文章编号:1671-4598(2021)08-0196-06 DOI:10.16526/j.cnki.11—4762/tp.2021.08.038 中图分类号:TP311 文献标识码:A

基于标识符的代码影响域自动化分析方法

纪 承,付高生,白 洋
(北京京航计算通讯研究所,北京 100074)

摘要:装备软件质量评测自动化需求日益提升;装备软件在质量评测过程要经过多轮迭代修改,修改后代码的影响域分析是软件质量评测中重要的一环;目前修改后代码的影响域分析完全依赖人工;由于装备软件代码规模大,人工分析耗时较长,拉长了装备交付周期;为了提高装备软件质量评测的效率,代码影响域的自动化分析的研究势在必行;针对装备软件代码影响域分析问题,提出一种基于标识符的代码修改特征提取方法,该方法可以快速提取大规模代码的修改特征;提出一种基于标识符检测队列的影响域分析方法,可以实现装备软件影响域的快速自动化分析,准确率达85.7%,该方法可以满足现有装备软件影响域自动化分析的需求。

关键词:装备软件;质量评测;修改特征;影响域;标识符

An Automatic Analysis Method of Code Influence Domain Based on Identifier

JI Cheng, FU Gaosheng, BAI Yang
(Beijing Jinghang Institute of Computing and Communication, Beijing 100074, China)

Abstract: The demand for automation of equipment software quality evaluation is increasing. Multiple rounds of iterative modification are necessary in the quality evaluation process of equipment software, the impact domain analysis of the modified code is focused on the software quality evaluation process. At present, the influence domain analysis of the modified code completely relies on manual labor. As the large scale of equipment software code, manual analysis takes a long time, and the equipment delivery cycle is lengthened. In order to improve the efficiency of equipment software quality evaluation, it is imperative to study the automatic analysis of code influence domain. To solve the problem of analyzing the influence domain of equipment software code, an identifier-based code modification feature extraction method is proposed, which can quickly extract the modification characteristics of large-scale codes; an analysis method of influence domain based on identifier detection queue is proposed, which can realize rapid automatic analysis of influence domain of equipment software, with an accuracy rate of 85.7%. This method can satisfy automatic analysis of influence domain of existing equipment software.

Keywords: equipment software; quality evaluation; modification characteristics; influence domain; identifier

0 引言

软件在军队装备中的作用和地位日益提高,软件质量对军队战斗力生成有着直接影响,只有确保装备软件可靠、稳定,才能保证军队装备在关键时候发挥作用[1]。为避免因软件故障而造成装备无法使用的情况,军队对装备软件的可靠性和稳定性要求极高。对装备软件进行质量评测可以大幅度提升装备软件的可靠性和稳定性,因而质量评测方法的研究对提高装备软件的质量有重要意义[2]。软件质量评测过程中自动化工具只能实现部分测试功能[3],还有较多工作是由人工操作完成。根据相关军用软件测试标准和规定,装备软件评测过程要经过多轮迭代,不断修正通过软件测试发现的软件问题。每一轮迭代前后的两个版本的代码差异较大,修改后代码的影响域分析至关重要,目前该部分软件评测完全依赖人工实现。为节约装备软件质量评测所耗费的成本和时间,在质量评测过程中使用代码自动化比对工具是现今测试机构广泛使用的方法,对软件评测自动化工具进行研究显得格外重要。本文首先介绍了基于标识符的代码对比方法提取修改特征的流程,并介绍了基于修改特征分析影响域的代码分析方法,然后通过实验证明在大型工程代码比对工作中该分析方法有较好效果,最后就该方法的有效性和实用性进行了分析,并对后续研究方向进行展望。

1 相关工作

软件评测领域与恶意代码检测、程序编译等多个领域均有交集,各种方法繁杂混乱,在软件影响域分析问题上并没有已经成型的方法。软件质量评测分析方法一般分为

收稿日期:2020-12-24; **修回日期**:2021-01-06。
基金项目:国家自然科学基金(11901544)。
作者简介:纪 承(1996-),男,北京人,硕士研究生,主要从事软件评测方向的研究。
引用格式:纪 承,付高生,白 洋.基于标识符的代码影响域自动化分析方法[J].计算机测量与控制,2021,29(8):196-201.

动态分析方法[4]和静态分析方法[5]两种，动态分析方法需要运行程序，而静态分析方法则是对源代码进行分析。代码自动化比对工具的分析内容以程序源代码为主，主要应用静态分析方法。代码的静态分析方法有很多种，一般使用较为广泛的包括基于抽象语法树、结构图、标识符、度量值的代码分析方法。基于抽象语法树的代码分析方法[6]在分析过程中引入编译器生成的抽象语法树，可以精准的分析代码结构和改变，但是时间复杂度较高，不适合代码规模较大的代码对比。基于控制流图的代码分析方法[7]从代码语义结构角度出发，分析思路与基于抽象语法树的方法有一定相似，但是也不适合大型工程代码比对。基于标识符属性特征向量相似度的代码分析方法[8]使用了标识符和属性度量值两种方法，具有时间复杂度低、容易扩展到多种语言的特点，但是需要根据具体问题衡量代码分析方法的有效性。现有代码对比静态分析方法主要从代码语句角度分析代码的相似度，但缺乏影响域分析等对代码的深层逻辑分析。代码深层逻辑分析可以挖掘软件的隐藏隐患，更加符合软件质量评测工作的需求，对代码深层逻辑分析方法的研究愈发受到重视。软件质量评测目前有成熟的静态分析工具，但该工具只分析每次迭代前后的代码变化项，无法分析代码变化项的影响域，导致装备软件迭代修改后的代码必须作为全新代码重新进行整体质量评测，耗费大量资源和人力，影响装备批产效率。因此，对装备软件代码变化项影响域的自动化分析方法进行研究是有重要意义的。

2　研究内容

2.1　数据分析

本文研究的装备软件程序为 c 语言工程代码，具有可靠性高、代码较规范和规模较大的特点。代码可靠性高即不需要考虑是否存在程序编译错误，装备软件可以直接按照可运行的标准 c 语言程序处理。代码规范整齐节省了很多数据清洗的时间，代码使用的标识符容易识别，不会出现混乱的情况，这是我们选择基于标识符的代码静态分析方法原因之一。代码规模限制分析算法的时间复杂度，大规模 c 语言代码[9]并不适合使用基于抽象语法树[10]等分析方法，所以我们选择时间复杂度较低的基于标识符的代码分析方法。

本文研究的装备软件程序包含多迭代版本，我们选取已经迭代稳定的版本作为研究数据，该数据不出现大规模代码修改，适合修改特征影响域分析。我们选取一个版本的程序称为代码 A，然后选取代码 A 经过软件质量评测后迭代修改一次后的程序称为代码 B，分析研究代码 A 和代码 B 之间的修改特征提取和修改特征影响域。代码 A 和代码 B 均为多文件 c 语言程序，代码 B 是基于代码 A 的修改版本，修改操作包含增、删、改等操作。迭代情况稳定后，代码 A 和代码 B 不出现连续 10 行以上的增、删操作。代码 A 和代码 B 都符合标准的代码书写规范，我们可以直接考虑行与行之间的修改对比，而不需要考虑空行、符号、断句不规范的问题。

2.2　基于标识符的修改特征提取

本文提取的修改特征主要是迭代前后代码直接变化的变量。本节首先介绍了数据清洗过程中代码行映射关系表的构建，然后介绍了本文使用的标识符表的构建方法。之后我们具体介绍了提取修改特征过程中，先提取代码 A 和代码 B 行对应状态，再依据对应状态提取变量变化情况的方法。最后我们介绍了特殊标识符（for、if、return 等）的修改特征提取。

2.2.1　数据清洗

代码 A 和代码 B 都是标准的装备软件程序，我们不需要考虑代码中出现错误语句、缺失语句等造成程序编译错误的问题，也不需要考虑代码中出现多余语句、无效语句等需要修改源代码的情况。对代码 A 和代码 B 的数据清洗[11]主要是除去代码中的注释、多余的空格和空号。程序数据清洗后会与数据清洗前代码出现行对应变化，为方便修改特征与源代码一一对应，我们构建了数据清洗行映射关系表，该映射关系表记录了代码数据清洗前后代码行对应关系。

2.2.2　标识符表

构建代码 A 和代码 B 的标识符表以用于提取代码修改特征。代码 A 和代码 B 均为同类型的代码，我们可以以相同的方法为代码 A 和代码 B 构建标识符表。我们提取的修改特征主要为变量赋值变化，以修改特征提取需求设计 3 个标识符表：类型表、函数表、变量表。

类型表存储 c 语言程序中变量和函数的类型，在构建函数表和变量表的时候可以以类型表中标识符为索引进行检索。首先在类型表中加入常用的变量类型（int、char、double、float 等）以及函数特有的类型 void，可根据程序加入 long long int、short 等变量类型。然后，我们在类型表中加入 c 语言程序的自定义类型，依次加入结构体、枚举、联合体等类型。以结构体为例，我们以 struct 标识符在代码中检索结构体声明位置，将结构体名记录到类型表中以供后续使用。在后续研究中我们将按照统一方式使用类型表中的标识符，不区分常用类型和自定义类型。

函数表存储程序中定义的函数，在修改特征提取和影响域分析过程中均会使用函数表。我们记录函数名为标识符，并记录函数类型、起始位置、终止位置，在后续需要分析函数变化的时候可以直接检索到函数位置。

变量表存储程序中定义的变量，以类型表中标识符检索变量声明位置，我们记录变量名为标识符，并记录变量类型、变量所属函数、变量声明位置，不考虑未声明直接使用的变量。我们对指针变量和数组亦做特殊处理，然后存入变量表中作为标识符使用。指针变量可以直接在变量名前增加“*”作为标识符，不影响后续使用。数组提前数组名作为标识符存入变量表，记录特殊数组标记以在后续使用时进行特殊处理。结构体等自定义类型变量在记录标识符时不做特殊处理，在后续使用时再进行特殊处理。

2.2.3 修改特征提取

本文提取的修改特征是在代码 A 和代码 B 之间对比提取修改、增加和删除的代码。由于代码 A 和代码 B 不出现大篇幅修改，我们先对比代码 A 和代码 B 行变化情况，再逐行对比代码 A 和代码 B 每行变量变化的情况。

行变化情况包括修改、增加、删除 3 种状态，再考虑行未发生改变的状态，我们设置 4 种行对应状态来描述代码 A 和代码 B 各行对应状态。如果只考虑未发生改变和修改两种状态，代码 A 中每一行均可在代码 B 中寻得，即代码 A 和代码 B 是逐行对应的，并且从代码开始到结束按顺序逐一对应。增加和删除两种状态影响了代码按顺序逐一对应的关系，并且增加或者删除的行可能和未发生改变或者修改的行产生歧义，即两行代码相似度非常高，影响行对应状态。

本文提出了一种基于行相似度匹配的滑动窗口方法，实现了代码行对应状态逐一对应关系检测。由于代码不出现连续 10 行以上的增、删操作，我们设置一个窗口长度为 10 的滑动窗口，将代码 B 逐行放入窗口队列中。然后从上往下逐行将代码 A 与窗口队列中每一行代码匹配，完全一致的代码认为行匹配成功，记录匹配信息的同时更新窗口队列；不完全匹配的代码则记录代码相似度，等待进一步处理。相似度可通过动态规划算法计算两行代码的最长公共子串得到。在代码 A 和代码 B 匹配完成后，我们得到一个已确定未发生改变状态的匹配对应关系，然后我们将其中不完全匹配的对应关系按照相似度确定修改、增加、删除 3 种状态。遍历匹配对应关系中不完全匹配的关系，如果存在相似度高于预设阈值且没有已经匹配到的修改状态对应关系的两行，我们可以将其设置为修改状态对应关系；然后将代码 A 中剩余行设置为删除状态；代码 B 中剩余行设置为增加状态。通过上述方法我们获取了代码 A 和代码 B 按顺序逐一对应的行状态。该方法具体流程如图 1 所示。

在代码 A 和代码 B 行对应状态确定之后，我们通过标识符识别提取代码 A 和代码 B 的修改特征，在本小节我们主要提取赋值语句中代码修改的情况，记录修改内容和修改位置（行号）。首先构建 3 个标识符表，考虑到装备软件程序为多文件 c 语言程序，我们首先建立代码 A 和代码 B 的文件调用关系，即检索每个文件的首部是否调用其他文件。然后我们依次构建每个文件的类型表、函数表、变量表，在每个表构建完成后均要按照文件调用关系进行标识符表通信，将每个文件所引用的标识符加入该文件对应的标识符表中。然后遍历代码 A 和代码 B 提取发生改变的赋值语句中修改的代码，未发生改变状态的代码不存在改变的语句，遍历过程中可以直接跳过。代码 A 中删除状态的语句和代码 B 中增加状态的语句可以直接记录修改特征。修改状态的行需要先进行断句，然后判断对应的赋值语句是否存在或者是否发生变化，不同的分别在代码 A 和代码 B 记录修改特征。经过上述处理，我们在代码 A 和代码 B 中分别记录了其修改特征。

2.2.4 特殊标识符

在本小节我们将提取几类特殊标识符的修改特征。特殊标识符的修改特征与赋值语句修改特征方法不完全相同，我们针对循环语句、条件语句和 return 等特殊标识符提出

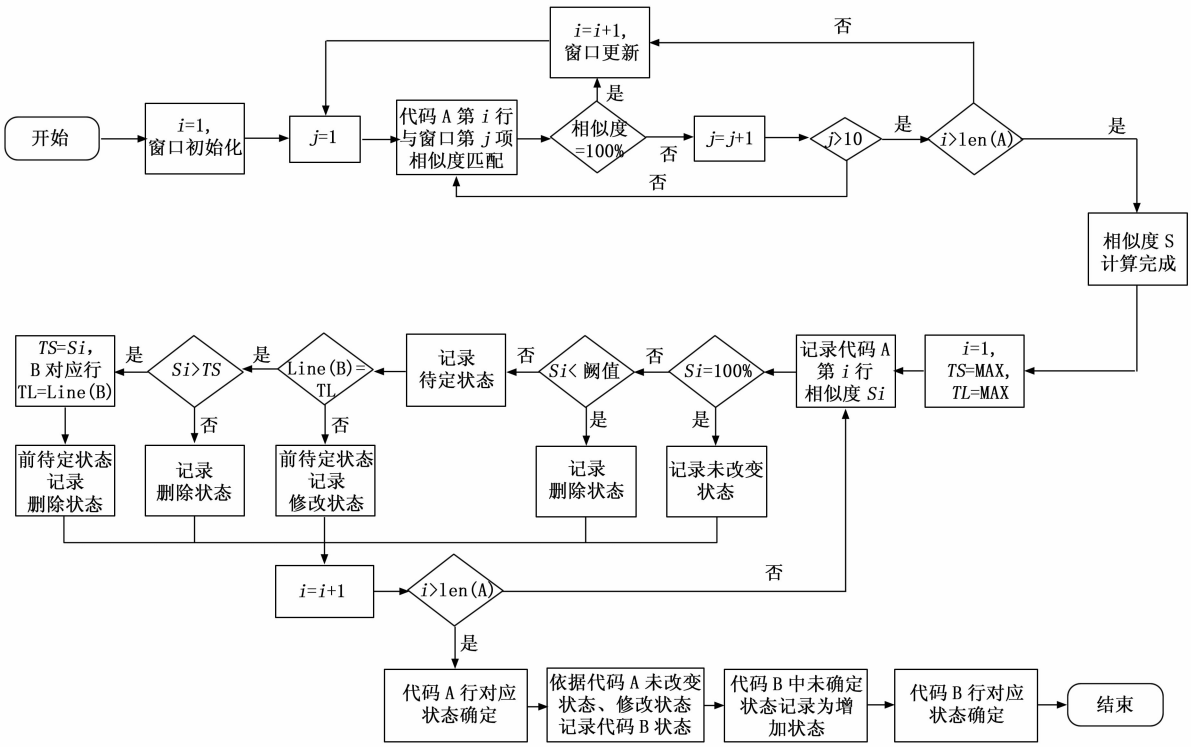


图 1 基于行相似度匹配的滑动窗口方法

了不同的提取方法。

循环语句（标识符 for、while）分为循环体和循环条件两部分，循环体内修改特征按照普通修改特征处理。循环条件的发生变化时，我们记录循环条件发生的变化以及变化的位置。

标识符为 if 的条件语句处理方法与循环语句相似，如果条件变化则记录条件语句修改特征。但是条件语句需要额外考虑在 if 后出现的 else if 和 else 标识符。标识符 else if 处理方法与 if 相同，标识符 else 则需考虑条件语句 if 和 else if 的条件是否变化，如果变化则在 else 处记录条件语句修改特征。

其他特殊标识符种类很多，大部分需要依据代码需求专门处理，我们主要提及两个常用的标识符 return 和 define 处理。标识符 return 主要出现在函数当中，用于函数的返回值，如果函数的返回值发生变化，我们对该函数记录下来该函数名和修改位置。标识符 define 是 c 语言程序中的宏定义，如果宏定义语句出现变化，我们可以将宏定义定义的标识符加入变量表，按照变量进行处理，并直接记录其修改特征。

2.3　修改特征影响域分析

修改特征提取只是对代码变化的直接处理，而影响域分析是基于代码逻辑对代码变化深层影响的分析，所以影响域分析是我们对代码进行深层次的逻辑分析，挖掘装备软件每次迭代对程序造成的影响。本节依次介绍了基层影响域和多层影响域的分析方法，然后提出了基于标识符检测队列的影响域分析方法，该方法可以实现复杂的装备软件影响域分析。

2.3.1　基于修改特征的影响域确定

基于修改特征的影响域是指在修改特征提取过程中未提取到修改特征，但受到修改特征影响的变量，我们主要记录受到影响的标识符（主要为变量名）和位置。本文的修改特征是代码中出现变化的赋值语句、条件语句等，依据不同类型的修改特征需要分别确定影响域。本文修改特征主要记录的是赋值语句中修改的代码和修改位置，影响域为赋值语句中被赋值的变量。

特殊标识符的修改特征需要特殊处理以确定影响域。循环语句修改特征是循环条件的变化，循环体内所有变量均可能因循环条件的变化而受到影响，所以影响域为循环体内所有赋值语句被赋值的变量。条件语句影响域与循环语句一样，只是需要分别考虑 if、else if 和 else 三种。标识符 return 用于函数的返回值，如果函数的返回值发生变化，即检测到 return 后内容发生变化，调用该函数的代码均会产生影响，所以 return 标识符修改特征的影响域为调用该函数的所有赋值语句的被赋值变量。

2.3.2　多层影响域分析

在上小节中我们介绍了基于修改特征的影响域确定，该影响域为修改特征直接确定的第一层影响域，我们称其为基层影响域。代码中可能出现不是修改特征或者基层影响域，但是程序运行过程中受基层影响域影响可能出现变化的变量，我们把该种情况的分析过程称为多层影响域分析。多层影响域我们主要记录多层影响域的标识符（主要为变量名）、位置和分析层数。相较于基层影响域分析，多层影响域分析逻辑相对复杂，且可能出现特殊循环情况。

多层影响域分析是以基层影响域分析为基石，进行整个程序的影响域分析。基层影响域记录的为变量标识符，我们按照变量类型确定基层影响域中每个标识符影响的代码，仿照基层影响域分析第二层影响域，记录变量标识符和位置，然后重复上述方法可得多层影响域。变量分为局部、全局、跨文件变量 3 种类型，变量类型可以通过查询变量表确定。局部变量只需在声明函数里考虑影响域，全局变量则在声明文件中考虑影响域，跨文件变量在所有文件中都可能受到影响。多文件代码全局变量需额外考虑出现跨文件调用全局变量的情况，我们在变量表建立过程中已经进行过跨文件情况的全局变量通信，只需在影响域检测过程中检测引用到的其他文件全局变量即可。

装备软件中出现的干扰影响域分析的循环情况主要包含代码循环和逻辑循环两种。代码循环是在 c 语言程序中出现的变量交互影响，即在某一位置变量 a 影响变量 b，然后在后续代码中变量 b 又影响变量 a，或者多个变量交互影响的情况，在影响域分析过程中需要特殊处理。逻辑循环出现在 c 语言程序中经常使用递归的方法，主要以递归函数为主，在递归函数中很容易出现一变量发生改变，并且传递到下一次递归中，这样会出现无穷多层影响域，也需要特殊处理。对代码中标记已检测的位置并不能很好地解决特殊循环问题，因为 c 语言程序中的递归很多，所以我们提出了一种解决影响域分析过程中特殊循环情况的方法。我们每次进行一层影响域的分析，在后续影响域分析过程中，对代码中已经标记影响域的位置，我们要检测新影响域变量标识符是否与已有影响域记录的标识符完全相同，如果相同则不会开启新一层影响域分析。在下一小节中我们叙述了通过标识符检测队列应用该策略解决特殊循环情况，保证多层影响域分析的可行性。

2.3.3　基于标识符检测队列的影响域分析

装备软件修改特征影响域分析既包含了从修改特征中分析基层影响域，又包含了从基层影响域分析多层影响域。多层影响域的分析逻辑相当复杂，时间复杂度、空间复杂度都比较高，我们提出了一种基于标识符检测队列的影响域分析方法，在降低了时间、空间复杂度的情况下准确分析了多层影响域，适合于装备软件修改特征影响域分析。

基于标识符检测队列的影响域分析用检测队列存储多层影响域分析中改变的变量，从程序运行逻辑的角度进行影响域分析，将多层影响域分析的树形逻辑转化为线性逻辑。依据装备软件 c 语言程序特点，c 语言程序的 main 函数一般是程序启动的入口，并且 main 函数可以控制程序调用其他函数。如果以 main 函数为分析的起始点，可以遵从程序运行的规则，避免盲目检索过程中重复分析不必要片

段的操作。我们依据程序运行逻辑依次检测 main 函数和其他调用函数，在每个函数内进行影响域分析，然后通过检测队列通讯实现函数之间的影响域分析。我们在每个函数内建立了两个标识符检测队列，分别为局部检测队列和全局检测队列。局部检测队列主要存储影响域分析过程中记录的局部变量，并且该函数影响域分析结束后清空该队列。全局检测队列中既要存储全局变量，又要存储跨文件变量。全局检测队列不仅作用于某一函数的影响域分析，还和其他函数的影响域分析有关。

基于标识符检测队列的影响域分析主要包含以下几个步骤。1）遍历代码，依据修改特征确定基层影响域；2）在 c 语言程序中找到 main 函数并加入待检测函数，以 main 函数为函数分析的起始点进行影响域分析，初始化标识符检测队列；3）遍历待检测函数中函数的代码，将基层影响域中的变量加入标识符检测队列；同时依据标识符检测队列中标识符判断该行代码是否受影响，受影响则进行影响域分析将结果加入标识符检测队列，此即为多层影响域分析。为解决循环问题，我们在将变量加入标识符检测队列时，先检测标识符检测队列中是否已有该变量，重复则不加入；4）在遍历代码过程中检测到调用函数的情况，将调用函数加入待检测函数，并将全局检测队列传递到该函数影响域分析过程中使用的全局检测队列中，重复第三步。为解决循环问题，我们记录每个函数已经分析过的标识符检测队列为历史记录，如果再次调用该函数时，标识符检测队列与历史记录相同，则不再重复对该函数检测。基于标识符检测队列的影响域分析具体流程如图 2 所示。

3 实验结果与分析

本节叙述了本文的实验设置和结果。本文进行了两个实验，功能实验以自行设计的代码为测试数据测试了基于标识符修改特征的提取方法和基于标识符检测队列的影响域分析方法，展示了本文方法的检测结果；公开数据集实验采用公开数据集，验证了本文提出的方法在装备软件质量评测工作中的有效性。

3.1 功能实验

功能实验分块测试了基于标识符的修改特征提取方法和基于标识符检测队列的影响域分析方法。修改特征分别设立了增加、删除、修改 3 种，影响域分析分别设立了赋值语句、条件语句、函数调用、跨文件调用 4 个模块的测试。功能实验数据集为自行设计的 c 语言程序，分为版本 1 和版本 2，每个版本有 main.c 和 test.c 两个文件，文件详细内容见附录。运行检测程序，实验运行时间不超过 1 s，程序运行结果如图 3 所示。

图 3 中 thisIsOld 和 thisIsNew 分别为版本 1 和版本 2 的分析结果，因代码修改中存在增加和删除，版本 1 和版本 2 的分析结果不一一对应。实验输出为行号和影响域内容（下文我们用｛行号，修改内容｝表示），分析结果的基层影响域如图 3（a）和多层影响域如图 3（b）。在 main.c 的 test1 函数中设置了赋值语句修改特征，依据修改特征实验得到基层影响域｛7，a｝和多层影响域｛8，b｝｛9，c｝。在 main.c 的 test2 函数中设置了条件语句修改特征，依据修改特征实验得到基层影响域｛19，temp｝和多层影响域｛21，result｝。在 main.c 的 test3 函数中设置了修改特征影响函数调用过程中的形参和 return，依据修改特征实验得到基层影响域｛34，x｝和受形参影响的多层影响域｛26，result｝、受 return 影响的｛36，z｝。在 main. c 的 test4 函数中设置了跨文件调用 test.c 的函数，依据修改特征实验得到基层

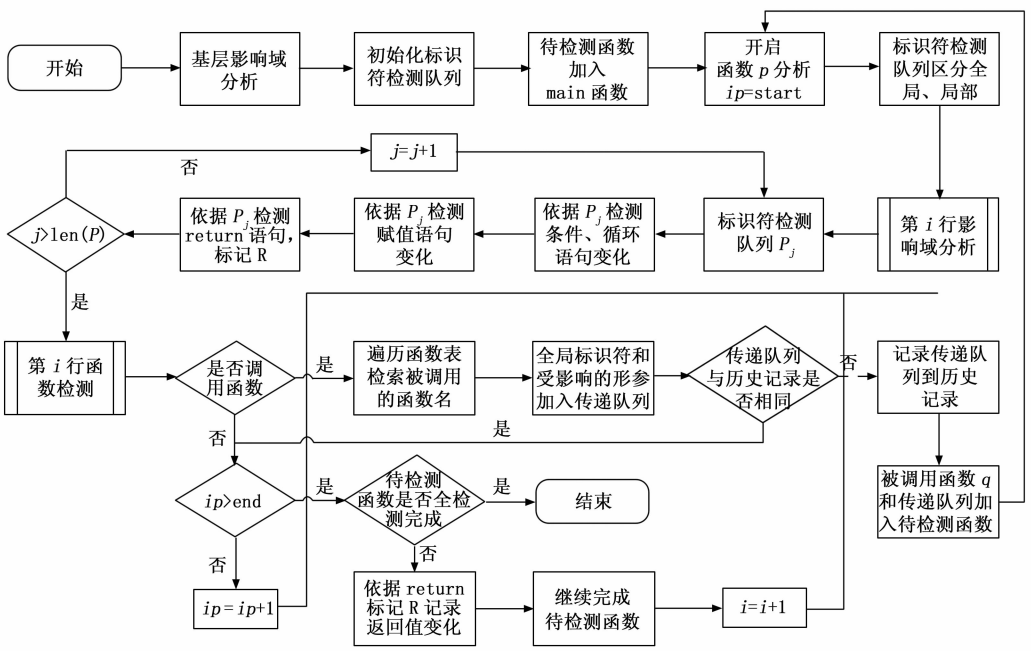
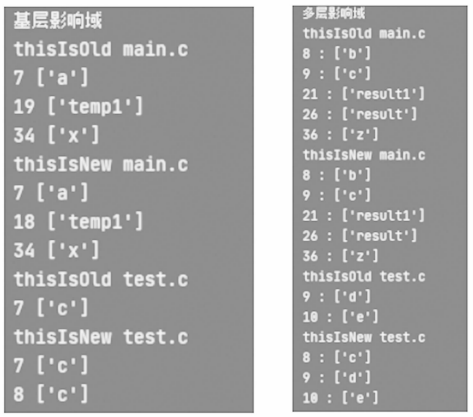


图 2 基于标识符检测队列的影响域分析



（a） 基层影响域　　（b） 多层影响域

图 3　功能实验结果

影响域 {7，c} {8，c} 和多层影响域 {8，c} {9，d} {10，e}。

在功能实验中我们检测出了所有设置的修改特征，并且分析出可能受影响的变量标识符，准确率达 100%。依据上述功能实验结果我们得出结论，以本文提出的修改特征提取和影响域分析方法设计的检测程序可以实现赋值语句、条件语句、函数调用、跨文件调用等情况下 C 语言程序的影响域分析。

3.2　公开数据集实验

公开数据集实验以公开数据集 https：//github.com/antirez/sds 中 sds.c、sds.h、sdsalloc.h、testhelp.h 为版本 3 程序，设置少量修改为版本 4 程序，对版本 3 和版本 4 代码进行影响域分析。版本 3 程序修改具体内容如表 1 所示运行检测程序，运行时间不超过 1 s，程序运行结果如表 2 所示。

表 1　公开数据集修改表

	行号	原内容	修改内容
修改 1	740	size_t newlen, len = sdslen(s)	size_t newlen, len = sdslen(s)+1 000
修改 2	796	l1 = sdslen(s1)	l1 = sdslen(s1)+1
修改 3	1 134	sds x = sdsnew ("foo"), y	sds x = sdsnew ("foo")+1 111, y
修改 4	1 259	int step = 10, j, i	int step = 10 000, j, i

表 2　公开数据集影响域分析结果

	基层影响域	多层影响域		错误识别
		sds.c	sds.h	
整体	{740,len}{796,l1} {1134,x}{1259,step}	38	2	6
修改 1	{740,len}	9	0	4
修改 2	{796,l1}	3	0	0
修改 3	{1134,x}	35	2	6
修改 4	{1259,step}	13	0	0

在公开数据集实验中，我们以公开数据集为数据测试了基于标识符的修改特征提取方法和基于标识符检测队列的影响域分析方法。公开数据集实验基层影响域共检测到 4 个，多层影响域共检测到 38 个，其中错误识别数量为 6 个，实验结果准确率为 85.7%。对实验结果分析，实验中错误识别影响域的原因是检测程序的检测规则设定不完善，可以按照 c 语言规则继续完善。公开数据集实验验证了本文提出的方法在大型工程 C 语言程序中的有效性。

4　结束语

软件质量评测的自动化可以大幅提高评测效率，是未来装备软件质量评测发展的主要方向。本文提出的基于标识符的修改特征提取方法和基于标识符检测队列的影响域分析方法自动化实现了装备软件修改特征提取和影响域分析，在质量评测领域提出了一种有效的影响域分析方法，推进了装备软件质量评测自动化。在后续的工作中，将软件影响域分析与其他自动化分析工具相结合，实现装备软件质量评测的全面自动化，是我们后续研究的重点内容。

参考文献：

[1] 贺英杰. Testbed 在装备软件测试中的静态分析应用 [J]. 电子世界，2018，549 (15)：105.

[2] 张　峰，孙陇平，陈　伟. 基于测试数据的装备软件质量评价 [J]. 舰船电子工程，2019，39 (7)：177－182.

[3] 但凝云. 常用的自动化软件测试工具评估方法 [J]. 电子技术与软件工程，2018 (3)：57－57.

[4] XUAN J，CORNU B，MARTINEZ M，et al. B－refactoring：automatic test code refactoring to improve dynamic analysis [J]. Information and Software Technology，2016，76：65－80.

[5] SESHAGIRI P，VAZHAYIL A，SRIRAM P. AMA：static code analysis of web page for the detection of malicious scripts [J]. Procedia Computer Science，2016，93：768－773.

[6] FU D，XU Y，YU H，et al. WASTK：a weighted abstract syntax tree Kernel method for source code plagiarism detection [J]. Scientific Programming，2017 (1)：1－8.

[7] 何　平，胡　勇. 一种基于本地代码特征的 Android 恶意代码检测方法 [J]. 信息安全研究，20184 (6)：511－517.

[8] 陈　凯，刘建宾. 代码标识符属性特征向量相似度检测技术研究 [J]. 福建电脑，2016，32 (1)：4－7.

[9] 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]. 西安：西安理工大学，2019.

[10] FAUZI E，HENDRADJAYA B，SUNINDYO W D. Reverse engineering of source code to sequence diagram using abstract syntax tree [A]. International Conference on Data & Software Engineering [C]. IEEE，2016.

[11] 郝　爽，李国良，冯建华，等. 结构化数据清洗技术综述 [J]. 清华大学学报（自然科学版），2018，58 (12)：1037－1050.