消除 GCC 抽象语法树文本中冗余信息的算法研究*)

李 鑫 王甜甜 苏小红 马培军
(哈尔滨工业大学计算机科学与技术系 哈尔滨 150001)

摘 要 由GCC编译器对C语言源程序进行语法分析产生的抽象语法树文本存在大量的冗余信息,如果直接对其进行解析,则会产生解析效率低、产生的抽象语法树会占用大量的存储空间的问题。针对此问题,在深入研究GCC抽象语法树文本结构和解析过程的基础上,提出了一种高效消除冗余的算法,通过实验证明了算法的正确性和适用性,并提出了GCC抽象语法树解析的数学定义。

关键词 抽象语法树(AST),抽象语法树文本,抽象语法树的解析,规范化的抽象语法树文本,冗余

Research on Eliminating Redundancies of GCC AST Text

LI Xin WANG Tian-tian SU Xiao-hong MA Pei-jun
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

Abstract There exist a lot of redundancies in AST text produced by GCC complier in the process of syntax analysis to a C program. If AST text is parsed directly, it will lead to a bad result that the efficiency of the parsing process is low and lots of memory is used by AST. In order to solve this problem, a highly efficient algorithm is put forward to eliminate redundancies based on a thorough research on the structure of GCC AST text and the parsing process. The correctness and practicality were proved by experiments and the mathematical definition of AST parsing was proposed.

Keywords Abstract syntax tree, AST text, AST parsing, Canonical AST text, Redundancy

1 引言

1.1 概念提出

GCC抽象语法树的解析是近几年兴起的研究课题,国内外对此课题都有一些研究,但存在概念不明确的问题,所以对如下几个概念进行明确的定义,以便于以后问题的描述。

定义 1(抽象语法树文本 gcc_astTxT) 抽象语法树的文本描述,即原文件经过GCC编译后产生的(.tu)文件。

定义 2(规范化的抽象语法树文本 canonical_astTxT) 即经过冗余消除的抽象语法树文本。

定义 3(抽象语法树 AST) 抽象语法树文本经解析后产生的某种数据结构。

定义 4(抽象语法树的解析) 设 σ_1,σ_2 是定义在抽象语法树文本集合 Ω 两个变换,gcc_astTxT,canonical_astTxT $\in \Omega$,AST $\in$ 抽象语法树集合 Φ,其中:

σ_1(gcc_astTxT)= canonical_astTxT,称为冗余消除变换;

σ_2(canonical_astTxT)=AST,称为重建变换。

令 $\theta=\sigma_1\circ\sigma_2$,则称 θ 是对 gcc_astTxT 的解析。

定义 5(固有冗余)由于文件包含产生的冗余,即 # include 命令产生的与原文件不相关的函数及其相关信息。

定义 6(系统冗余) 在编译过程中,由编译器产生的冗余,主要是一些内部函数以及与其相关的变量声明、类型声明、常量等。

定义 7(字段冗余)描述字段中与分析数据流、控制流无关的字段及其指向结点已被消除的字段。

1.2 抽象语法树文本简介

GCC编译系统以源程序的过程为单位生成抽象语法树,抽象语法树与过程一一对应。使用GCC的编译参数-fdump-translation-unit可以使源程序生成名为 *.tu 的文件,其中 * 为源程序名。该文件就是抽象语法树文本。其中的一个结点的结构如图1所示。此文件由若干个这样的结点组成。结点的类型有以下几种:

(1)标识符结点(Identifiers)
(2)类型结点(Types)
(3)声明结点(Declarations)
(4)函数结点(Functions)
(5)范围结点(Scope)
(6)语句结点(Statements)
(7)表达式结点(Expressions)

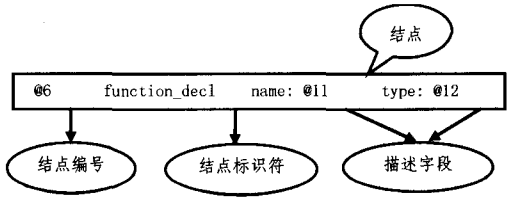

图1 抽象语法树结点的结构

1.3 研究现状及存在的问题

由GCC产生的抽象语法树文本并不很适合做代码分析,

*)国家自然科学基金(No. 60373000)。**李 鑫** 硕士生,研究方向为编译技术、程序识别、程序理解;**王甜甜** 博士生,研究方向为程序识别、程序理解、编程题自动评分;**苏小红** 博士,教授,博士生导师,研究方向为软件工程、图像信息处理、信息融合;**马培军** 博士,教授,博士生导师,研究方向为软件工程、图像信息处理、信息融合。

编译器的目的是将高级语言转化为汇编代码。故而，GCC 产生的抽象语法树文本中包含许多有助于编译的细节信息，例如由 # include 命令产生的未被源程序用到的函数及结构，以及编译过程中产生的一些内部函数、类型声明、出错信息、常量等，但这些信息不利于代码分析。在数量上，对一个很小的编译单位，大概能产生其 1000 倍的抽象语法树文本，最终产生的抽象语法树会占据整个内存。

现有解决方法：

(1)使用基于事件的分析器，使得抽象语法树不在内存中建立。

(2)根据不同的任务过滤掉不必要的信息。

国内外对此问题的研究不多，在文献[7]中提到了这一问题但没有具体的算法。国内对 GCC 语法树的解析方法没有涉及消除冗余这一问题，绝大部分方法[2,4,6]是直接对抽象语法树进行解析，因此研究此问题就显得尤为重要。

2 冗余消除算法的基本思想

2.1 算法达到的目标

为了达到引入 GCC 改善评分系统[1]性能的目的，算法达到目标是：消除抽象语法树文本中所有与分析数据流、控制流无关的信息并保持信息的完整性(不能丢失有用信息)。将结点分为三种类型：

(1)有用结点(useful_Node)：与分析程序数据流、控制流相关的结点。

(2)无用结点(useless_Node)：与分析程序数据流、控制流不无关的结点。

(3)待定结点(unknow_Node)：当前类型还未被确定的结点。

2.2 算法的基本思想

Step1 根据 srcp(来源)字段的值将所有结点分为三类：

(1)if Node. srcp＝源文件名，then 此结点被标示为 useful_Node；

(2)if Node. srcp＝其他值，then 此结点被标示为 useless_Node；

(3)if Node 不含有 scrp 字段，then 此结点暂时为 unknow_Node。

Step2 对 unknow_Node 的子树中的结点(简写为 SubNode)进行分类(是一个类似图宽度遍历的过程)：

(1)if (Node is useful_node) && (SubNode is unknow_Node)，then SubNode 被标识为 useful_node；

(2)if (Node is useful_Node) && (SubNode is useless_Node)，then SubNode 被标为 useless_node；

(3)if(Node is useless_Node) && (SubNode is unknow_Node)，then SubNode 被标识为 useless_Node；

(4)if(Node is useless_Node) && (SubNode is useful_Node)，then SubNode 被标识为 useful_Node；

(5) 直到 unknow_Node 的个数为零。

Step3 由于(1)、(2)步将所有的库函数、内部函数及其相关信息都删除了，还要找回源文件用到的库函数。主要是检查 call_expr(调用表达式)的 Sub_Node。

2.3 算法的详细描述

输入：gcc 产生的抽象语法树文本(. tu 文件)

输出：规范化的文本抽象语法树文本。

算法过程：

声明：int flag[n]＝2；//n 是总行数

int number；//存放编号

CString CResult＝""；//存放最后结果

(1)对 gcc_txtAST 进行格式化，使得描述同一的主结点的从结点在同一行上；

(2)将整个 gcc_txtAST 存储在一个字符串数组中；

(3)for i：＝1 to n (总行数) do

(4)if 第 i 行含有子串"scrp" then

(5)if scrp：字段的值＝文件名

then flag[i]：＝1；

(6)else flag[i]：＝0；

(7)else flag[2]：＝2；

(8)for j：＝1 to n do

(9) if flag[j]！＝2 then

(10)begin

(11)number：＝找到它子结点的编号；

(12)while 本行不结束 do

(13)begin

(14) if(flag[j]＝1&&flag[number]！＝0)

then flag[number]：＝1；

(15)else

if(flag[j]＝0&&flag[number]！＝1)

then flag[number]：＝0；

(16) number：＝找到下一子结点的编号；

(17)end；

(18)end；

(19)找回源程序中用到的库函数的信息；

(20)for k：＝1 to n do

(21) if flag[k]＝1 then

(22) 将第 k 行连入 CResult 并写入一名为 result. txt 的文件中；

(23)return CResult；

(24)算法结束。

2.4 算法复杂性分析

(1)算法耗时：一是在寻找子串"scrp"上，如采用 kmp 算法，算法的时间复杂度为 O((max(*length*(所有行))＋5) * *n*)；

(2)二是对每个非待定结点为根的子树的遍历，算法的时间复杂度为 O(max(*length*(所有行))^2 * *n*)；

综上，算法的时间复杂度为 O(max(所有行)2 * *n*)；

算法占用的空间主要是一个字符串数组。

3 实验分析

在 vc＋＋6. 0 环境中实现以上算法，实验程序取自文献[3]，实验结果如图 2。

实验程序	源程序代码行数	消除冗余前后结点数对比	信息是否保持完整
计算连续 100 个自然数之和	7	1280：76	是
学生的百分制成绩向 5 分制转化	14	1374：125	是
二分查找	15	1522：147	是
矩阵乘法	55	2022：202	是

图 2 冗余消除的实验结果

从图 2 可以看出，总体上效果达到了预计的目标，较好地删除了抽象语法树文本中的冗余信息。图 3 和图 4 是冗余消除前的抽象语法树文本与冗余消除后的抽象语法树文本的一个对照，以计算连续 100 个自然数的和为例取前 10 个结点。

```
@1 namespace_decl name: @2  srcp: <internal>:0 dcls: @3
@2 identifier_node strg: :: lngt: 2
@3 function_decl   name: @4  type: @5  srcp: ex1.c:3
@4 identifier_node  strg: main     lngt: 4
@5 function_type    size: @8     algn: 64     retn: @9
@6 function_decl    name: @11  type: @12    chan: @13
@7 compound_stmt    line: 8    body: @14    next: @15
@8 integer_cst      type: @16  low : 64
@9 integer_type     name: @17  size: @18   algn: 32
@10     tree_list         valu: @21
```

图 3　消除冗余前的抽象语法树文本

```
@3   function_decl     name: @4   type: @5   extern          body: @7
@4   identifier_node   strg: main     lngt: 4
@5   function_type  size:@8   algn:64   retn: @9    prms: @10
@7   compound_stmt     line: 8          body: @14       next: @15
@8   integer_cst       type: @16        low : 64
@9   integer_type      name: @17        size: @18       algn: 32
@10   tree_list        valu: @21
```

图 4　冗余消除后的抽象语法树文本

结束语　实验表明，本文算法达到了很好的效果并且具有较低的时间和空间复杂度。作为改善评分系统前端的重要一步，现已用到评分系统中，取得了良好的效果。与传统的没有冗余消除解析方法相比，消除冗余后的抽象语法树具有更好的实用性。再进行进一步处理，可以非常方便地应用于对程序分析及其它领域。

参 考 文 献

[1] 王甜甜. 基于语义相似度的编程题自动评分系统研究与应用. 硕士论文. 哈尔滨工业大学，2005：1-53

[2] 石峰，等. 一种解析 GCC 抽象语法树的方法[J]. 计算机应用，2004，24(3)：115-116

[3] 苏小红，陈惠鹏，孙志岗，等. C 语言大学实用教程[M]. 电子工业出版社，2004，8：231-374

[4] 刘文伟，等. 一个重建 GCC 抽象语法树的方法[J]. 计算机工程与应用，2004，8：125-128

[5] Power J F. Program annotation in XML：a parse-tree based approach[C]// Proceedings of the Ninth Working Conference on Reverse Engineering. 2002：1095-1350

[6] 王相懂，等. 基于 gcc 抽象语法树对 c++源程序结构的分析[J]. 计算机工程与应用，2006(6)：97-100

[7] Antoniol G. XML-Oriented gcc AST Analysis and Transformations[C]// Proceedings of the Third IEEE International Workshop on Source Code Analysis and Manipulation. 2005，7：869-901

(上接第 133 页)

可以测试大于 1G 的文件，说明 OBS 在大文件支持方面的能力有所提高。

4.2　文件操作性能

先创建两个文件集 a 和 b，每个文件集共有 32768 个文件，10 个目录。文件集 a 的文件大小在 80B 到 120B。而文件集 b 的文件大小在 800B 到 1200B，测试创建、读和删除的 I/O 性能，结果如表 1 和表 2 示。从表中可以看出，OBS 在创建、读和删除大文件方面的性能比 Ext2 要强得多，而且，随着文件大小的增加，Ext2 的速度下降得很快，特别是顺序读操作，而 OBS 的速度则保持稳定，几乎没有太大的起伏。从 CPU 的占用率来看，随着文件系统容量的增加，OBS 的 CPU 占用率有所升高，这是因为使用了复杂的搜索算法和数据结构。

表 1　文件集 a 的文件操作性能

文件系统	文件数 最大尺寸 最小尺寸 目录数	顺序创建		读		删除		随机创建		读		删除	
		k/sec	%cpu	k/sec	%cpu	k/sec	%cpu	k/sec	%cpu	k/sec	%cpu	k/sec	%cpu
Ext2	32：120：80：10	7132	69%	10102	23%	32252	30%	7288	69%	none	none	9887	66%
OBS	32：120：80：10	6339	77%	12647	20%	8386	60%	6730	82%	558	1%	336	2%

表 2　文件集 b 的文件操作性能

文件系统	文件数 最大尺寸 最小尺寸 目录数	顺序创建		读		删除		随机创建		读		删除	
		k/sec	%cpu	k/sec	%cpu	k/sec	%cpu	k/sec	%cpu	k/sec	%cpu	k/sec	%cpu
OBS	32：1200：800：10	7647	74%	18631	25%	26135	27%	7670	72%	none	none	8939	55%
Ext2	32：1200：800：10	5045	66%	2225	7%	2386	19%	4242	57%	255	1%	171	1%

参 考 文 献

[1] Hitz D，Lau J，Malcolm M. File System Design for an NFS File Server Appliance// Anon，ed. Proceedings of USENIX 1994. San Francisco：Network Appliance，1994：103-105

[2] Schmuck F，Haskin R. GPFS：A Shared-disk File System for Large Computing Clusters// Darrell D E，ed. Proceedings of the Conference on File and Storage Technologies. Monterey：FAST02，2002：231-244

[3] Scltzer M. Analysis of extent allocation policies in file systems. Masters Report. EECS Dept.，University of California，Berkeley，Ca，1988：1-73

[4] Berchtold S，Keim D A，Kriegel H P. The X-Tree：An index structure for high-dimensional data// Anon，ed. Proceedings of the 22th International Conference on Very Large Data Bases India. 1996：28-39

[5] Johson T，Shasha D. The performance of concurrent B-tree algorithms. ACM Transaction on Database Systemns，1993，18(1)：51-101

[6] Bryant R，Forester S R，Hawkes J. Filesystem performance and scalability in linux 2. 4. 17// Anon，ed. Proceedings of the USENIX Annual Technical Confernce. CA：USENIX Association，2002：235-245