文章编号:1001 - 9081(2006)08 - 1925 - 03

一种基于 Re-Engineering 的自动翻译方法研究

张红艳[1,2],李 森[1]
(1. 中国科学院 合肥智能机械研究所,安徽 合肥 230031;
2. 中国科学技术大学 信息科学技术学院,安徽 合肥 230027)
(muyuzio@mail.ustc.edu.cn)

摘 要:分析了结合 Re-Engineering 技术的翻译器 Bogart 的工作原理,并与传统翻译器进行了比较,同时对源程序的存储方法提出了改进,采用了新的支持程序语言定义的语言处理系统。测试结果表明,该方法能够提高自动翻译的效率。

关键词:Re-Engineering;自动翻译器;抽象模型
中图分类号:TP18;TP391.2 **文献标识码**:A

Research of automatic translation based on Re-Engineering

ZHANG Hong-yan[1,2], LI Miao[1]
(1. *Institute of Intelligent Machines, Chinese Academy of Sciences, Hefei Anhui* 230031, *China*;
2. *School of Information Science and Technology, University of Science and Technology of China, Hefei Anhui* 230027, *China*)

Abstract: By analyzing the operating principle of the translator Bogart which integrates Re-Engineering technique and comparing it with the traditional translator, an improved storage method was suggested. This method adopted a new language processing system which supported the definition of program language. The test results indicate that the method can significantly improve the efficiency of automatic translation.

Key words: Re-Engineering; automatic translator; abstract model

目前,许多地方需要把旧代码移植到新的体系结构中,以便在不同的结构下运行。传统的代码翻译有两种方法,一种是通过逐行翻译结合手工修正,旨在从源代码产生语义相同的目标代码。但它忽略了其他方面,如提高可读性、易维护性、可复用性等,并且它只能用于语法相似的语言,如 Fortran 与 C。另一种采用概括与重新实现[1]的方式,据说可以满足上述目标,但是由于程序的复杂性和实现成本高,该方法尚未得到应用。Re-Engineering[2]则先对源代码进行分解及了解的过程,然后再由 Reverse Engineered Systems 形成目标代码[3]。图 1 以 Bogart 为例,表示结合了Re-Engineering技术的翻译器的工作流程。

1 Bogart 的工作原理

如图 1 所示,Bogart 的整个过程主要有解析、Re-Engineering 和翻译三个部分。

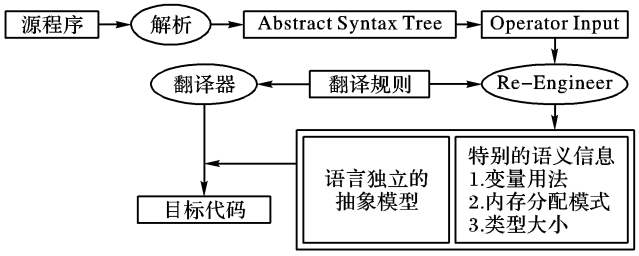

图 1 结合 Re-Engineering 翻译器 Bogart 的原理

1)解析。因为程序是以文本形式存储的,所以要对源程序进行编辑、字符搜索等完成一棵抽象语法树[4],它作为程序的中间表示形式,可以遍历此 AST 得到语义信息而不需要再解析源代码。

2)Re-Engineering。除了满足正确性外,翻译同样需要达到其他要求,而 Re-Engineering 满足了翻译要求的灵活性,可以完成许多任务,包括程序语义搜集和分析、重构和产生抽象模型。程序语义搜集和分析指在 AST 和用户输入的有关设计知识库集合的基础上,分析源代码和搜集嵌入在源代码中的设计语义。程序语义包括依赖于语言和机器的信息,如变量的类型和大小、内存寻址方式等,独立于语言的语义包括函数、变量、相互调用关系等,语义搜集阶段的目的就是为生成抽象模型和翻译提供必需的语义信息。

这些搜集的语义信息帮助分析程序的数据和控制关系、数据共享等,作为程序分析的基本输入,在产生独立于语言的数据模型和控制流模型后,进一步生成专门的抽象模块作为翻译的框架。图 2 是翻译过程中 Re-Engineering 和翻译的关系图,其中 Re-Engineering 生成框架(抽象模型),翻译完成源语言与目标语言之间语义的映射。这样 Re-Engineering 把软件重新构造成一个有层次结构的包,详细的重构方法见文献[5]。该层次图能够准确地表示出一个子部分与其他部分的依赖关系,为翻译提供了方便的框图,更有助于像 C、Fortran 之类具有模块化、抽象数据类型语言的翻译,并且程序的层次结构表示更有利于系统的理解和分析。

3)翻译。由于源语言与目标语言间语言结构的语义不同,所以语义影射的关系既有一对一、多对一或一对多,我们的方法是基于语义形式匹配,它需要了解两种语言结构的语义,然后通过映射规则把源语言语义映射到目标语言语义。翻译器是基于规则的系统,只有当语义形式匹配时采用,每个

收稿日期:2006 - 03 - 06;**修订日期**:2006 - 05 - 10 **基金项目**:国家 863 计划资助项目(2003AA118040)

作者简介:张红艳(1982 -),女,湖北钟祥人,硕士研究生,主要研究方向:模式识别与智能系统; 李森(1955 -),女,安徽庐江人,研究员,主要研究方向:人工智能、农业知识工程.

翻译规则需要前提、后备、输入语义信息、输出语义信息、翻译规则、预翻译规则和后备翻译规则。

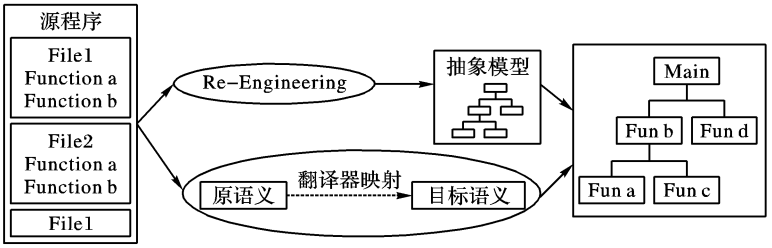

图2 Re-Engineering 与翻译器的关系

2 传统翻译器与 Bogart 的比较

汇编源程序：

```
HORNER CSECT
   STM R14, R12, 12 (R13)
   LR R12, R15
   USING *, R12
* * * * * * * * * INITIALIZATION * * * * * * * * *
    LA R7, COEF
    L R5, 0 (R7) SUM = A0
    LA R9, 0 I = 0
* * * * * * * * * TEST FOR EXIT * * * * * * * * *
LOOP CR R9 , R2
     BNL OUT
* * * * * * * * ADJUSTMENT STEP * * * * * * *
    LA R9, 1 (R9)
    LA R7,4 (R7)
NEXT COEFFICIENT
* * * * * * * * * BODY OF LOOP * * * * * * * * * *
    MR R4, R3 SUM * X
    A R5, 0 (R7)
    SUM = SUM * X + AI
    B LOOP
OUT LR R0, R5
    LM R1, R12,24 (R13)
    BR R14
COEF DS 10F
END
```

传统翻译器生成 C 代码：

```
void HORNER( tagSAPReg * Reg)
 {
   T_stm(14, 12, ((Reg[13].ucp +12)), Reg);
   Reg[12].sw = Reg[15].sw;
   Reg[7].pv = &(COEF[0]);
   Reg[5].sw = *(sWord *) Reg[7].ucp;
   Reg[9].sw = 0;
 LOOP:
   if ((Reg[9].sw) > =Reg[2].sw) goto OUT;
   Reg[9].sw + = 1;
   Reg[7].sw + = 4;
   T_mult(&Reg[4], Reg[3].sw);
   Reg[5].sw + = *((sWord *)(Reg[7].ucp));
   goto LOOP;
 OUT:
   Reg[0].sw = Reg[5].sw;
   T_lm(1, 12, ((Reg[13], ucp +24)), Reg);
   return;
 }
```

从上面看出，传统翻译方法采用数组联合类型表示汇编寄存器，并且逐行把汇编指令翻译成一致的 C 表达式。

Bogart 生成的 C 代码：

```
sWord HORNER( sWord r2sw, sWord r3sw)
    {
      sWord r5sw;
      sWordPtr r7swp;
      sWord r9sw;
      r7swp = (sWord *)(&COEF[0]);
      r5sw = *r7swp;
      r9sw = 0;
      while (r9sw < r2sw)
        {
          r9sw ++;
          r7swp = r7swp +4;
          r5sw = r5sw * r3sw + *r7swp;
        }
      return r5sw;
    }
```

与上段比较看出，Bogart 生成了更简单有效的代码，可以从以下方面说明：

1）去掉多余的代码。一些汇编指令不需要在 HLL 中出现，如开始及最后的寄存器保存等。

2）组合表达式。汇编语言不支持表达式组合，因此传统翻译器也不支持，但是由于 Bogart 的抽象模型完成了数据流分析，它就可以正确地把几条指令合成一个表达式。

3）去掉无用的计算结果。有时候由于指令的原因而生成多个结果，有些结果后面根本没有用，传统翻译器对这些结果不分析，因此每步都需要执行翻译，而 Bogart 可以分析结果有没有用，大大简化了翻译。

4）类型分析。传统的翻译器依赖类型转换，经常有类型不匹配错误发生，而 Bogart 可以完成一些简单的类型分析，这样就产生了可读性更高的代码。

3 对 Bogart 的改进及其实现

在解析建立 AST 的过程中，不难发现采用字符搜索不支持目标语言定义的正规表达式，也不支持程序整体的自动翻译，因此在 Bogart 的基础上，我们提出了一种更好的分析和翻译的方法，从以下两方面进行改进：

1）对源程序建立面向对象数据库替代文本格式。该数据库保证了程序及其相关对象以符号抽象语法树形式的一致性存储，并且可以保证多用户的并发性，此数据库被集成在语言处理系统中。

2）支持程序语言定义的语言处理系统，它用语言的正则文法（即形式描述语言）作为输入。正则文法是用高级语言（如 C 语言）写的，它包括产生式右边的操作符、优先表、产生式的语义动作，这样就减少了递归产生式的需求，增加了语法说明的可读性，它的关键部分是一个 LALR（1）解析产生式。另外它还提供了定义程序模板的能力，这些模板可用做形式匹配，测试一个程序是否是一个模板的实例，解析程序模板的产生式规则自动地由语言处理系统添加到用户定义的语法中。

图 3 是改进后的 Bogart 组件关系图。无论从空间和时间上来说，Re-Engineering 翻译器都产生更有效的代码。因为采用抽象模型表示，精化了翻译过程，Re-Engineering 翻译器生成的代码只有传统翻译器的 1/2 或 3/4 大，速度是它的 2 倍，改进后的翻译器空间上没有多大减少，但是时间还是缩短到 Re-Engineering 翻译器的 3/4 左右。表 1 是改进后的 Bogart、Bogart 与传统翻译器一起翻译三个小程序的结果比较，证明

了本文对 Bogart 的改进是成功的,特别是在速度方面。

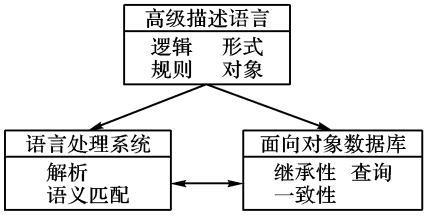

图 3　改进后的 Bogart 组件关系

表 1　翻译程序的结果比较

	传统翻译器		Bogart		改进的 Bogart	
	时间/s	空间/byte	时间/s	空间/byte	时间/s	空间/byte
Bin	63	4 170	33	2 802	23	2 798
Horner	10	3 302	3	2 465	2	2 402
Random	9	5 447	4	2 741	3	2 636
SAPDBMS	18	41 700	9	29 043	6	28 741

4　结语

介绍了把 Re-Engineering 结合到自动翻译器中的方法,目的在于提高目标代码的质量。从前面的描述可以看到,应用Re-Engineering的翻译器可以使代码达到一般的正确性、可读性、易维护性和可复用性等特点,但是它需要建立一个好的抽象模型,并且抽象模型的再实现本身也是很复杂的过程,因此 Re-Engineering 翻译器的功能尚不完善。在此基础上,我们提出了两个改进方案,并通过测试证明了改进的正确性。今后的主要工作是以 Bogart 为蓝本,设计出效率更高的 Re-Engineering 翻译器,使其达到实用化的程度。

参考文献:

[1] WATERS RC. Program Translation via Abstraction and Reimplementation[J]. IEEE Trans Software Engineering, 1988, 14(8): 1207 - 1228.

[2] CHU CW, PATEL S. A Re-engineering Tool for the Reuse of Large Scale Software Systems[A]. Proceedings of The Fifth International Conference on Software Engineering and Knowledge Engineering [C]. 1993. 94 - 101.

[3] BLAIR A. A methodology for the design of knowledge based decision support systems[A]. Proceedings of International Conference on Expert Systems for Development[C]. 1994. 18 - 23.

[4] WILLS LM. Automated program recognition by graph parsing[R]. Technical Report 1358, MIT Artificial Intelligence Lab, 1992.

[5] CHU CW, PATEL S. Software Restructuring By Enforcing Localization and Information Hiding[A]. Proceedings of The International Conference on Software Maintenance[C]. 1992. 165 - 172.

(上接第 1921 页)

3　实验结果

实验环境是一台拥有四个 Intel Itanium2 1.3G 处理器和内存为 2.0G 的服务器,操作系统内核是 linux2.4.21-4.EL,所用的编译器是 ICC 8.0,对本课题测试组提供的测试集中的所有经 ICC 编译后生成的汇编代码中出现 br.ctop 指令的例子进行测试。实验中,首先使用 ICC 编译器对源程序编译成汇编代码,然后运用我们所提出的反流水算法对出现的软件流水循环代码转换成串行代码,再将该汇编程序汇编成可执行文件,最后在 IA-64 上运行此可执行文件以验证该算法的正确性。实验结果完全正确。表 1 列出了软件流水和反流水后循环体中指令束个数和通用寄存器个数的对比结果。

表 1　实验结果

汇编程序	软件流水循环体		反软件流水后循环体	
	指令束个数	分配通用旋转寄存器个数	指令束个数	使用通用寄存器个数
10_16_BitCardStructure.s	8	16	11	11
10_20_Mask_Bits.s	2	8	3	3
10_3_Card_Shuffle.s	7	16	12	10
10_28_Array_Modify.s	2	8	2	4
10_30_Surveydata_Analysis.s (1)	1	8	2	3
10_30_Surveydata_Analysis.s (2)	4	8	7	4

4　结语

软件流水作为一种先进的编译技术,对于提高指令执行的并行性有重要的作用,但也给逆向工程带来了困难,该论文描述的反流水算法,不仅可消除代码的硬件特性,还可以减少循环对寄存器的需求,降低寄存器压力,同时也提高了原代码的可读性。然而对于 while 循环,其难点在于其循环次数受到条件的限制,并且我们在研究过程中发现一些程序中在循环体外会使用循环体内流水后的寄存器。下一步主要工作将针对此类循环研究其反流水算法。

参考文献:

[1] LAM M. Software Pipelining: An Effective Scheduling Technique for VLIW Machines[A]. Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation[C]. Atlanta, Georgia, United States, 1988. 318 - 328.

[2] HUFF RA. Lifetime-Sensitive modulo scheduling[A]. Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation[C]. New York: ACM Press, 1993. 258. 267.

[3] MOSBERGER D, ERANIAN S. IA-64 Linux Kernel Design and Implementation[M]. Beijing: Tsinghua University Press, 2004.

[4] TANG ZZ, LI WL, SU BO. A de-pipeline algorithm for software-pipeline[J]. Journal of Software, 2004, 15(7): 987 - 993.

[5] EVANS JS, TRIMPER GL. Itanium Architecture for Programmers: Understanding 64-Bit Processors and EPIC Principles[M]. Beijing: Tsinghua University Press, 2004.

[6] LADSARIA A. Modulo Scheduling[EB/OL]. http://www.crhc.uiuc.edu/ece411/sp02/Slides02/modulo.pdf, 2002 - 05 - 05.

[7] Intel IA-64 Architecture Software Developer's Manual——Overview of Volume 3: Instruction Set Reference[EB/OL]. http://developer.intel.com/design/litcentr, 2000 - 01.

[8] Intel Itanium Architecture Software Developer's Manuall—— Volume 1: Instruction Set Reference[EB/OL]. http://www.intel.com/design/itanium/manuals, 2002 - 10.

[9] MUCHNICK SS. Advanced Compiler Design Implementation[M]. Beijing: China Machine Press, 2003.