Software Engineering and Applications 软件工程与应用, **2021, 10(5), 654-660**
Published Online October 2021 in Hans. http://www.hanspub.org/journal/sea
https://doi.org/10.12677/sea.2021.105070

Hans汉斯

基于GCC的C语言抽象语法树重建与可视化研究

林　渤，王枭雄，胡建鹏

上海工程技术大学电子电气工程学院，上海
Email: mr@sues.edu.cn

收稿日期：2021年4月14日；录用日期：2021年10月22日；发布日期：2021年10月29日

摘　要

抽象语法树(abstract syntax tree, AST)作为一种重要的中间表示形式，在代码静态分析领域有着重要的研究意义。本文通过研究GCC生成的抽象语法树文本内容，给出重建抽象语法树及可视化的方法。GCC编译器生成的抽象语法树内容存在大量冗余，不能直接进行解析。针对此问题，本文提出一种改进的去冗余算法，从根节点中先找到main函数后进行遍历，相比常规的去冗余算法有更高的效率，同时针对可视化操作进行了结构性的优化，能够将原始的文本转换为JSON字符串，配合web树形控件的展示形式实现了可视化界面。

关键词

抽象语法树，GCC，C语言，可视化

Study on AST Reconstruction and Visualization of C Language Based on GCC

Bo Lin, Xiaoxiong Wang, Jianpeng Hu

School of Electronic & Electrical Engineering, Shanghai University of Engineering Science, Shanghai
Email: mr@sues.edu.cn

Received: Apr. 14th, 2021; accepted: Oct. 22nd, 2021; published: Oct. 29th, 2021

Abstract

The abstract syntax tree (AST), as an important intermediate representation, plays an important role in the field of static analysis of code. This paper gives a method to reconstruct and visualize the

文章引用：林渤，王枭雄，胡建鹏．基于GCC的C语言抽象语法树重建与可视化研究[J]．软件工程与应用, 2021, 10(5): 654-660. DOI: 10.12677/sea.2021.105070

AST based on the textual content of the AST generated by the GCC compiler. The contents of the abstract syntax tree generated by the GCC compiler are heavily redundant and cannot be parsed directly. To address this problem, this paper proposes an improved redundancy removal algorithm, which finds the main function from the root node first and then traverses it. It is more efficient than the conventional redundancy removal algorithm and is easy to convert the raw text of AST into JSON strings. Compared with the conventional redundancy removal algorithm, it has higher efficiency and structural optimization for visualization operation, which can convert the raw text of AST into JSON string and realize a visual interface with the presentation of web tree control.

Keywords

AST, GCC, C Language, Visualization

Copyright © 2021 by author(s) and Hans Publishers Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/

Open Access

1. 引言

随着互联网和计算机技术的发展，传统的教学方式已经不能满足当前的需求，特别是程序设计类课程。C 语言作为最基础高级语言之一，使用人数众多，实验教学面临改革[1]。传统实验课上仅凭课上时间让教师检查代码几乎不可能，已有的 OJ 系统仅能通过测试用例去判断代码的正确性，因此代码检测已经成为一个热点问题。目前在代码检测方面已经有很多研究并且有成熟的产品存在，其中在性能较好的静态代码检查方面主要有机器学习和抽象表示两个方向。如 C/C++代码检测中的一个产品 cppcheck 其中会使用到正则表达式的方法来匹配一类代码错误。同时，在抽象语法树方面也有大量的研究。抽象语法树本质上是一棵树，树形结构可以直观的表示出源程序的语法结构，同时作为一种中间表示，它还包含了程序完整的信息，因此适用于静态分析领域。GCC 是由 GNU 开发的编译器，它可以处理包括 C 语言在内的多种语言[2]。GCC 是目前使用的最为广泛的编译器系统之一，经过大量的实践证明，GCC 编译系统生成的代码具有很高的可靠性和运行效率。可以凭借 GCC 的命令生成抽象语法树后对其解析得到需要的结果。GCC 生成的抽象语法树复杂且有依赖性，目前大部分研究只将抽象语法树解析后用于计算相似度，还没有人将抽象语法树进行可视化处理得到适于人阅读的图形化界面，因此本文提出了对 GCC 生成的抽象语法树进行重建并进行可视化处理的一种方法。

2. GCC 抽象语法树

抽象语法树(Abstract Syntax Tree, AST)是一种抽象的表示，它能以树状的形式表现出高级语言的语法结构，而不会表示出真实语法中出现的细节。作为一种中间表示形式，主要被用于预处理，因此可以应用在代码分析领域。代码经过词法分析得到字符序列，再由语法分析得到抽象语法树。抽象语法树的结构简单，根节点表示整个程序，内部节点是抽象语法结构或者单词。图 1 为 GCC 生成的抽象语法树部分节点内容。

GCC 生成的抽象语法树常用的节点类型大致分为 7 种，分别为声明节点(_decl)、标识符节点(identifier_node)、类型节点(_type)、常量节点(_cst)、表达式节点(_expr)、列表节点(_list)、其它节点。每个节点都是以“@”符号开始跟上节点的索引，接着是节点类型和信息，包括变量名、类型、所属函数、大小等等，这些信息的值有的是直接显示的，有的是对应节点的索引值。

@1	type_decl	name: @2	type: @3	chain: @4
@2	identifier_node	strg: int	lngt: 3	
@3	integer_type	name: @1	size: @5	algn: 32
		prec: 32	sign: signed	min : @6
		max: @7		
@4	type_decl	name: @8	type: @9	chain: @10
@5	integer_cst	type: @11	low : 32	
@6	integer_cst	type: @3	high: -1	low : -2147483648
@7	integer_cst	type: @3	low : 2147483647	
@8	identifier_node	strg: char	lngt: 4	
@9	integer_type	name: @4	size: @12	algn: 8
		prec: 8	sign: signed	min : @13
		max : @14		
@10	type_decl	name: @15	type: @16	chain: @17

Figure 1. GCC generates some nodes of the abstract syntax tree
图 1. GCC 生成抽象语法树部分节点

使用 GCC 编译器生成的抽象语法树存在大量的冗余信息。这些信息过于庞大很难直接进行解析，因此需要对其优化从而去除多余和无意义的节点信息。针对这个问题，本文提出了一种去除冗余的方法对抽象语法树进行重建并进行可视化，使其达到易于阅读的目的。

目前已经有很多软件应用基于抽象语法树实现，如程序自动评分[3]，它是通过对生成的抽象语法树进行解析和标准化，使用树的编辑距离计算两者之间的相似度后对其进行评分。程序修复[4]技术在缺陷定位方面也是基于抽象语法树实现的，通过将错误模式和错误代码的抽象语法树进行匹配得到相应的节点。抄袭检测[5]也是通过对 C 语言代码进行规范化处理，对代码中的变量和函数进行了无类型化处理，经过词法分析和语法分析后得到抽象语法树，对其进行分析处理并计算相似度来判断是否抄袭[6]。

3. 抽象语法树的重建

3.1. 生成抽象语法树

由于 GCC 版本的不同，生成抽象语法树的命令也不同。本文介绍一种生成抽象语法树的相关命令：

-fdump-translation-unit 命令会将整个翻译单元的树结构表示形式转储到文件中。生成与源文件名相同的 tu 文件，同时允许调用者以翻译单元(TU)表的形式提取这些信息。

本文使用-fdump-translation-unit 生成抽象语法树，该命令可以得到完整的抽象语法树，其中包含了翻译单元的所有信息。翻译单元中有关于代码的各种信息，包括变量声明、数据结构的定义等。TU 文件中的信息适合进行解析，因此选用该命令生成 AST。

3.2. 重建抽象语法树

GCC 直接生成的抽象语法树文本包含了大量的冗余信息，如果直接对原始文本进行解析不仅会降低效率，还可能影响分析的准确率。因此需要对原始的抽象语法树进行重建。重建的主要方法主要是由预处理、标准化、构建树三个方面组成[7]。其中，预处理是实现重建抽象语法树的基础。整个重建的步骤如图 2。

3.2.1. 初始化节点

原始的 tu 文本由多个节点组成，每个节点都是由标号、节点标识、属性组成。首先要读取 tu 文件，以“\n@”为分界符将原始文本分成数个节点，再将其放入一个列表。接着遍历列表，处理每一个节点，将其转换为一个节点属性字典，处理后进行实例化。然后处理实例中“attr”字符串，其中属性节点可以有多个，它不仅表示属性，也表示子节点。针对本文所需的可视化需求添加相应实例属性，构建树的结

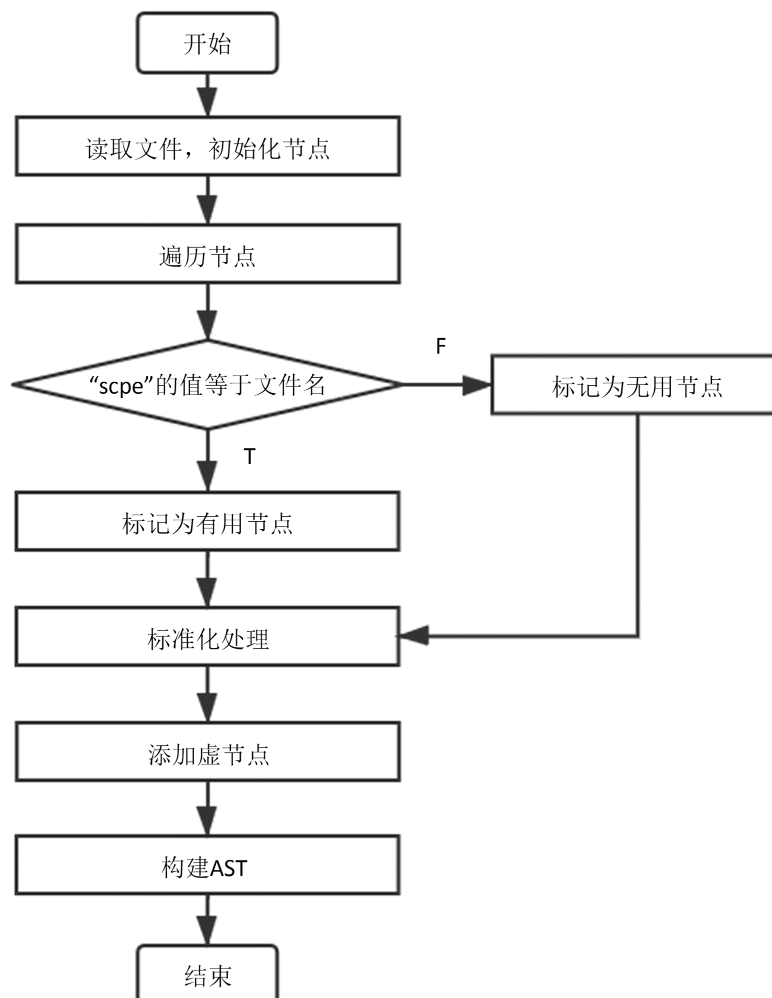

Figure 2. The process of rebuilding the AST
图 2. 重建 AST 流程

构。最后将处理后的节点重新放入节点列表。

3.2.2. 标记有用节点

有用节点指与数据流和控制流相关的节点，无用节点指与数据流和控制流无关的节点。遍历节点列表，再对当前节点属性进行遍历标记出有用的节点。如果键是“srcp”但不是本程序，则将其标记为无用节点，如果键是“srcp”且属性值为本程序，则将其标为有用节点。继续遍历节点列表完成关联节点类型，如果检测到属性值为节点，则替换为相应节点序号和对象元组。最后将标记为有用的的节点放入有用节点集合中，同时将原始编号放入有用节点原始编号集合中。

3.2.3. 标准化节点

该操作主要是为了去除冗余节点。遍历有用节点集合，比对每个有用节点的子节点原始编号是否在有用节点原始编号集合中或该节点的子节点名是否为“scpe”，若是则将该节点的子节点弹出。接着移除空的子节点，根据树形结构重新给节点编号，最后移除无效的子节点，完成对节点的标准化操作。

3.2.4. 构建抽象语法树

准备一个空字典。遍历有用节点集合，如果是第一个有用节点，则将空字典的一个有用节点原始编

号位置的值设置为 1。查找有用节点的子节点的原始编号作为序号，如果该序号所对应的有用节点的原始编号不在字典中，则将该节点处的“关系”字段设置为当前有用节点对应子节点的名字，并将该对应的有用节点加入子节点中，以便显示。否则，将对应节点深拷贝一个虚拟节点，加入当前节点的虚拟节点中，最后弹出第一个节点。将处理好的集合转换为 JSON 字符串返回。

3.2.5. 叶子节点及虚节点

由于 GCC 生成的抽象语法树中包含了大量冗余的信息，尤其是针对子节点没有一个明确的定义，所以本文中设定将子节点中依然还有子孙的节点提至上层，只显示代码中的信息，不显示类型定义，进行规范化。同时，原始的 AST 文本实际上是一种图形结构，存在大量的相互引用，因此文本设置了“虚节点”，即将那些反复出现的、变量声明的或者函数声明引用时的节点称之为虚节点，目的是为了将重建的抽象语法树化成一棵真正的树。

4. 抽象语法树的可视化

本文针对可视化需求设计了友好的 UI 界面，可以在浏览器中查看树状结构图。展示树状结构图使用的是开源的 OrgChart 组件，它提供了丰富的功能，包括展开节点、缩放操作、支持移动设备等。同时支持本地或远程的数据源，只需要提供 JSON 格式的数据源就能将其渲染为树形结构。基于它小而精的特点，可以通过二次开发定制各种繁琐的需求，达到所需目的。在代码编辑方面，系统使用了了开源的 Monaco 编辑器，支持快速的代码补全提示、自动换行、格式化代码等功能。用户可以实时查看代码对应的抽象语法树的树形结构图。同时系统还提供了 JSON 的数据源和 tu 文本，可以清楚的看出重建抽象语法树前后的内容差异。

本文中重建后的抽象语法树还不能直接作为可视界面的数据源，需要进行处理后才能使用。数据处理主要操作是将原数据节点内容进行区分，通过添加字段来区分该节点是否为虚节点，并控制节点的显示样式，图 3 中浅色带虚框的节点即为虚节点。在加载节点时还需要调用相关的初始化函数，控制节点显示的标题、内容、关系等信息以及为每个节点设置点击事件来查看该节点的详细信息，图中每个节点的上方为关系字段，下方为标号和类型字段。加载完成后可以直接点击树状结构图中的节点查看该节点的内部信息及其相应的介绍。图 3 为生成的可视化树状结构图(隐藏了部分节点)。

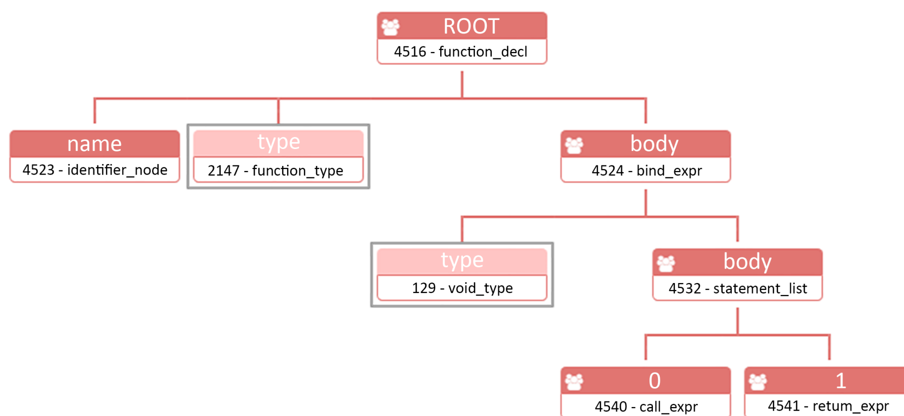

Figure 3. Visualization tree structure
图 3. 可视化树状结构图

5. 应用范例：代码结构相似性

为了展示重建 GCC 生成抽象语法树的效果，本文提供了两段相似的源代码，如表 1，两段代码的功

能都是用循环结构打印从 1 到 3 的三个数字，而第一段代码定义的变量名为 i，且使用 for 循环。第二段代码定义变量名为 j，使用的是 while 循环。两种循环结构生成的抽象语法树类似，且不同的的变量名对抽象语法树的结构不产生影响，结果如图 4、图 5。

Table 1. Two similar codes
表 1. 两段相似代码

代码 1	代码 2

#include <stdio.h>
int main() {
int i;
for(i = 0; i < 3; i++) {
printf("%d", i);
}
}
	#include <stdio.h>
	int main() {
	int j = 0;
	while(j < 3) {
	printf("%d", j);
	J++;
	}
	}

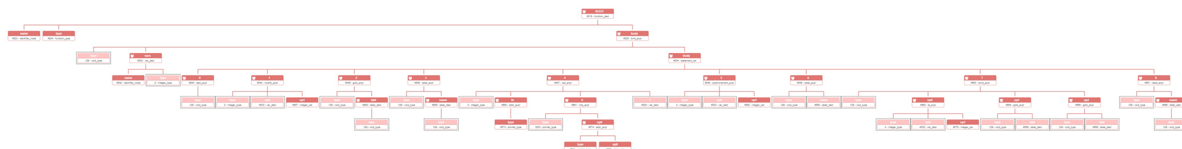

Figure 4. For loop results
图 4. For 循环结果

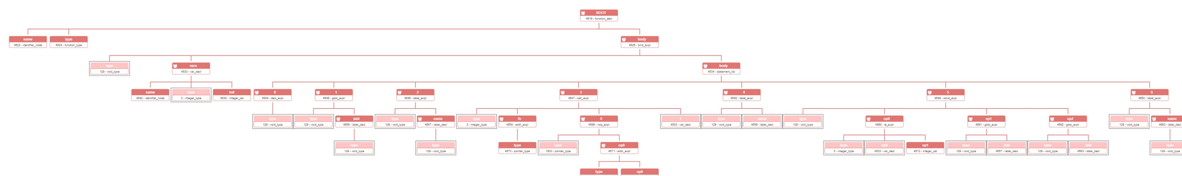

Figure 5. While loop results
图 5. While 循环结果

从两个生成的抽象语法树结构对比看来，两段代码功能相似，树形结构相似。观察树中的具体节点内容可以看出，两段代码中的变量声明、判断条件和打印输出等完全相同的代码所对应的节点内容相同，只有在循环结构上才出现树形结构差异情况。可见，本文所介绍的方法在静态代码分析领域有者显著的效果，可以用于软件缺陷预测技术[8]。

6. 结束语

本文提出了一种重建 GCC 生成的抽象语法树的方法，实现了抽象语法树的直观展示。但是在解析处理抽象语法树文件过程中仅使用了改进的去冗余算法，效率比常规去冗余算法提高了一些。未来将对去除冗余的算法进行优化，使其达到一个更高效的程度。代码检测无论在数据分析还是在数据挖掘等领域中都占据了重要的作用。后续，我们将在算法和应用方面做进一步的研究，使其成为一个成熟的工具，并能将其应用在 C 语言实验平台上完成自动静态评分功能。

参考文献

[1] 徐伟, 陈凯明, 马建辉, 赵家兴. 计算机专业程序设计课程在线教学改革探究[J]. 软件导刊, 2020, 19(12): 181-184.

[2] Richard, M. and Stallman, G.N.U. (2003) Compiler Collection Internals. http://gcc.gnu.org/onlinedocs/gccint

[3] 焦秀秀. 基于抽象语法树的 C 编程题自动评分方法研究及应用[D]: [硕士学位论文]. 西安: 西安理工大学, 2019.

[4] 周风顺, 王林章, 李宣东. C/C++程序缺陷自动修复与确认方法[J]. 软件学报, 2019, 30(5): 1243-1255.

[5] 冯君远, 赖明钦, 李启良. C 语言源代码抄袭识别的研究[J]. 福建电脑, 2013, 29(5): 34-36.

[6] Young-Chul, K. and Jaeyoung, C. (2005) A Program Similarity Evaluation Using Keyword Extraction on Abstract Syntax Tree. *The KIPS Transactions*: *PartA*, **12A**, 109-116. https://doi.org/10.3745/KIPSTA.2005.12A.2.109

[7] 于俊, 李雅洁, 程礼磊, 连顺, 谭昶, 丁德成, 刘淇. 高教程序代码作业抄袭检测的方法研究与实践[J]. 中国科学技术大学学报, 2020, 50(8): 1048-1057.

[8] 张启航. 基于抽象语法树的代码缺陷检测技术设计与实现[D]: [硕士学位论文]. 北京: 北京邮电大学, 2020.