有三大方面:

(1)为了数据类型重新定义，存储器容量和寄存器个数不受限制;

(2)表达形式分层定义，适合控制流分析，易于逐层抽象地还原高级语言的程序架构;

(3)为了重建自定义函数和结构，中间语言必须具有 C 语言的基本结构特征。

在结构化分析汇编语言与 C 语言层次上的异同后，得到具体的 AIL 的各层结构定义如下：

第一层：AIL 各分程序体结构形式=<预定义><主函数><函数><程序结束标志>

AIL 各分程序体结构组成::（预定义）{(主函数)}∪{（函数）n}　(n=0,1,2,…)

第二层：AIL 函数体结构形式=<函数名><函数体><函数结束标志>

AIL 函数体结构组成::=∪{（指令块）n}(n=0,1,2,…)

第三层：指令块结构形式=<语句群>

指令块结构组成::∪{(AIL 自定义语句)n}(n=0, 1, 2,…)

第四层：AIL 自定义语句结构形式=<表达式群>

AIL 自定义语句结构组成::∪{(表达式语句)n |（赋值语句）n |（判断语句）n |（跳转语句）n |（函数调用语句）n |（返回语句）n |（条件语句）n…}(n=0, 1, 2,…)

第五层：AIL 自定义表达式结构形式=<表达式名称>(<条件 1>，<条件 2>，<结果 1>，<结果 2>)

AIL 自定义表达式结构组成::∪{(一元表达式)n |(二元表达式)n |(三元元表达式)n |(四元表达式)n }(n=0, 1, 2,…)

第六层：AIL 自定义项值结构形式=<项值名称>(项值数)

AIL 自定义项值结构组成::∪{(静态常量)n |(变量)n |(操作符)n }(n= 0, 1, 2,…)

1.2.2 AIL 的表达因子定义 AIL 的优越性

分类、归纳、生成数据类型的信息过程是 AIL 语言建构的前提条件，分类、归纳、生成结构化控制流程的信息过程是 AIL 抽象表示汇编语言的核心步骤[6]。为了更加准确的描述结构化算法,定义出 AIL 的表达因子。每个表达因子对应了 C 语言中的数据定义符和操作函数，但又兼有汇编语言的动作特征，这样的设计使到中间语言能用 C 语言的流程表示出汇编语言的执行过程，而又避免以往逆编译过程中过多的手动指定地址跳转的繁琐，以下是表达因子定义:

CONST(i) 所有的值常量 等同于 C 语言中的 const type (i)

NAME(n) 所有符号常量 等同于 C 语言中的 name type(n)

TEMP(t) 标志临时变量（相当于 CPU 中的寄存器，但中间语言可以抽象出数量级别更多的寄存器）

OP(o)操作符变量 重载操作符

BINOP(o,a,b) 对操作数a,b施加二元操作符o表示的操作,a 的计算优先于 b (担负了操作的转换功能)

MEN(n) 存储器地址 n

CALL(f, n) 以参数 n 调用函数 f

ESEQ(a, b) 计算 a 对 b 的作用

MOVE(TEMP a, b) 计算机 b 送入临时单元 a

MOVE(MEM(e1),e2) 计算 e1，生成地址 a，计算 e2，送入地址 a 开始的单元（担负了转换大多数的寄存器操作）

EXP(a) 计算 a

GOTO(a, l) 跳转到地址 a，目标地址可以是文字标号(例如 NAME 表示的标号,也可以是其他种类的表达式计算出来的一个地址,l 指出表达式 a 可能计算出的所有目标地址)

CJUMP(o,a,b,n1,n2) 二元操作符跳转，先计算 a,b 的生成值 x,y，然后用操作符 O 比较 a,b,为真跳转到 n1,为假跳转到 n2（用于表示分析出来 if-else, switch 等条件选择语句）

NEXT(a,b) 语句定义关系，a 语句之后为 b 语句,a 可以为 next 值（执行下一条语句）

LABEL(n) 定义当前机器代码地址 n

WHITE(N) N 可以为语句或者符号 CONST(i) 、NAME(n) 、TEMP(t)等（为了避免转换系统的膨胀，不增设 FOR 语句，可以根据分析结果进行转换）

RETURN 等价于高级语言的 return

1.3 AIL 的优越性

通过 AIL 的定义可以看出，该语言具有很多特点:数据类型定义默认缺省，控制结构基于 C 语言但是更加简练，语句表达类似于汇编语言但又使用了诸如函数名、比较条件语句等许多 C 语言的语句特征。逆编译中控制流程恢复是导致各种困难的直接原因，AIL 针对控制流程恢复的困难使用了从汇编语言代码中提取各种特征信息，各特征信息可以互相对比表示，通过对比结果进行控制流程恢复的方法。

2 汇编语言到中间语言转换算法

2.1 转换算法中的难点和识别模块举例

一直以来，用户自定义函数或结构体的识别是逆编译中的难题，究其原因可能是多方面的，但主要还是与用户自定义函数或结构体的一些语言的必须特征如自定义函数或结构体的名称和自定义参数类型等不易得到有关[7]。通过大量实例分析与论证，AIL 将识别过程分为四个阶段并依此设计出四个分析器。四个阶段的流程如图 2：

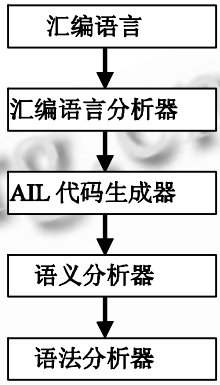

图 2 AIL 四个阶段分析流程图

以上四个阶段具体分工如下：

(1)汇编语言分析器，作用是分析出语言子模块；

(2) AIL生成器将自定义函数或结构体的代码翻译为可以逆编译的中间语言程序，然后恢复出相应的自定义函数或结构体的名称、参数个数及类型等。

(3) AIL语义分析器根据这些恢复的信息按照系统库函数或标准结构体的汇编语言模块来构造自定义函数或结构体的识别模板，将其转换为 AIL 的组织形式并进行标示。

(4)语法分析器利用一些基于汇编语言结构形式的函数体模板完善整个 AIL 的表达。函数体模板的建立是，比如下面识别流程中使用到的 for 和 while 等循环语句的模板：

for 的模板：

```
mov <循环变量>,<初始值> ;      //给循环变量赋初值  jmp B ; //跳到第一次循环处
A:(改动循环变量);   //修改循环变量
B: cmp <循环变量>,<条件变量> ;    //检查循环条件  jgp 跳出循环
(循环体)
jmp A ;      //跳回去修改循环变量
```

while 的模板：

```
A: cmp <循环变量>,<条件变量>      //检查
```

```
循环条件                                    jge B
(循环体)
jmp A;    //跳回去修改循环变量
B:                                          //循环结束
```

2.2 算法实现

2.2.1 逆编译环境

在 windows XP 系统环境下，使用 VC++6.0，我们自定义一个用户函数，使用了结构、数组、字符指针等比较难以识别的数据类型：

```
typedef struct {
int x;
int y;
int z;
} ailStruct;
int ailFunction(int a,int b)
{
char *buf[100];
ailStruct *strs = (ailStruct *)buf;
int i;
for(i=0;i<10;i++)
{strs[i].x = 1;
strs[i].y = 2;
strs[i].z = 3;
}return 0;
}
```

2.2.2 汇编语言实现

抽取自定义函数中最关键的实现部分：

```
int i;
for(i=0;i<10;i++)
{strs[i].x = 1;
strs[i].y = 2;
strs[i].z = 3;
}
```

使用 IDA pro4.8 逆编译这一段之后，相对应得到的汇编代码如下

```
0041367A mov dword ptr [i],0 ;
00413684    jmp    myfunction+45h
(413695h);
00413686 mov eax,dword ptr [i];
0041368C add eax,1;
0041368F mov dword ptr [i],eax;
00413695 cmp dword ptr [i],10;
0041369C jge myfunction+94h (4136E4h);
0041369E mov eax,dword ptr [i] ;
004136A4 imul eax,eax,0Ch ;
004136A7 mov ecx,dword ptr [strs] ;
004136AD mov dword ptr [ecx+eax],1 ;
004136B4 mov eax,dword ptr [i]
004136BA imul eax,eax,0Ch;
004136BD mov ecx,dword ptr [strs];
004136C3 mov dword ptr [ecx+eax+4],2 ;
004136CB mov eax,dword ptr [i];
004136D1 imul eax,eax,0Ch;
004136D4 mov ecx,dword ptr [strs];
004136DA mov dword ptr [ecx+eax+8],3;
004136E2    jmp    myfunction+36h
(413686h) ;
004136E4 xor eax,eax ;
```

2.2.3 使用汇编语言分析器

```
//注释文字为汇编语言分析器分析结果：
{0041367A mov dword ptr [i],0 ;
00413684    jmp    myfunction+45h
(413695h);
00413686 mov eax,dword ptr [i];
0041368C add eax,1;
0041368F mov dword ptr [i],eax;
00413695 cmp dword ptr [i],10;
0041369C jge myfunction+94h (4136E4h);
}  //分析出一个 for 循环
{0041369E mov eax,dword ptr [i] ;
004136A4 imul eax,eax,0Ch ;
004136A7 mov ecx,dword ptr [strs] ;
004136AD mov dword ptr [ecx+eax],1 ;
}  //分析出一个结构体
{004136B4 mov eax,dword ptr [i]
004136BA imul eax,eax,0Ch;
004136BD mov ecx,dword ptr [strs];
004136C3 mov dword ptr [ecx+eax+4],2;
}  //分析出一个结构体
{004136CB mov eax,dword ptr [i];
004136D1 imul eax,eax,0Ch;
```

```
004136D4 mov ecx,dword ptr [strs];
004136DA mov dword ptr [ecx+eax+8],3;
} //分析出一个结构体
004136E2 jmp myfunction+36h (413686h); // 循环结束
004136E4 xor eax,eax ; // ;eax 清 0
```

2.2.4 使用 AIL 生成器生成和语义分析器

```
CONST(i);
i=0;
WHITE(i<5);
{LABEL(a)
MOVE(ptr [i], ecx+eax)
CJUMP(<,ptr [i],5, GOTO(a, labs), NEXT (next，0))
MOVE(ptr [i], ecx+eax+4)
CJUMP(<,ptr [i],5, GOTO(a, labs), NEXT (next，0))
MOVE(ptr [i], ecx+eax+8)
CJUMP(<,ptr [i],5, GOTO(a, labs), NEXT (next，0))
GOTO(myfunction+36h, 413686h)
} //本段循环识别为 for 结构
```

2.2.5 使用语法分析器

语法分析器将 AIL 代码转化为类 C 语言的 for 循环输出：

```
CONST i=0;
for(i=0;i<10;i++)
{strs[i].a = 1;
strs[i].b = 2;
strs[i].c = 3;
}
return 0;}
```

3 结束语

目前适合于对逆编译结果程序进行框架分析的工具很少，特别是缺少能够适用于大多数主流处理器逆编译结果的通用型的框架分析工具。但由上可见，使用 AIL 中间语言表示之后的中间代码，能够比汇编代码更能有效的标识和还原到原先高级代码的控制结构，而又保留了汇编代码的风格模式，避免了其他的中间语言因为保留了类 C 语言的数组、结构体等复杂数据类型所导致的各平台通用性降低，从而提升了逆编译的重现完整性和识别率。

使用 AIL 中间语言的逆编译系统已初步实现。如上面实例所示，重点需要加强的是高级语言的控制结构之间的替代性研究。工程证明在逆编译项目中使用 AIL 具有如下优势：(1)能减少逆编译工程中对不同的机器语言和系统环境的依赖性；(2)使原本没有明确分层的逆编译过程变成结构化的、分步归纳的、逐层抽象的过程；(3)工程人员可以分层地对逆编译过程进行干预，减少最后由人工处理部分的难度和工作量。

参考文献

1 Horwitz S. Identifying the Semantic and Textual Differences Between Two Version of a Program. Proc. of PLDI'90. New York, USA: ACM Press, 1990:234－245.

2 Emmerik MV, Waddinton T. Using a Decompiler for Real-world Source Recovery Proc. of WCRE'04.Delft, Netherlands: IEEE Computer Society Press, 2004:27－36.

3 杨克娇.反编译中处理分支指令的关键技术研究与实现[硕士学位论文].郑州:解放军信息工程大学, 2007.

4 陈凯明.逆编译中几项关键技术研究[博士学位论文].合肥:合肥工业大学,2004.

5 熊文新,袁琦.中间语言转换过程中的增强处理. 计算机工程与应用, 2005,(9):171－172.

6 陈凯明,刘宗田.符号执行过程的 DFA 和 CFA.计算机工程, 2002,28(11):95－96.

7 许向阳,雷涛,朱虹.反编译中的静态库识别研究.计算机工程与应用, 2004,9:37－38.