SCIENCE CHINA
Information Sciences

• LETTER •

Improving Actionable Warning Identification via the Refined Warning-inducing Context Representation

Xiuting GE, Chunrong FANG*, Xuanye LI,
Quanjun ZHANG, Jia LIU*, Zhihong ZHAO & Zhenyu CHEN

The State Key Laboratory for Novel Software Technology, Nanjing University 210093*, China*

Citation

1 Appendix A

In Appendix A, the new warning slicing criterion *WarningSC* is designed to replace the original slicing criterion in Joana.

$$WarningSC = (warningStm,\ warningVar,\ warningS) \quad (1)$$

In Equation (1), *warningStm* is statements with the corresponding warning line numbers. *warningVar* is variables in *warningStm*. *warningS*, the class/method containing *warningStm*, is the end point of program slicing in Joana. In detail, as for a warning that locates in the method, our approach sets this method as *warningS*. As for a warning that does not locate any method, our approach sets the class containing this warning as *warningS*.

2 Appendix B

In Appendix B, the detailed process of our proposed adjustment algorithm is shown in Algorithm 1. Our approach takes *SC* (i.e., the source code in the class/method containing a warning) and *LineNums* (i.e., the source code line numbers obtained by Joana) as inputs. It is noted that as for a warning that locates in the method, *SC* is the source code in the method containing a warning. As for a warning that does not locate any method, *SC* is the source code in the class containing a warning. The output is *Res* (i.e., the refined warning-inducing context). Our approach first copies *SC* to *Res* and extracts all *Nodes* by using JavaParser[1] to hierarchical traversal *Res* (lines 1-2). As for $node \in Nodes$, our approach performs the following processing for *node*. Specifically, if *node* is the root node and the *CatchClause* node, our approach terminates to traverse *node*. Otherwise, our approach directly proceeds to the next traversal (lines 4-6). Although our approach sets the program slicing scope to *SC* in Section **??**, *LineNums* could still bring statements outside *SC* due to the interprocedural analysis of Joana. As such, our approach removes statements irrelevant to *SC* in *LineNums* (lines 7-16). In particular, our approach individually handles the *SwitchEntry* and *CatchClause* nodes. When *Lines* only contains the *SwitchEntry* node, our approach judges whether three conditions (i.e., (1) the parent node of *node* exists, (2) the parent node of *node* is *SwitchEntry*, and (3) *node* is the first child node in all children nodes of the parent node of *node*) are satisfied. If satisfied, the *flag* of *node* is set as *TRUE* (lines 17-20). If *node* is not a *BlockStmt* node (e.g., *Parameter*) of *CatchClause*, our approach retains *node* along with *CatchClause*. To avoid accidental deletion, our approach recursively searches the parent node of *node* (lines 21-29). Specifically, our approach copies *node* to *temp*. If three conditions (i.e., (1) *temp* is *Parameter*, (2) the parent node of *temp* exists, and (3) the parent node of *temp* is *CatchClause*) are satisfied, the *flag* of *temp* is set as *TRUE*, and our approach breaks out of the loop. After that, our approach removes *node* that are marked *FALSE* (lines 30-32). Finally, our approach returns *Res*, which is the refined warning-inducing context.

* Corresponding author (email: fangchunrong@nju.edu.cn, liujia@nju.edu.cn)

[1] https://javaparser.org/

Algorithm 1 The adjustment algorithm for the warning-inducing context completion

```
Input: $SC$ (the source code in the class/method containing a warning); $LineNums$ (the source code line numbers obtained by Joana);
Output: $Res$ (the refined warning-inducting context).
$Res = SC$;
$Nodes$ = getHierarchyTraversal($Res$) via JavaParser;
for each $node \in Nodes$ do
    if isExisted(getParentNode($node$)) and $node$ == $CatchClause$ then
        return;
    end if
    $startLine$ = getStartLine($SC$);
    $endLine$ = getEndLine($SC$);
    $flag$ = FALSE;
    # Remove the statements outside $SC$
    for $line$ in $LineNums$ do
        if $line \geqslant startLine$ and $line \leqslant endLine$ then
            $flag$ = TRUE;
            break;
        end if
    end for
    # Handle the SwitchEntry node
    if isExisted(getParentNode($node$)) and getParentNode($node$) == $SwitchEntry$ and isFirstChildNode(getParentNode($node$)) then
        $flag$ = TRUE;
    end if
    # Handle the CatchClause node
    $temp = node$;
    while isExisted(getParentNode($temp$) do
        $temp$ = getParentNode($temp$);
        if $temp$ == Parameter and isExisted(getParentNode($temp$) and getParentNode($temp$) == $CatchClause$ then
            $flag$ = TRUE;
            break;
        end if
    end while
    if !$flag$ then
        $node$.remove();
    end if
end for
return $Res$
```