IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received April 17, 2018, accepted June 7, 2018, date of publication July 12, 2018, date of current version May 24, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2853153

Scenario Oriented Program Slicing for Large-Scale Software Through Constraint Logic Programming and Program Transformation

SHENGBING REN AND MENGYU JIA
School of Software, Central South University, Changsha 410075, China
Corresponding author: Mengyu Jia (154711013@csu.edu.cn)

This research work is supported only by the Central South University Graduate Research Innovation Project under Grant 2017zzts615.

ABSTRACT Program slicing, as a technique of program decomposition, is widely used in the field of program testing, model checking, software verification, symbolic execution, and other fields. However, the traditional approaches of program slicing tend to produce too large slices and the static program analyses are hard to be precise enough. Scenario-oriented program slicing, which considers the actual usage of software, gives a more precise perspective of program slicing. In this paper, we propose an approach of scenario-oriented program slicing, which combines constraint logic programming and program transformation. According to the observation that the output of a program transformation is a semantically equivalent program where the properties of interest are preserved, we can apply a sequence of transformations, more powerful than those needed for program specialization, refining the slicing to the desired degree of precision. And constraint logic programming has been shown to be a powerful, flexible formalism to reason about the correctness of programs. The novel contributions of this paper are as follows: 1) converting the problem of program slicing into program transformation and retrieval; 2) presenting a set of constraint handling rules for scenario-oriented program slicing in constraint logical programming programs; and 3) deriving a scenario-oriented program slicing algorithm. The method of scenario-oriented program slicing has been implemented and we have demonstrated its effectiveness and efficiency on three open source software projects in GitHub.

INDEX TERMS |Constraint logic programming, large-scale software, program slicing, program transformation, scenario.

I. INTRODUCTION

Software applications will play more and more important role in our daily life. With the continuous expansion of software applications and the recognition of open source concepts, software code is growing rapidly. And the scale and structure of software architecture are more and more complicated. The analysis for large-scale software tends to become infeasible for their state spaces more easily becoming too big [1].

Program slicing, as an efficient way of model checking, software verification, symbolic execution and reverse engineering and other fields, is an important decomposition technique for large-scale programs [2]. Although traditional methods of program slicing can reduce the size of final slices, they either highly depend on executable system and representative test cases, or have serious efficiency problems in applying into large-scale industrial programs [3].

Requirement plays a key role in many development processes, and the value is generated only if the system is actually used. It is much better to focus on how the system will be used than on what functions or features it will offer. Scenario provides its focus by concentrating on how the system will be used to achieve a specific goal for a particular user [4]. Across the set of slices all the activities could go on in parallel and the scenario slice is the most important element of Use-Case [5]. As a program is often tangled with lots of different scenarios, we believe that slicing program under a specified scenario may achieve better precision [6]. A scenario can exclude some program paths from being passed and thus exclude many program states from being reached. These scenarios capture a subset of the program's functionalities and often permit developers to perform an ad hoc form of program slicing. We believe that combing program decomposition and

scenario oriented program slicing may provide a promising support for understanding large-scale programs [7].

As shown by a number of applications, program transformation, as an operation which can be applied to a program to generate another equivalent program (provided any given applicability conditions are satisfied), is a powerful technique for the development and optimization of large-scale programs [8]. It is possible to easily take into account alternative operational semantics definitions for modeling additional language features. Since the output of a program transformation is a semantically equivalent program where the properties of interest are preserved, we can apply a sequence of transformations, more powerful than those needed for program specialization, thereby refining the analysis to the desired degree of precision [9]–[12].

Constraint logic programming, also named constrained Horn clauses, has been shown to be a powerful, flexible formalism to reason about the correctness of programs [13]. Constraint logic program is an artificial intelligence-based constraint satisfaction model with the ability of both logical programming and constraint solving. It can be used to represent attributes, programs and execution. In recent years, the declarative language that separates the logical components from the control components has been widely studied. Simultaneously, constraint logic programming algorithms are mainly divided into two parts: control and logic, which are easy to implement in parallel. The information on the constrained network can be propagated at the same time and used to store solving problems. The logic between the information used at any time and any way the information is used. And constraint logic programming has been applied into model checking of both finite and infinite state systems [14].

Inspired by the preliminary work of the article [15] and our previous work in representation, storage and retrieve for large-scale software, in order to address the problems of the state space exploration and the precision for analyzing large-scale software, a method named scenario slicing is proposed which combines and extends the ideas which were developed in the fields of constraint logic programming and program transformation. There is no restriction on the languages with which our approach can be used, but for the simplicity we have only explored its use with java. The actual novelty of the work is that scenario is used as a property to reduce the size of the state space providing a more precise result. The idea is to render properties as scenarios and exploit their locality to reduce the size of the verification space. In order to use the result of the scenario oriented program slicing for program verification, the slicing is measured by the basic unit of constraint logical fact instead of the program statement. The objective of the transformation is to derive a new constraint logical program. Our method is based on transformations of constraint logical programs that preserve the least model semantics related with certain scenario.

The rest of this paper is organized as follows. Section II discusses the related work, highlighting the original contributions of this paper. After giving the motivating example in Section III, we introduce the overall approach of the proposed solution in section IV. And section V presents the experimental implement and evaluation. Section VI concludes summarizing the contributions of this paper, and outlining our future research plans.

II. RELATED WORKS

Weiser [16] introduced static program slicing in 1982 which has opened the research boom of program slicing. Recently, the method of calculating the static program slice includes the data flow equation and the dependency graph relationship. Though the specific data values doesn't need to be considered, the result of static program slicing are often too large. Scenario is usually described as UML sequences diagram. Generally UML sequences diagram works at conceptual level, and doesn't consider implementation details. Even so, it provides a very interesting means for program analysis.

Kumar *et al.* [3] proposes a scenario oriented program slicing method which takes the user specified scenario into account and finds all the program parts relevant to a special computation under the give execution scenario. Although the actual uses of the user are taken into account, this paper lacks the necessary theoretical basis.

With the maturation of the constrained logic theory, the constraint solution accelerates the development of the program slicing theory. Hofer and Wotawa [17] and Hofer and Wotawa [18] applied the CONBAS algorithm, which combines constraint solving together, into reducing the size of dynamic slicing with the average reduction of the number statements of more than 28% in the resulting slice compared with relevant slice However with the apparent weakness of its complexity, CONBAS can't be applied for large-scale software, and dynamic slicing, which takes test case into account, can only get slice results related with a specific input. Particularly, ignoring the object-oriented part and method calls, the CONBAS algorithm can only handle Integer and Boolean data types.

III. A MOTIVATING EXAMPLE

Constraint logic programming, also known as constrained Horn clauses, can be used to determine the correctness of a program. Constraint logic programming has been successfully applied to perform model checking for finite and infinite systems, and it turns out that constraint logic programming can be used to express symbolic execution and invariants of imperative programs.

A constrained logic program is an artificial intelligence-based constraint satisfaction problem model that combines the power of logic programming and constraint solving. It can be used to represent attributes, procedures, and execution. In recent years, the declarative language that separates the logical components from the control components has been widely studied. The constrained logic programming algorithms are mainly divided into two parts: control and logic. It is easy to implement in parallel, and the information on the

constrained network can be simultaneously propagated for storage problem solving. The logic between the information used at the time and the way the information is used.

The program transformation is generally to translate the program of easy-to-understand but not-efficiently-executed into a program with higher execution efficiency, and to ensure the correctness of the conversion process and the conversion result. The purpose of program transformation is to generate a new CLP program that is related to a specific scenario, and to improve the analysis efficiency of the program while ensuring the correctness of the program. Our method based on the CLP program transform retains minimal model semantics. Many applications show that program transformation can generate another equivalent program with the same semantics as the source program. It is an effective method for the development and optimization of large-scale complex programs. Since the output of the program transformation is a semantic equivalent program that preserves the properties of interest, we can apply a series of transformations to refine the analysis to the required precision

IV. SCENARIO ORIENTED PROGRAM SLICING

Since the problems, which includes the explosion of state space and the difficulties in balancing accuracy and scalability for program slicing, have limited the credibility of large industrial software, the focus of our work is to design an approach to effectively and effectively realize scenario oriented program slicing for large-scale software through constraint logic programming and program transformation. In this section, we will introduce the overall method of scenario oriented program slicing from two steps of transformation, two CHRs and an algorithm for scenario oriented program slicing in detail.

A. TRANSFORMATION 1: GENERATION OF CLP BASED SOFTWARE NETWORK

Profile, whose suffix is .xml or .yml or .fxml etc, plays an important role in some web project, even have a great influence on the control flows, and the profile is supposed to be taken into account when comes to scenario oriented program slicing. Firstly, in order to model the source code for large-scale software, we form the architecture of generating constraint logical programming based software network as follows in Figure 1.

The scheme of generating constraint logical programming based software network in shown in Figure 1. There are three components in the architecture, naming XML Parser, File Parser and Redis DB. Firstly the config file belonging to source code and information flow extracting from scenario can be putted into XML Parser to get Java object, and then the java object, combining with source and resource file, can be used as the input of file parser to get CLP facts stored into Redis database.

And then we define the CLP facts related with large-scale software as follows, the definitions of related constraint logical facts are depicted in Figure 2.

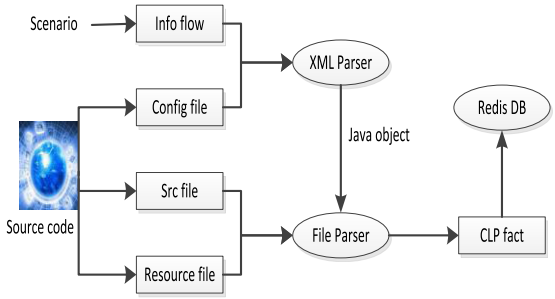

FIGURE 1. Architecture of CLP based software network.

Definition of related constraint logical facts
1.package(packName).
2.class(packName,className).
3.method(packName,className,methodName,reType,visibility,param,enteryPointId).
4.entryPoint(entryPointId,methodName_row_col).
5.instr(methodName_row_col,ite(expr),methodName1_row1_col1,methodName2_row2_col2).
6.instr(methodName_row_col,while(expr),methodName1_row1_col1,methodName2_row2_col2).
7.instr(methodName_row_col,assign(expr),methodName1_row1_col1).
8.instr(methodName_row_col,call(expr),methodName1_row1_col1).
9.instr(methodName_row_col,goto(expr),methodName1_row1_col1).
10.instr(methodName_row_col,halt).

FIGURE 2. Definition of related constraint logical facts.

The definition 1 shows that there is a package named packName. Definition2 specifies that there is a class named className that belongs to the package of package. Definition3 means that there is a method named medthodName that belongs to the class of className in the package of packName, where the retType, visibility, param, entryPointId means the return type, access permission, parameters and the identity of entry point of the method respectively. Definition 4 specifies that there is an entry point of method where the entryPointId and methodName_row_col means the identity of entry point and the corresponding entry point depicted by method name, the number of row and line. Similarly, the definition in the range of 5 and 10 corresponds to the statement of branch, loop, assignment, call, goto and halt separately.

B. TRANSFORMATION 2: REMOVAL OF UNRELATED SCENARIO

Slice criteria, proposed by M. Weiser firstly, is denoted as SC = <n, V>. And particularly static slice criteria can be denoted as SCsta = <n, V>. A dynamic slice criterion is denoted as SCdyna = <n, V, I_0 >. The notations in the slice criteria denote as follows. The variant n is one of the program execution points, V is a collection of variables, and I_0 is a specific input in programs.

Similarly, we define scenario criteria as SCscenario = <enteryPointList, MedthodList>, which can be obtained by extracting from scenario model where entryPointList is a list of entry points relevant to the given scenario, and the medthodList is a list of method related with the given scenario.

Scenarios are concerned with situations that might reasonably occur in the operations of the system. A scenario typically presented by a sequence diagram (or a collaboration diagram) in UML. The core element of a scenario is the message flow.

Firstly we denote formal definition of sequence diagram as follows:

Definition 1: Let M be a message, let O be an Object, the sequence diagram denoted by SD can be noted as SD = < M, O>.

Definition 2: Let msg_id be message identity, the method which is called by the message is denoted by the notation of Method, fromObj is a sender of the message, toObj is a receiver of the message and the notation of guard denotes optional for guard condition, a message Mi (Mi ∈ M) can be denoted as Mi = ⟨ msg_id, Method, fromObj, toObj, [/guard] ⟩.

Definition 3: The method called by message named as Method can be denoted as Method = ⟨ medthod_name, return_type, visibility, pre_condition, post_condition, parameter ⟩.

CHR offers a rule-based programming style to express constraint simplification and constraint propagation which includes simplification, propagation and merging. Easily, with the above of three definitions we can conclude the following constraint handling rules for scenario in Figure 3.

```
Constraint handling rules for scenario
:- lib(ech).
SD(I) <=> M(I),O(I).
M(I) <=> msg_id(I),Method(I),fromObj(I),guard(I).
Method(I) <=> ,methodName(I),reType(I),visibility(I),precondition(I),postCondition(I),param(I).
```

FIGURE 3. Constraint handling rules for scenario.

In figure 3, we import the constraint's package named ech firstly to insert the constraint handle rules. In order to only preserve the needed constraint lastly, we apply the simplification into constraint handling rules for scenario.

Rule 1 (Removal of Constraint Facts Unrelated With the Given Scenario): While there's a constraint fact F which contains an unsatisfiable constraint related with the given scenario in TransfC do: TransfC = TransfC-{F}, which can easily be realized by retrieving from CLP based software network generated by transformation1, to generate scenario related file suffixed by pl.

Rule 2 (Simplification): When encountered with conditional expression and assignment statement, simplification tips are as follows in Figure 4.

The simplification tips depicted in Figure 4 can easily be implemented, and it should be applied with priority from left to right.

Simplication tips

1.A ∩ true => A.

2.A ∩ A => A.

3.A ∪ true => A.

4.A ∪ (A ∩ B) => A.

5.(A ∩ B) ∪ (A ∩ C) => A ∪ （B ∩ C）.

6.A + B – B => A.

7.A * B / B => A.

FIGURE 4. Smplification tips.

Our algorithm depicted in Figure 5 is used to compute control dependences based on constraint facts obtained from transformation, whose body contains not just control dependences, but also data dependences, which are extremely vital for program slicing.

This algorithm for scenario oriented program slicing can be applied into after simplification and generating scenario related file which is suffixed by .ecl. Where the clause1 builds the constraint among package, class and method, clause2 denotes the constraint between method and entry point, taking the different number and type of parameter into account, clause3 are used to express the constraint between entry point and statement, whereas clause4 constructs the constraint between statement and statement, clause5 builds the constraint of branch and clause6 shows the constraint of assignment and call statement.

V. IMPLEMENTATION AND EVALUATION

In this section, we will describe our prototype implementation firstly, and experimentally evaluate our approach by case studies based on three open source software.

A. IMPLEMENTATION

Based on the approach for scenario oriented program slicing proposed in section II, we can conclude our architecture in Figure 6.

In figure 6, there are three components, including parser, transformarion 1 and transformation 2. The architecture of the approach is realized by JAXB, ANTLR, Redis, ECL^{i} PS^{e}, and python. JAXB (Java Architecture for XML Binding), which combines the benefits of DOM parser and API for XML Parser, is used as an important part of XML Parser proposed in Figure 1 which can be used to convert valid XML file into java object that are instances of the classes by using unmarshalling process. ANTLR is used to parser source

Algorithm: Scenario slicing
Input: constraint facts of source code, scenario constraint
Output: scenario slice

```
% P : package, C : class, F : method, PP : entryPoint,
% L1: methodName_row_col1，L2:methodName_row_col2
:- lib(ic).
1.f(P,C,F,E,PP) <=> package(P),class(P,C),meth(P, C, F,
PP).

2.meth(P, C, F, PP) <=> (method(P, C, F, ret, visib,[], PP);
                         method(P, C, F, ret, visib, [P1], PP);
                         method(P, C, F, ret, visib, [P1,P2], PP);
                         method(P,C,F, ret, visib, [P1,P2,P3],PP)),
                        entry(PP,E).

3.entry(PP,E) <=> entry_point(PP,E),inst(E).

4.inst(E)<=>write(E+"\n"),
(instr(E,ite(Expr), L1,L2), exe(Expr,L1,L2);
instr(E,while(Expr),L1, L2),exe(Expr,L1,L2);
instr(E,assign(Expr),L1),assig (Var,Val),inst(L1)).

5.exe(Expr,L1,L2) <=> Expr,inst(L1);
                      not Expr,inst(L2).

6.assig(Var,Val) <=> ((assign(Var,multiply(I,J));
                       assign(Var,div(I,J));
                       assign(Var,plus(I,J));
                       assign(Var,minus(I,J));
                       assign(Var,neg(I));
                       assign(Var,dec(I));
                       assign(Var,inc(I))
                       assign(Var,call(Fun)),meth(P,C,Fun, PP)),
                            expr(I,J,Var)).
```

FIGURE 5. Algorithm for scenario oriented program slicing.

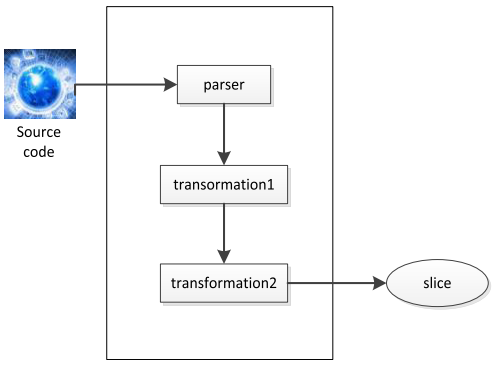

FIGURE 6. Architecture for the scenario oriented program slicing.

code which can generate lexical analyzer, parser, listener etc which are related with certain program language using related grammar automatically, and the exaction of source code information is written by python for transform source code to software network with constraint logic programming. Redis, as one of the NoSQL database, is very suitable for solving the problem for its rich data types, high speed and performance in concurrent read and write, which can easily be used to store and select the results of software network. ECL^{i} PS^{e}, as an open-source software system for the cost-effective development and deployment of constraint logic programming applications, can be applied into scenario oriented program slicing. Consequently, JAXB and ANTLR are applied into the component of parser, Redis is the core element in transformation1, and ECLiPSe is used to realize the component of transformation 2.

B. EVALUATION: CASE STUDYS

In this section we will evaluate our prototype implementation on three open source projects written by Java including SoundSea (https://github.com/sacert/SoundSea), renren-generator (https://gitee.com/babaio/renren-generator) and commons-pool (https://github.com/apache/commons-pool). SoundSea, as a music platform, is one of the top ten Java program worth your attention in Github. renren-generator can generate source code including entity, XML, dao, service, js and so on. And commons-pool is the Apache Commons Object Pooling Library.

And firstly, we evaluate our transformation1 naming generation of constraint logical programming based software network by the tool of SoNet in Figure 7.

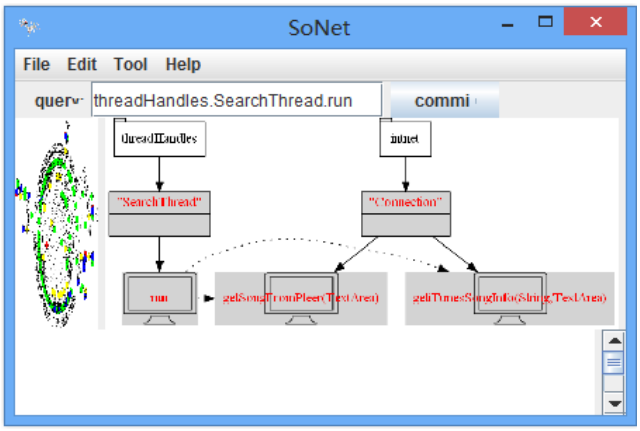

FIGURE 7. GUI of SoNet.

SoNet can be used in two ways including command line mode and GUI for convenience. And in GUI the window of SoNet composed of components of menu bar, text field of input, display panels of global and sliced views and text area of error output, shown in Figure 7. In order to make the usage easier, the menu item of File/Import can be clicked in GUI and file chooser is used to select directory of source code regarding open source software. And then Parser module can be activated to extract the primary information including package, class, function, variant and relations among them. And then the primary information can be stored into Redis by calling storage module for cluster storage. Consequently the global view can be displayed in the left of GUI by invocating the layout of twopi provided by Graphviz in visualization

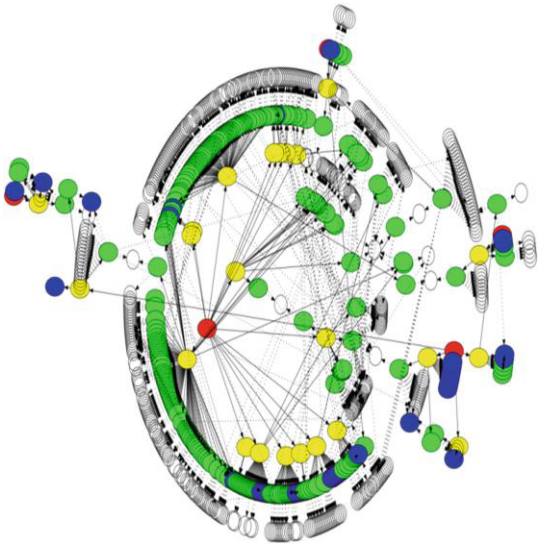

FIGURE 8. Global view related SoundSea.

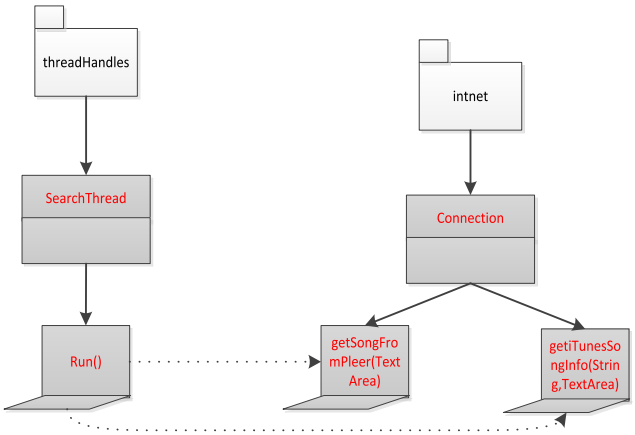

FIGURE 9. The partial view based on the method of threadHandles.SearchThread.run in SoundSea.

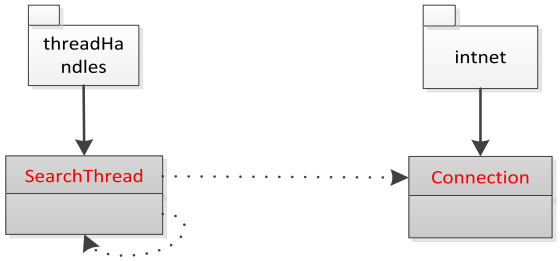

FIGURE 10. The partial view based on the class of threadHandles.SearchThread in SoundSea.

FIGURE 11. The partial view based on the package of threadHandles in SoundSea.

module shown in Figure 8. Particularly, the color of red, yellow, green and blue represent the nodes of package, class, function and variant respectively. And in order to distinguish the functions from all kinds of functions, the library functions, which play a key role in software and don't exist in Redis however, are denoted by circle, while others are denoted by ellipse.

In order to obtain the information related with a given keyword, the commit button can be clicked when a keyword putted into the text field. And then the transformation1 realized by SoNet will be invocated to search constraint logic facts and partial view stored in Redis. Consequently the constraint facts will be outputted suffixed by .ecl and partial view will display in the right of GUI respectively. For the purpose of distinguishing the nodes in different granularity, in the partial view related to a given keyword, the tab and record are used to represent package and class, and self-defined shape of computer and cylinder are applied into the node of function and variant separately.

Further we visualize the access rights including public, private, protected and friendly using different font colors such as red, gold, black and blue. And the relationships of inclusion and reference are denoted by the line of solid and dotted separately. Taken the three keywords in different level including threadHandles.SearchThread.run (a function), threadHandles.SearchThread (a class) and threadHandles (a package) as example, the partial views are depicted in Figure 9-11 respectively. Especially, the information of errors and exceptions are outputted in text area under the GUI.

In order to evaluate our prototype implements, we conduct the experiments using nine scenarios from the three open source softwares mentioned above, and consequently obtain the result for scenario oriented program slicing depicted in Table 1. And in Table 1, we present the following data: the name of the program (Program), sum time of generating constraint facts (SumTime), the number of lines of code (LOC), the number of constraint facts (SumFacts), the serial number of scenario (Scenario), the number of constraint facts obtained by scenario slicing, the time of scenario slicing. Obviously, we can observe that SumFacts usually is more than LOC since a line of code may embody more than one constraint facts.

Compared with generating constraint facts, the time of obtaining sliced facts is smaller, and when comes to scenario oriented slicing, the process for generating constraint logical facts only need once for a certain open-source project program, you can slice for different scenarios to reduce time consumption.

Particularly, we can observe that it is not a serious positive correlation relationship between the slicedFacts and SliceTime, since there are constraint logic facts including package, class, entry point and the statement of branch,

TABLE 1. Result for scenario oriented program slicing.

program	sumTime (s)	LOC	sumFact	scenario	sliceFact	sliceTime (s)
SoundSea	1.93	10023	10335	1	215	0.09
				2	321	0.04
				3	364	0.07
Renren-generator	2.32	27533	29842	1	353	0.06
				2	244	0.02
				3	211	0.08
Common s-pool	2.42	28791	32646	1	224	0.03
				2	422	0.07
				3	324	0.05

iterate, goto and halt in the library of constraint logic facts, the capacity of the scenario oriented program slicing for all kinds of constraint logical facts are not always actually equal.

Accuracy and recall are two metrics which are widely used in the field of information retrieval and statistical classification. Rather than take mean of precision and recall, F1 score (also F-score or F-measure) uses the harmonic mean. The higher the F1 score, the better the predictions. Similarly, in order to verify and validate the results of slice and obtain the threshold of slice, we use four parameters in Figure 11 where A is the number of constraint facts that belong to the scenario which is obtained by slicing, B is the number of constraint facts that don't belong to the scenario which is obtained by slicing, C is the number of constraint facts that belong to the scenario which can't be obtained by slicing, and D is the number of constraint facts that don't belong to the scenario which can't be obtained by slicing. Simply redefine the three terms as follows:

$$accuracySlice = \frac{A}{A+B} \tag{1}$$

$$recallSlice = \frac{A}{A+C} \tag{2}$$

$$F1Slice = \frac{accuracy \times recall}{accuracy + recall} \tag{3}$$

Traditional approaches for slicing include dynamic slice, static slice and the combination of both of them, which cater of the most actual needs effectively and efficiently using all kinds of theories and techniques. However there is no uniform provision for the method of generating actual slice. And the actual total number of constraint facts that belong to a certain scenario, which correspond to A and C depicted in Figure 12, can't be obtained. Simply, we can easily get the total number

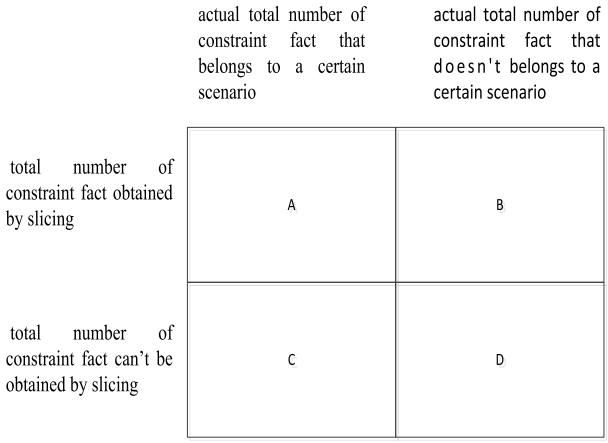

FIGURE 12. Four parameters of metrics.

of constraint facts obtained by slicing, which corresponds to the sum of A and B depicted in Figure 12 of (slicedFacts), and the total number of constraint facst can't be obtained by slicing, which corresponds to the sum of C and D depicted in Figure 12 (SumFacts- slicedFacts). In order to evaluate the effect of the approach, we introduce a metric naming sliceRate denoted by formula 4.

$$sliceRate = \left(1 - \frac{slicedFacts}{sumFacts}\right) \times 100\% \tag{4}$$

We replace the three open source softwares of SoundSea, Renren-generator and Commons-pool with S, R and C for simplicity. And consequently the nine scenarios can be denoted as S1-S3, R1-R3, C1-C3. In Figure 13 we depicted the siceRate of our approach related with the nine scenarios in the three programs.

In Figure 13 we can easily observe that the slice rate of scenario oriented program slicing is more than 96%, and there is no necessary relationship among the total slicing rate and the total number of lines, the total number of constraint facts and the slicing time.

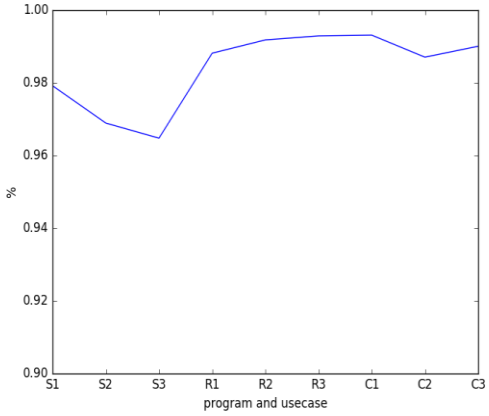

FIGURE 13. The slice rate of scenario oriented program slicing.

We further investigate the efficient of method of scenario oriented program slicing. Compared with the number of statements in total (LOC), the number of constraint fact (SumFact) plays a key role in complexity of software. Simply, we introduce a metric naming slicing time rate denoted by formula 5.

$$timeRate = \frac{sumTime + sliceTime}{sumFact} \times 100\% \tag{5}$$

And in Figure 14 we depict the trend of timeRate of our approach. In order to analyze the trend of the timeRate of slicing along with the total number of constrained facts (sumFact), we rank the sumFact of nine scenarios in the three open source softwares.

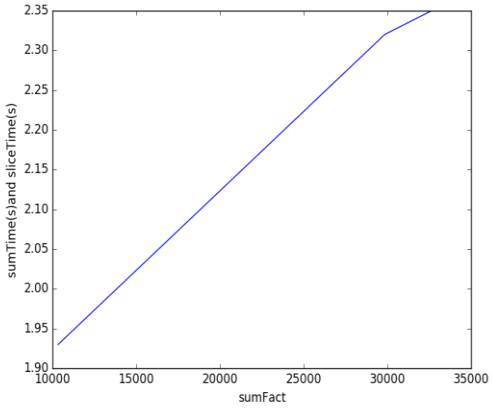

FIGURE 14. The time rate of scenario oriented program slicing.

Simply we can get that the growth of slicing time rate is linear other than exponent, and along with the growth of SumFact the acceleration is more and more smaller, which obviously indicates that our approach is efficient enough to some extent.

VI. CONCLUSION

Software applications will play more and more important role in our daily life. With the continuous expansion of software applications, software code is growing rapidly, and the scale and structure of software architecture are more and more complicated. The problems including the explosion of state space and the difficulties in balancing accuracy and scale have limited the credibility of large-scale industrial software.

Inspired by the preliminary work of the article [15] and our previous work in representation, storage and retrieve for large-scale software, in this paper we present a novel approach of scenario oriented program slicing based on the combination of constraint logic programming and program transformation, which is implemented as a tool. And our experimental evaluation suggests that our approach can realize scenario oriented program slicing efficiently and effectively.

There are also exists some limitations in the approach, however, which are supposed to be considered in future work. Rigorous mathematical foundation is very important to program analysis and manipulation. Without such a foundation, it is all too easy to assume that a particular transformation is valid, and come to rely upon it, only to discover that there are certain special cases where the transformation is not valid. In the future work, the results of scenario oriented program slicing is supposed to be formal validated based on rigorous mathematical foundation, and visual link between the result of slicing and source code are also can be created for error positioning, consequently the slice can be applied into the analyze the range of program variants.

REFERENCES

[1] E. Pira, V. Rafe, and A. Nikanjam, "Deadlock detection in complex software systems specified through graph transformation using Bayesian optimization algorithm," *J. Syst. Softw.*, vol. 131, pp. 181–200, Sep. 2017.

[2] M. Weiser, "Program slicing," in *Proc. 5th Int. Conf. Softw. Eng.* Piscataway, NJ, USA: IEEE Press, 1981, pp. 439–449.

[3] R. S. Kumar, M. G. Nath, and U. Raaghul, "An approach for UML based scenario oriented slicing," *Int. J. Recent Trends Eng. Technol.*, vol. 1, no. 2, pp. 141–143, 2009.

[4] I. Jacobson, I. Spence, and B. Kerr, "Use case 2.0: The guide to succeeding with use cases," *Commun. ACM*, vol. 59, no. 5, pp. 61–69, 2016.

[5] I. Jacobson, I. Spence, and B. Kerr, "Use-case 2.0," *Commun. ACM*, vol. 14, no. 1, pp. 61–69, 2016.

[6] J. Qian and B. Xu, "Scenario oriented program slicing," in *Proc. ACM Symp. Appl. Comput.*, 2008, pp. 748–752.

[7] C. Pietsch, M. Ohrndorf, U. Kelter, and T. Kehrer, "Incrementally slicing editable submodels," in *Proc. 32nd IEEE/ACM Int. Conf. Automated Softw. Eng.* Piscataway, NJ, USA: IEEE Press, 2017, pp. 913–918.

[8] S. Etalle and M. Gabrielli, "Transformations of CLP modules," *Theor. Comput. Sci.*, vol. 166, nos. 1–2, pp. 101–146, 1996.

[9] E. De Angelis, F. Fioravanti, A. Pettorossi, and M. Proietti, "Verification of imperative programs by constraint logic program transformation," in *Proc. EPTCS*, vol. 129, 2013, pp. 186–210.

[10] E. De Angelis, F. Fioravanti, A. Pettorossi, and M. Proietti, "A rule-based verification strategy for array manipulating programs," *Fundam. Inform.*, vol. 140, nos. 3–4, pp. 329–355, 2015.

[11] E. De Angelis, F. Fioravanti, A. Pettorossi, and M. Proietti, "Verifying relational program properties by transforming constrained horn clauses," in *Proc. CILC*, 2016, pp. 69–85.

[12] E. De Angelis, F. Fioravanti, A. Pettorossi, and M. Proietti, "VeriMAP: A tool for verifying programs through transformations," in *Proc. Int. Conf. Tools Algorithms Construct. Anal. Syst.* Berlin, Germany: Springer, 2014, pp. 568–574.

[13] E. De Angelis, F. Fioravanti, A. Pettorossi, and M. Proietti, "Semantics-based generation of verification conditions via program specialization," *Sci. Comput. Program.*, vol. 147, pp. 78–108, Nov. 2017.

[14] F. Fioravanti, A. Pettorossi, M. Proietti, and V. Senni, "Generalization strategies for the verification of infinite state systems," *Theory Pract. Logic Program.*, vol. 13, no. 2, pp. 175–199, 2013.

[15] S. Ren, M. Jia, F. Huang, and Y. Liu, "Visualization analysis framework for large-scale software based on software network," in *Proc. Int. Conf. Pioneering Comput. Sci., Eng. Educ.* Singapore: Springer, 2017, pp. 751–763.

[16] M. Weiser, "Programmers use slices when debugging," *Commun. ACM*, vol. 25, no. 7, pp. 446–452, 1982.

[17] B. Hofer and F. Wotawa, "Combining slicing and constraint solving for better debugging: The CONBAS approach," *Adv. Softw. Eng.*, vol. 2012, pp. 1–18, 2012.

[18] B. Hofer and F. Wotawa, "Reducing the size of dynamic slicing with constraint solving," in *Proc. 12th Int. Conf. IEEE Qual. Softw. (QSIC)*, Aug. 2012, pp. 41–48.

SHENGBING REN was born in Yueyang, Hunan, China. He received the B.Sc. degree in computer software from the Department of Mathematics, Huazhong Normal University, Hubei, China, in 1992, the master's degree in computer application technology from the Department of Computer Science, Central South University of Technology, Hunan, in 1995, and the Ph.D. degree in control theory and control engineering from the School of Information Science and Engineering, Central South University, Hunan, in 2007. He is currently an Associated Professor with the School of Software, Central South University. He has published over 50 papers. His research interests include software engineering, embedded system, image processing, pattern recognition, and dependable software. He is dedicated to the research concentrated mostly on dependable software and pattern recognition. He accomplished 10 research projects including two the National Natural Science Foundation of China as a Key Member. He is a Senior Member of the China Computer Federation.

MENGYU JIA is currently pursuing the master's degree with the School of Software, Central South University, China. She has published a paper and has applied for a patent. Her research interest is dependable software.