ISBN 978-981-11-7861-0
Proceedings of 2018 the 8th International Workshop on Computer Science and Engineering (WCSE 2018)
Bangkok, 28-30 June, 2018, pp. 647-651
doi: 10.18178/wcse.2018.06.108

Extracting Data Invariants from Promela

Wisut Suksribangteuy[+] and Wiwat Vatanawood

Department of Computer Engineering, Chulalongkorn University, Bangkok, Thailand

Abstract. Data Invariants are the essential conditions that hold during the execution of program. In software verification, the data invariants should be assessed to ensure the correctness of the system behavior. Unfortunately, it has been difficult to indicate the data invariants in a given source code. In this paper, we aim to extract relevant data invariants automatically from the Promela source code. The Promela code is typically used as the software modelling language in model checking tool, called SPIN model checker. We exploit the program slicing technique to minimize the Promela source code and track all of the relevant active bound variables to recognize the data invariants in term of their possible and valid bounded values. The data invariants are written in propositions and their temporal logic formula (LTL formula). The resulting data invariants are checked using model checking tool.

Keywords: data invariants, LTL, program slicing, dynamic slicing, promela, SPIN, formal verification

1. Introduction

Data invariants are constant data that could persist data or control program integrity. They are determined in specific range of the program and have to be true throughout the program execution.

LTL formulae is a modal temporal logic with modalities referring to time for specifying correctness requirement of program, it is possible to prove the correctness of the function for executable program

Many software programs emphasize to improve software quality by verify software program to help organizations. Verification is one of most effective way that provides a correctness of intended algorithms underlying a software with respect to a certain formal specification or property.

Sometimes, in the creating manual of LTL formulae is difficult. It is complicated problem of program. Promela language is difference from other languages that have multiple programming at the same time. Conceptually a program may be thought of as a collection of threads. Several threads may compute values of the same variable. Many of these threads may overlap others. Some people are not aware the correctness and completeness of the value to define program by LTL formulae.

This paper presents an automatically generate LTL from data invariants. LTL is used to determine the data invariants of Promela program. It would be used to determine conditions and verify program for formal verification of safety property. We present an approach to extract data invariants from promela code by using program slicing technique. It allows to use LTL formulae to verify an algorithms in Modeling language on the SPIN program.

2. Background

2.1. Invariants

[+] Corresponding author.
E-mail address: Wisut.S@student.chula.ac.th

Invariants conditions referring to the executing program that must always be true throughout the operation of the program and used by logical assertion. Data invariants are data in the form of the program referenced by the terms specified. It may be in the form of variables, equations or conditions that must be true throughout the operation of the program and allows the program to have safety properties. When program is executed, data invariants must always be stable.

2.2. Linear temporal logic

Linear Temporal Logic (LTL) is a mathematical language that used to present linear time logic and related to time and create in a formule related to paths that occur in the future. LTL is a small formulae, but very complex when is integrated. There are three types of symbols in the LTL include:

1. Atomic Proposition Symbols such as p, q, r, .., etc.

2. Temporal Connectives such as [] (Globally), <> (Eventually), O (Next), U (Until) presented in Fig. 1

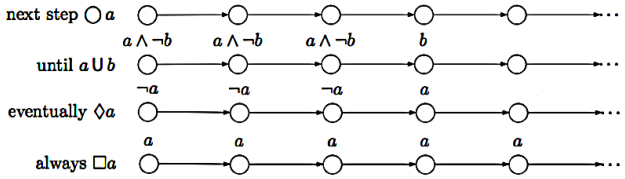

Fig. 1: Intuition for the main LTL operators [1].

3. Boolean Connectives such as ∧ (Conjunction), V (Disjunction), ¬ (Negation), -> (Unless)

2.3. Data extraction

Data extraction [2] is a process of extracting data that users require from large data. It obtains useful data in which is reliable and important to be further utilized such as data management or decision to do something. Data extraction is user's requirement selected data to apply in the future. It may be separation of program features to store in the table,

2.4. Program slicing

Program Slicing [3] is a technique to distinguish the components of a program, which may directly or indirectly affect the values of interest. Program Slicing is reduce the size of the program to a small size until you find the desired behavior. The classification of the program slicing has various techniques. However, we use the dynamic slicing in this research.

A dynamic slice [4] contains all statements that actually affect the value of a variable at a program point for an executable part of the program rather than all statements which behavior is identical. Dynamic Slicing [5] is computed on a given input and computed slice. The idea of dynamic slicing has been also extended for distributed programs. It is easier to identify those statements in the program in Fig. 2.

```
1   input (n,a);
2   max := a[1];
3   min  := a[1];
4   i := 2;
5   s:= 0;
6   while i ≤ n do
    begin
7       if max < a[i] then
        begin
8           max := a[i];
9           s := max;
        end;
10      if min > a[i] then
        begin
11          min := a[i];
12          s := min;
        end;
13      output (s);
14      i := i +2;
    end;
```

```
1¹   input (n,a);
2²   max := a[1];
3³   min  := a[1];
4⁴   i := 2;
5⁵   s := 0;
6⁶   i ≤ n
7⁷   max  <  a[i];
8⁸   max  := a[i];
9⁹   s := max;
10¹⁰ min > a[i];
13¹¹ output(s);
14¹² i := i +2;
6¹³  i ≤ n
```

An execution trace of the sample program on input n =3, a=(1,2,3).

Fig. 2: An example of dynamics slicing [6].

2.5. Promela and SPIN

Promela is a model language for asynchronous programs and communicate via shared variables or message channels. Promela model is constructed consist of processes, message channels and data objects. Promela [7] is a language for building verification models and supports the specification non-deterministic control structures. Promela models can be analyzed with the SPIN model checker, to verify that the modeled system produces the desired behavior. The example of promela code is shown in Fig. 3

```
1   int RedLight = 0;                    15          atomic {
2   int YellowLight = 0;                 16            RedLight = 0;
3   int GreenLight = 0;                  17            YellowLight = 0;
4   active proctype Trafic()             18            GreenLight = 1;
5   {                                    19          }
6     StateStart : goto StateRed;        20          goto StateYellow;
7     StateRed:                          21      StateYellow:
8           atomic {                     22          atomic {
9             RedLight = 1;              23            RedLight = 0;
10            YellowLight = 0;           24            YellowLight = 1;
11            GreenLight = 0;            25            GreenLight = 0;
12          }                            26          }
13          goto StateGreen;             27          goto StateRed;
14      StateGreen:                      28  }
```

Fig. 3: Overview of extraction data invariants.

SPIN is a tool for checking correctness of process interactions. The verification is random simulations of the program execution. There are various types of SPIN properties such as deadlock, violated assertions, unreachable code, system invariants or general LTL properties.

3. Approaches

Data extracted tool development. The first step is the process of gather promela code for extracting data into the extraction table. Data is separated by the specific defined data types. Second, dynamic slicing data by data and control dependence based on algorithms with criterions to get data invariants. The next step is translating data invariants to LTL formulae. The last step, we add LTL formulae on program to verify program on SPIN. All Steps can be shown in diagram below. In Fig. 4.

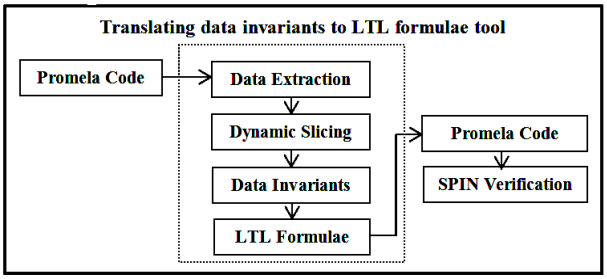

Fig. 4: Overview of translating data invariants to LTL formulae.

3.1. Extracting data from promela code

Extracting code program to data extraction. Data is separated to extraction table from code program. An example of program from Fic. 3, In Table 1. Data can be classified into various types such as line number, action, function, statement and reference data. If data not found or ignored data is skipped or put blank data.

Table 1: Extracting data of the program from Fig. 3

Line Number	Action	Function / Label	Statement	Reference Variable
1	Declare		RedLight = 0	RedLight
2	Declare		YellowLight = 0	YellowLight
3	Declare		GreenLight = 0	GreenLight
4	Proctype	Traffic		
6	Label	StateStart	Goto StateRed	
7	Label	StateRed		
9	Condition	StateRed	RedLight = 1	RedLight
10	Condition	StateRed	YellowLight = 0	YellowLight
11	Condition	StateRed	GreenLight = 0	GreenLight
13	Goto	StateRed	Goto StateGreen	

Line Number	Action	Function / Label	Statement	Reference Variable
14	Label	StateGreen		
16	Condition	StateGreen	RedLight = 0	RedLight
17	Condition	StateGreen	YellowLight = 0	YellowLight
18	Condition	StateGreen	GreenLight = 1	GreenLight
20	Goto	StateGreen	Goto StateYellow	
21	Label	StateYellow		
23	Condition	StateYellow	RedLight =0	RedLight
24	Condition	StateYellow	YellowLight = 1	YellowLight
25	Condition	StateYellow	GreenLight = 0	GreenLight
27	Goto	StateYellow	Goto StateRed	

3.2. Find variable from data and control dependence

Consider variables is interested or relevant variables from reference variable in the extraction table. Find variables by criterion is defined and respect data and control dependence in program such as intra-proctype, inter-proctype or init function. Because of several threads of variables scope in promela language. They may compute values of the same variable. Many of these threads may overlap other variables.

3.3. Identify slicing criterion and slicing of a program

A dynamic slice is an executable part of the program whose behavior is identical, for the same program input of the original program with respect to a variable of point at some execution position.[8] A slicing criterion of program P executed on program input x is C = (x,nq,V) where nq is action program, V is a set of program variables. A dynamic slicing of program P on slicing criterion C. If input is no value that set default as no input data. In Table 2 show overview of execution trace of the program.

Table 2: Execution trace of the program by 3 criterions

Slice(YellowLight)

Line Number	Action	Function / Label	Statement
2	Declare		YellowLight = 0
4	Proctype	Traffic	
6	Label	StateStart	Goto StateRed
10	Condition	StateRed	YellowLight = 0
13	Goto	StateRed	Goto StateGreen
14	Label	StateGreen	
17	Condition	StateGreen	YellowLight = 0
20	Goto	StateGreen	Goto StateYellow
21	Label	StateYellow	
24	Condition	StateYellow	YellowLight = 1
27	Goto	StateYellow	Goto StateRed

Slice(GreenLight)

Line Number	Action	Function / Label	Statement
3	Declare		GreenLight = 0
4	Proctype	Traffic	
6	Label	StateStart	Goto StateRed
9	Condition	StateRed	GreenLight = 0
13	Goto	StateRed	Goto StateGreen
14	Label	StateGreen	
18	Condition	StateGreen	GreenLight = 1
20	Goto	StateGreen	Goto StateYellow
21	Label	StateYellow	
25	Condition	StateYellow	GreenLight = 0
27	Goto	StateYellow	Goto StateRed

Slice(RedLight)

Line Number	Action	Function / Label	Statement
1	Declare		RedLight = 0
4	Proctype	Traffic	
6	Label	StateStart	Goto StateRed
9	Condition	StateRed	RedLight = 1
13	Goto	StateRed	Goto StateGreen
14	Label	StateGreen	
16	Condition	StateGreen	RedLight = 0
20	Goto	StateGreen	Goto StateYellow
21	Label	StateYellow	
23	Condition	StateYellow	RedLight = 0
27	Goto	StateYellow	Goto StateRed

3.4. Find relevant variable based on criterions

Find relevant variable from different function or label. Apply related variables from the previous step to arrange related groups for generate LTL formulae and combine relevant variable from criterions for get the result as Eq. 1, when F is function or label of program, Cn is Criterion number.

$$\text{DataInvResult}(Cn) = \Sigma\ (Fc1\ U\ Fc2\ U\ \ldots\ U\ Fcn) \tag{1}$$

DataInvResult(Cn) = (Redlight = 1 ∧ Yellowlight = 0 ∧ Greenlight = 0) v (Redlight = 0 ∧ Yellowlight = 0 ∧ Greenlight = 1) v (Redlight = 0 ∧ Yellowlight = 1 ∧ Greenlight = 0)

3.5. Generating LTL formulae and verify data invariants

Generate LTL formulae for safety property from program using extracting data invariants from criterions for verify the functionality of promela program to be true during execution of program. The scope of the value must be considered to prevent the data out of the. This will cause the program malfunction.

The algorithms considered and obtained the results in the previous step. Take the results from the combination different criterions to use in with the LTL formulae and use the symbol "[]" (Globally) keep in front of the LTL formulae by command LTL. To defined assertion safety property in the program. An example from Fig. 3. The result is as follows :

LTL ([](Redlight=1 ∧ Yellowlight=0 ∧ Greenlight=0) v (Redlight=0 ∧ Yellowlight=0 ∧ Greenlight=1) v (Redlight=0 ∧ Yellowlight=1 ∧ Greenlight=0)))

The last step is put LTL formulae in promela code. It will be executed for ensure the data invariants has been preserved. Data invariants must be exactly the same as the result from the original program.

4. Literature References

Promela language supports the concurrent threads. It has assertion or LTL formulae to verify variable or condition in program during execution. Sometimes, we are not sure the value of variable used to generate LTL formulae that is desired range or correctness, so this may cause the program to malfunction.

A lot of research has recently been done to reduce the size of static and dynamic slices. Many problems of the program slicing are inefficient slices. [9] Finding all statement in a program that directly or indirectly affect the value of a variable occurrence is referred to as program slicing that may uncover statement.

Several algorithms have been proposed for dynamic slice computation. [8] The existing methods of dynamic slice computation are based on backward analysis after the executing trace of the program is first recorded cause more detailed description of the algorithm. This problem may be partially alleviated by using some methods that reduce the amount of recorded information.

In general, execution trace based algorithms have to store the execution trace in the extraction table and their space complexity depends on the size of a particular execution. The space and time complexity of these algorithms is a function of the size of a program and the number of variables.

5. Summary

We have shown that our techniques may have to generate LTL formulae. It can also be useful for verify software program from scope of variable to be true during executing the program. Dynamic program slicing is techniques significantly reduce the size of the program.

In this paper we have could extract data invariants from promela code. We presented dynamic slicing methods and discussed the algorithms of computation of dynamic program slicing.

For future work, we plan to created LTL formulae covers all conditions inside the scope that provide research challenges for those working on and around program slicing and Our approach applies to several software program in organization.

6. References

[1] Christel Baier and Joost-Pieter Katoen, Principles of Model Checking, MIT Press, 2008,

[2] N. Kamanwar and S. Kale, Web data extraction techniques : A review, Conference on Futuristic Trends in Research and Innovation for Social Welfare, 2016,

[3] Keith Gallagher and David Binkley, Program Slicing, Frontiers of Software Maintance, China, 2008.

[4] S.S. Barpanda and D.P. Mohapatra, Dynamic Slicing of distributed object-oriented programs, department of Computer Science and Technology, India, 2010.

[5] Franz Wotawa, On the use of constraints in dynamic slicing for program debugging, Fourth International Conference on Software Testing, Verification and Validation, 2011.

[6] Bogdan Korel and Jurgen Rilling, Dynamic program slicing method, Information and Software Technology Volume 40 issues11-12, 1998.

[7] Rene Neumann, Using Promela in a Fully Verified Executable LTL Model Checker, International Conference VSTTE 2014, Austria, 2014.

[8] Satish Yalamanchili and Bogdan Korel, Forward computation of dynamic program slices, ACM SIGSOFT international symposium on Software testing and analysis, Seattle USA, 1994.

[9] Hiralal Agrawal and Joseph R. Horgan, Dynamic Program Slicing, Conference on Programming Language Design and Implementation, New York, June 20-22, 1990.