支持多程序语言的静态信息提取方法

逄 龙，王甜甜，苏小红，马培军

（哈尔滨工业大学 计算机科学与技术学院，150001 哈尔滨，hitpanglong@163.com）

摘 要：为了满足代码分析对多语言静态信息提取的需求，克服当前构建单语言提取重用率低、过程复杂等不足，采用直接修改GCC特定解析阶段源代码的方法建立统一的提取接口.针对所需静态信息的不同，按GCC内部机制，提出了运行改入点与内部辅助函数重用相结合的提取方法，具体包括类型和函数声明信息的采集、函数体内程序语句的遍历以及多语言统一中间表示的获取，重用了GCC内部高质量代码，从而降低了构建静态信息提取所需的重复开销.通过对比试验表明该方法程序语言解析能力稳定健壮且效率高，能够直接提取大型开源程序的静态信息.

关键词：静态信息；GCC编译器；程序的中间表示；代码静态分析

中图分类号：TP311 **文献标志码**：A **文章编号**：0367－6234(2011)03－0062－05

Retrieval method of static code information for multi-language

PANG Long, WANG Tian-tian, SU Xiao-hong, MA Pei-jun

(School of Computer Science and Technology, Harbin Institute of Technology, 150001 Harbin, China, hitpanglong@163.com)

Abstract: There are many requirements for the multilanguage static information retrieval, and it is wasteful and complex to build specific front end for each language. So to meet the need and conquer the weakness we present a method based on GCC source code change to provide a uniform interface for retrieval. According to the static information type and the GCC inside mechanism, this method integrates the specific hook point with the GCC's internal functions to gather the needed. The details to be collected include: the type and function declaration, the statements traverse and the uniform multilanguage middle-representation. The reusability of this method reduces the duplicated cost of the construction for each language. The comparison experiments shows that this method can efficiently and robustly parse multilanguage and be directly applied to large－scale open source code to retrieve the static information.

Key words: static code information; GCC compiler; program middle-representation; static code analysis

代码静态分析已经广泛应用于程序理解、软件缺陷检测、程序验证等方面.静态信息提取是代码静态分析的基础，是将代码文本转换为可分析处理的中间逻辑表示结构的过程，其所支持解析的程序语言种类、生成的中间表示的表达能力都极大影响静态代码分析的应用范围.当前主要采用的方法为：1）Yacc类语法生成解析器[1－2].该方法语言适应能力强，但后续语法的消歧和调试开销大，中间表示结果单一；2）专用前端解析，如CIL，Clang等[3－4]，文档完整，但支持语言单一，需其他预处理工具支持，中间表示单一；3）对编译器扩展[5－6]，应用便捷，但对编译器内部已有功能重用率低，执行效率不高；4）解析编译器中间调试文件，重构中间表示[7－10]，支持多种语言，但实现复杂，后续测试开销大.

为了支持多语言解析，并提供统一且具备不同表达能力的中间表示，满足健壮性和稳定性的要求，通过在代码级别改造开源GCC编译器来实现静态信息的提取.该编译器应用广泛、成熟可靠，所以在此基础上改造可以降低后续分析程序

收稿日期：2010－01－31.

基金项目：国家自然科学基金资助项目(60673035)；高等学校博士学科点专项科研基金资助项目(20092302110040).

作者简介：逄 龙(1984—)，男，博士研究生；
苏小红(1966—)，女，教授，博士生导师；
马培军(1963—)，男，教授，博士生导师.

的复杂度，但这种方法的主要难点是代码规模较大、整体复杂、文档不完整、代码的设计目标与静态分析的目标不同，因而需要针对静态分析的目标在代码级别对其进行修改. 本文首先对静态信息所依赖的 GCC 源代码内部运行过程和相关机制进行分析，然后针对类型和函数声明定义、函数体内语句遍历和中间表示访问等静态信息，在 GCC 编译器的不同阶段提出了运行改入点和内部辅助函数相结合的提取方法，并给出整体的修改、应用流程，最后通过提取典型静态信息的对比实验，表明该方法提高了执行效率，增强了静态信息提取的可靠性.

1　与静态信息提取相关的 GCC 源代码分析

GCC－4.3.0 编译器整体结构如图 1 所示. 整体划分为 3 层：前端解析源代码、中端负责机器无关的优化和后端负责目标机器相关的优化和变换. 源代码在前端生成原始中间表示后，通过后续的层间流转、层内变换和处理，最终生成目标代码. 本文主要针对与静态信息提取相关的前端、中端和涉及的中间表示进行分析.

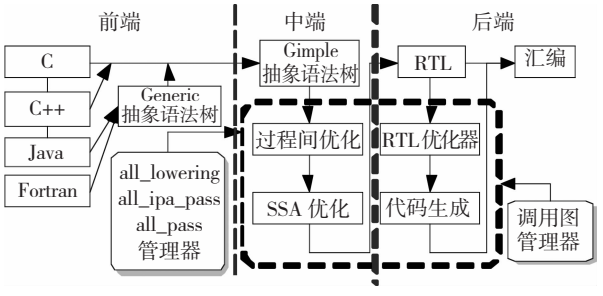

图 1　GCC 编译器结构框图

1.1　整体流程

编译器 GCC 开始执行后，针对编译对象的不同程序语言，调用对应语言前端解析，将该编程语言源代码文本解释为与语言相关的抽象语法树形式的中间表示. 经前端 genericize 函数去除语言相关性的节点，转换为 Generic 形式或直接为 Gimple 形式，输入中端进行优化处理. 中端优化包括与目标机器无关的过程间和过程内优化. 调用图管理器负责维护中间表示的过程间调用结构逻辑关系，用以管理过程间优化处理的中间结果，遍管理器负责维护编译器内对中间表示进行优化处理的各步骤定义、依赖关系和调用顺序，用以指导优化处理次序.

1.2　中间表示

GCC 内部中间表示是各层间流转和层内处理对象的重要数据结构，根据不同处理阶段转换为特定形式，同时也是静态信息提取的直接来源，所蕴含静态信息的质量和结构组织的复杂性决定着静态信息提取方法的质量和效率. GCC 前端和中端主要采用抽象语法树、Generic、Gimple 和 SSA 这 4 种形式中间表示.

抽象语法树是经过解析源代码文本后得到与之直接对应的逻辑结构，与语言相关. Generic 形式是对抽象语法树的简化，统一语句结构，去除语言的相关性，包含副作用(Side Effect). Gimple 形式是在 Generic 形式的基础上限制每条语句的操作数 <3 的一种约简，简化了复杂表达式结构，引入中间临时变量. SSA 形式是在 Gimple 基础上增加数据流信息的一种中间表示，在每次赋值时，对左边变量都增加序号加以辨别，而且后续读取该变量的表达式直接指向该变量序号的定义处. SSA 和 Gimple 均适合作为静态信息提取的中间表示.

1.3　遍(Pass)

在 GCC 优化中，对编译单元(函数或文件)的一次遍历处理，称为遍(Pass). GCC“遍管理器”是对遍定义、组织和执行而实施管理的一组机制. 根据静态信息种类，提取过程可抽象为遍，通过在合适的步骤后对合适的中间表示遍历来实现，这样可以重用 GCC 内部已经提供的通用功能和信息，如冗余代码去除、控制流图、数据依赖等. GCC 通过遍来关联中端和后端内具体的优化处理步骤，遍的机制分为 3 个部分：遍的定义、遍的组织和遍的执行.

2　静态信息提取

在分析与静态信息提取相关的 GCC 内部代码组织机制的基础上，根据信息来源，给出确切的改入点，结合内部辅助函数从获得的中间表示中提取所需信息. 核心思想是在 GCC 运行过程中，在合适的位置得到合适的中间表示，使用合适的方法提取静态信息.

2.1　改入点

改入点是修改 GCC 源代码的位置，因为不同改入点会获得不同中间表示，如图 2 所示.

1)遍改入点. 通过 GCC 内部遍机制，以遍历函数体内语句方式来提取静态信息. 因为遍管理器处于编译器中端和后端部分，获得语言无关的中间表示，所以遍改入点可直接应用于编译器所支持语言. 根据所需提取静态信息种类和 GCC 已定义遍所提供的不同中间表示，在适当类型的遍中适当顺序位置处加入自定义遍. 在图 2 中 0 处

init_optimization_passes,初始化遍组织关系:低级化遍(all_lowering_passes)、过程遍(all_ipa_passes)和所有遍(all_passes).

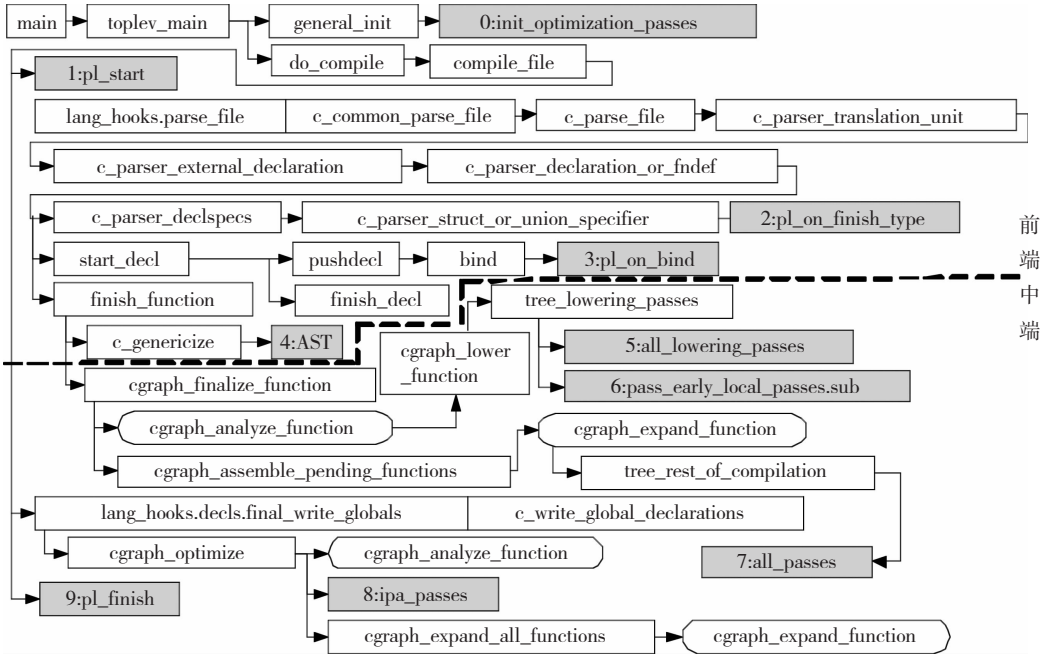

图2 GCC中语言前端和统一中端调用图

为获得Gimple形式中间表示,在低级化遍中的pass_build_cfg遍后增加自定义遍.为获得数据流相关的中间表示,在过程间遍中的pass_early_warn_uninitialized遍后增加改入点.其中:pass_build_cfg遍已经建立了过程内控制流图和基本块,通过遍历程序基本块间关系和块内语句可获得初步的Gimple格式的统一中间表示;pass_remove_useless_stmts遍去除无用冗余的语句;pass_early_warn_uninitialized遍检测使用未初始化变量情况;pass_build_ssa遍建立了SSA的中间表示,通过遍历定义-使用链获得数据流相关信息.

调用改入点是GCC调用内部遍的位置,可根据需要调整具体调用次序.在图2中,获取Gimple的自定义遍从属于低级化遍,在5处调用.获取SSA的自定义遍从属于过程间遍的第2层子遍pass_early_local_passes,在6处和8处调用.所有遍在7处调用.

2)解析相关改入点. 通过GCC前端解析提取语言相关静态信息.

复合类型改入点为获得结构体、联合体和类的类型定义相关静态信息.在声明与函数定义翻译单元加入改入点.在C语言前端的c_parser_struct_or_union_specifier函数内1 979行处增加改入点,该处获得的是解析出的类型定义,如图2中2所示.在C++语言前端(cp/Class.c)的finish_struct_1函数返回前增加改入点,以获得类定义中的域相关静态信息.

标识符改入点为获得全局变量相关静态信息.GCC的标识(Identity,ID)为3类:位置标签、复合类型名和和其他ID. ID通过struct c_binding与具体中间表示关联,以作用域(struct c_scope类型)为集合通过链表方式存储.在关联函数bind内加入改入点,如图2中3所示.

原始中间表示改入点为获得抽象语法树等静态信息.GCC通过lang_hooks类型定义各语言解析、后端处理等入口函数.C语言中,在c_genericize函数中,加入改入点,如图2中4所示,获得原始抽象语法树中间表示.C++语言中,在cp/decl.c的finish_function中调用cp_genericize函数前增加改入点,获得原始抽象语法树中间表示.

3)辅助改入点. 为后续分析处理传递提取的静态信息,在翻译单元编译开始前和结束后进行必要的初始化和终结工作,在调用语言类函数指针前和之后增加改入点,如图2中1和9处所示.此处可以负责处理文件描述符、过程间通讯和数据库操作等与外界交换信息的建立与消解,还可以设置初始化环境等.

2.2 辅助函数

辅助函数是在特定改入点获得对应中间表示后,根据静态信息种类和GCC代码内部组织机制,来提取所需静态信息的具体方法.

1)位集合操作.数据流分析经常使用集合操

作，GCC 内部实现了集合相关操作运算的抽象数据类型 sbitmap，定义包含在 Sbitmap. h 内；提供基于位内部表示和相关操作，初始化与可变长度调整：sbitmap _ alloc、sbitmap _ resize；集合运算操作：sbitmap _ difference、sbitmap _ not；向量位集操作：sbitmap _ vector _ alloc；位集合设置内联函数：SET _ BIT、sbitmap _ iter _ init 等.

2）辅助信息遍历. 辅助信息包括控制流图和数据流信息. 低级化遍中建立控制流图，在所有遍生成的 SSA 中间表示蕴含数据流信息. struct basic _ block _ def 和 struct edge _ def 定义了基本块和边. 以 ENTRY _ BLOCK _ PTR 宏获得入口基本块开始，以 FOR _ EACH _ BB 宏基本块间流转，实现基本块遍历. 在基本块内，bsi _ start 获得块内首语句，bsi _ next 获得下一条语句和 bsi _ end _ p 判定块内语句结束与否来实现基本块内语句遍历. walk _ use _ def _ chains 函数遍历数据流信息.

3）表达式遍历. 遍历表达式中的操作数来提取变量访问. walk _ tree 函数遍历抽象语法树表达式；walk _ stmts 函数遍历 Gimple 形式表达式，提供了赋值表达式中区别左边和右边表达式的标志. FOR _ EACH _ SSA _ TREE _ OPERAND 配合类型标志位组合遍历 SSA 操作数.

4）中间表示节点访问. 抽象数据类型 tree 统一表达各种中间表示，提供了对应访问函数. TREE _ CODE 宏获得节点种类；TREE _ OPERAND 获得节点下指定操作数；针对 COMPONENT _ REF 类型节点，DECL _ CONTEXT 获得其域变量所属类型；TYPE _ NAME 获得变量的类型节点；DECL _ NAME 获得类型节点名称，如在 C 语言中 typedef 定义的类型名称；IDENTIFIER _ POINTER 获得具体 ID 字符串；TYPE _ FIELDS 获得结构体或联合体内域的列表.

2.3　其他相关修改

在特定修改点利用辅助函数获取了所需的静态信息后，需要通过修改配置文件实现与 GCC 的整合. 修改 GCC 源代码提取静态信息的整体过程如图 3 所示.

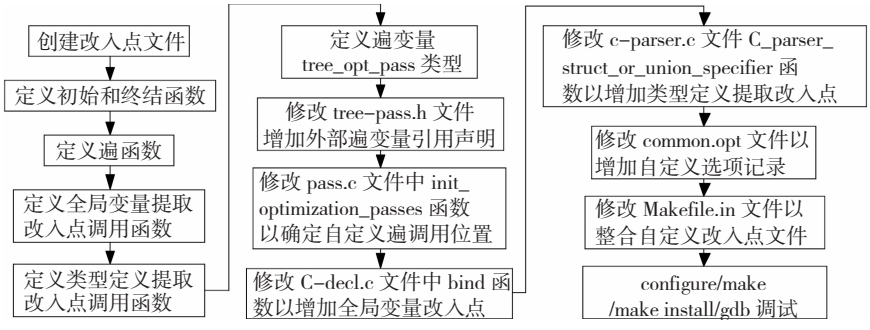

图 3　修改 GCC 源代码提取静态信息过程图

1）控制选项配置修改. 是外部调用 GCC 可设置的参数，通过添加与控制静态信息提取相关选项来控制是否运行. 为便于控制，利用增加 common. opt 文件中选项定义记录的方式来控制是否运行静态信息提取.

2）工程配置修改. 将自定义的文件通过编译、链接整合到 GCC. 首先增加生成该源代码文件的目标文件的依赖关系，然后将此目标文件加入 OBJS-common 所依赖列表中实现整合.

3　结果与分析

为了验证本文所提出方法的有效性、效率和健壮性，在 2. 6 GHz 主频 CPU、2 GB 内存和 Ubuntu 9. 10 操作系统的测试环境下，分别按本文方法和 CIL 方法，提取结构体域变量读、写和函数访问等静态信息，应用于 5 个开源软件进行测试，测试结果如表 1 所示. 其中通过 UCC 统计代码规模，Linux 系统指令 time 统计运行时间.

表 1　实验结果对比表

测试项目名称	版本号	程序语言	规模/千行	本文方法耗时 t/ s	CIL 方法耗时 t/ s	耗时降低比率/%	本文方法错误数	CIL 方法错误数
gtk	2. 18. 5	C	611	908. 49	1 362. 02	33. 30	0	3
httpd	2. 0. 48	C	160	170. 18	219. 23	22. 37	0	0
openssh	5. 3	C	63	101. 91	131. 16	22. 30	0	1
mysql	5. 0. 48	C/C + +	912	1 572. 81	不支持	不支持	0	不支持
tomcat	6. 0. 20	Java	165	5 373. 33	不支持	不支持	1	不支持

如表1所示,由于本文方法是在GCC源码级上修改,所以继承了GCC特点,在支持语言、耗时和错误数方面优于CIL方法. CIL方法由于对扩展语法支持不完备而发生的错误有:openssh中对64位整数常量不能识别;gtk中可变参数无法定义别名属性. 由于GCC对Java扩展库实现不完整,导致本文方法提取tomcat时部分依赖于此的文件无法编译,但可以利用Open Java对应库文件替换予以解决. 本文方法在处理Java语言时,较C语言耗时较大,主要由于前端频繁启动虚拟机造成的,通过改进前端工作方式降低耗时. CIL方法由于对编译器参数处理不完善,导致分析gtk项目时,无法自动完成需手工干预,易用性方面弱于本文方法.

4 结 论

1)一次实现后可支持多语言前端,可直接应用于C/C++/Java等主流编程语言.

2)通过增加参数选项,按项目配置文件自动执行静态信息提取.

3)重用了GCC内前端和中端组织机制,避免了重复实现,保证了效率、健壮性和稳定性.

参考文献:

[1] TERENCE P. The Definitive ANTLR Reference: Building Domain-Specific Languages[M]. Raleigh, Dallas: Pragmatic Bookshelf, 2007.

[2] SCOTT M, GEORGE N. Elkhound: A fast, practical glr parser generator[C]//Proceedings of the 13th International Conference on Compiler Construction. Barcelona: EATCS, EASST, EAPLS, ACM, 2004:325-336.

[3] GEORGE C N, SCOTT M, SHREE P R, *et al*. CIL: Intermediate language and tools for analysis and transformation of C programs[C]//Proceedings of the 11th International Conference on Compiler Construction. Grenoble: EATCS, EASST, EAPLS, ACM, 2002:213-228.

[4] CHRIS L, VIKRAM A. LLVM: A compilation framework for lifelong program analysis & transformation[C]//Proceedings of the international symposium on Code generation and optimization. PaloAlto: ACM, 2004: 75-92.

[5] SEAN C, DANIEL J D, EREZ Z. Extending GCC with modular gimple optimizations[C]//Proceedings of GCC Developers' Summit. Ottawa: Linux Symposium, 2007: 31-37.

[6] TARAS G, DAVID M. Using GCC instead of grep and sed[C]//Proceedings of GCC Developers' Summit. Ottawa: Linux Symposium, 2008:21-32.

[7] 李鑫, 王甜甜, 苏小红, 等. 消除GCC抽象语法树文本中冗余信息的算法研究[J]. 计算机科学, 2008, 35(10):170-172.

[8] KRAFT A, MALLOY A, POWER F. A tool chain for reverse engineering C++ applications[J]. Science of Computer Programming, 2007, 69(13):3-13.

[9] GSCHWIND T, PINZGER M, GALL H. TUAnalyzer-analyzing templates in C++ code[C]//Proceedings of the 11th Working Conference on Reverse Engineering. Delft: IEEE, 2004: 48-57.

[10] ANTONIOL G, DIPENTA M, MASONE G, *et al*. Compiler hacking for source code analysis[J]. Software Quality Journal, 2004, 12(4):383-06.

(编辑 张 红)