C-Language Verification Tool Using Formal Methods “VARVEL”

TOKUOKA Hiroki, MIYAZAKI Yoshiaki, HASHIMOTO Yuusuke

Abstract

The newly developed C-language verification tool VARVEL utilizes the model checking technology that is one of the formal methods and detects bugs (runtime errors) that are inherent in programs by modeling the source codes written in C-language and verifying them statically and exhaustively. This paper is intended to introduce the technical background of VARVEL and outline its functions.

Keywords

static verification, C-language, formal method, model checking

1. Introduction

The significant increase in the scale of software development in the embedded domain has meant that the assurance of software quality has become an important issue. Among a variety of development methods and tools proposed for dealing with this issue, the methods known as the formal methods are attracting particular attention. At NEC, we applied the model checking technology, which is one of the formal methods, and developed VARVEL (V 1.0) as a tool for detecting bugs (runtime errors) that are inherent in programs written in C-language.

2. Formal Methods

The formal methods refer generically to techniques based on logic, set theory and algebra for the specification and verification of systems. As a kind of formal method, model checking is “the technology for thoroughly checking abnormal behavior of a system that may either be software or hardware by inputting its state transition model in a computer,”[1] and regarded as the good method capable of early discovery of troubles during execution. It has initially been used in design verification in the fields with high reliability and safety requirements, such as aerospace, communications and semiconductor industries, but its application into the verification of source codes has been advanced recently.

NEC laboratories have also been conducting research into application of a model checking engine that has been proven in LSI design verification for the verification of source codes[2] and VARVEL is one of the achievements of this research (**Fig. 1**). VARVEL analyzes the source codes in C-language and extracts control flow that consists exclusively of the parts associated with runtime errors. It then develops a gigantic logical formula that represents the transition of variable values in the control flow and a logical formula that represents each runtime error and calculates a variable value that satisfies both of the two formulae by using a model checking engine based on a highly efficient algorithm for the satisfiability problem (SAT). The strength of VARVEL lies in the construction of a small-sized model to compensate for the weakness against state explosion of model checking and the use of a model-checking engine of the world’s fastest class. The model-checking engine performs an exhaustive search of 4.2 billion (2^{32}) value combinations per variable so that it can detect troubles that would be overlooked with traditional review or test methods.

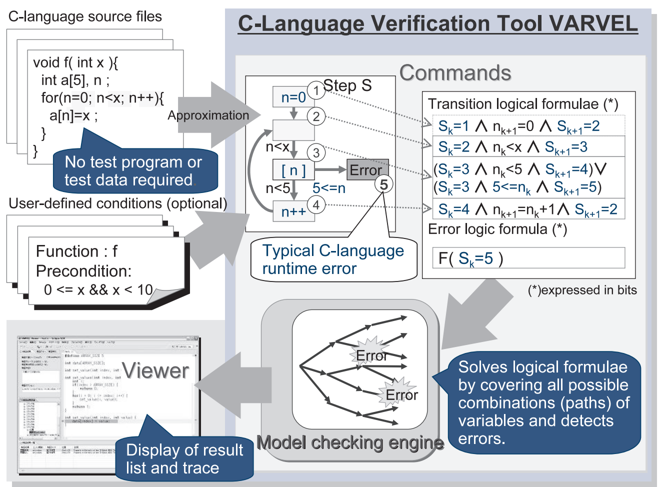

C-language source files
void f(int x){
int a[5], n ;
for(n=0; n<x; n++){
a[n]=x ;
}
}
No test program or test data required
Approximation
User-defined conditions (optional)
Function : f
Precondition:
0 <= x && x < 10
C-Language Verification Tool VARVEL
Commands
Step S
n=0
n<x
[n]
Error
n<5
5<=n
n++
1
2
3
4
5
Typical C-language runtime error
Transition logical formulae (*)
$S_k=1 \wedge n_{k+1}=0 \wedge S_{k+1}=2$
$S_k=2 \wedge n_k<x \wedge S_{k+1}=3$
$(S_k=3 \wedge n_k<5 \wedge S_{k+1}=4) \vee (S_k=3 \wedge 5<=n_k \wedge S_{k+1}=5)$
$S_k=4 \wedge n_{k+1}=n_k+1 \wedge S_{k+1}=2$
Error logic formula (*)
$F(S_k=5)$
(*)expressed in bits
Error
Error
Model checking engine
Solves logical formulae by covering all possible combinations (paths) of variables and detects errors.
Viewer
Display of result list and trace

Fig. 1 C-language verification tool VARVEL.

3. Target of Development

Runtime errors that are generated only under specific conditions during program execution, such as NULL pointer dereference or array bounds violation, have usually been detected by test processes that consist of checking by executing the program for several cases. VARVEL makes it possible to detect runtime errors by means of static verification (verification without program execution) of the source codes at earlier stage, instead of by the traditional detection method using the test process as described above. Since it is generally said that bugs are hard to find as the process advances and the man-hours required for returning to the previous process increases when they are detected in a later process, the early discovery possibility of VARVEL is expected to improve the efficiency of quality assurance.

C-language, the target of VARVEL, is used widely in the embedded software domain because it employs simple language specifications and enables tuning of performance to a deep level. On the other hand, it also presents a problem, which is the presence of language elements that tend to cause runtime errors such as handling pointers and character strings. Since there are a huge number of combinations that can be taken by the paths and variables during execution, it is hard for the manual testing alone to detect all of the possible runtime errors in a practical period and there are not a few cases in which the problems in the program embedded in a product become clear after it has been shipped. VARVEL has made it possible to detect bugs that have been hard to find with the traditional method, by means of the automatic checking of easily overlooked run time errors by exhaustively modeling the possible combinations of paths and variable values.

	CDI Tool	VARVEL
Verification target	Detection of simple bugs, and descriptions and coding convention violations that are hard to maintain and which lead easily to troubles.	Detection of runtime errors that are hard to find by a code review or by using limited test patterns.
Verification range	Area closed inside a function.	Area across functions (including the case in which they are distributed in multiple files).
Verification method	Syntax check and partial modeling that is independent of variable values.	All-inclusive modeling including variable values.
Verification time	Relatively short.	Relatively long.
Main user process	Mainly used as a tool for code review after packaging in order to improve maintainability and reliability.	Mainly used before the execution (dynamic) testing connecting several modules in order to improve the reliability.

* CDI Tool: Code Inspection Tool.

Table 1 Comparison of CDI tool and VARVEL.

4. Features and Positioning of Product

Currently, the code inspection tools (CDI tools) are used widely as the static source code verification tools for detecting descriptions that would cause troubles and the violations of coding conventions. **Table 1** shows the differences between the CDI tools and VARVEL.

As shown in the table, the two tools have different applications and purposes, and can be regarded as being in complementary relationship. VARVEL is not a tool to replace existing tools. Instead, it is assumed to be combined with an existing tool as a solution for detecting bugs effectively.

5. Outline of VARVEL

Table 2 shows the main specifications of VARVEL (V 1.0).

(1) Functional Configuration and Operating Environments

VARVEL is composed of the Command function that inputs the analysis target C-source code and outputs the verification result (XML format), and the Viewer function that displays the verification result output from the Command and supports the interactive confirmation, analysis and survey. The operating environments of these functions are Linux and Windows respectively.

(2) Verification Types

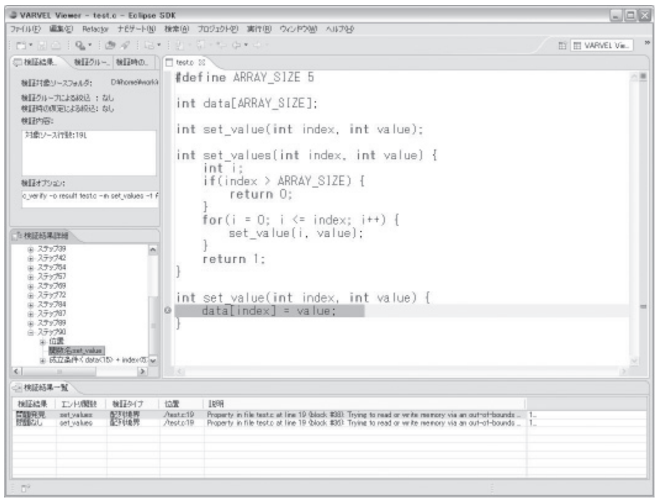
●Tool specifications

	Description
Input specifications	C-language source file(s). ISO/IEC 9899:1990(E). * The compilation environment including the header file should be available.
Operating environments	(Command) Linux (Operation checked with FedoraCore 4) (Viewer) Windows XP sp2, Eclipse 3.2 or after.

●Verifiable runtime error category

Verification Type	Detected Runtime Errors
Invalid pointer dereference	Detection of errors in dereferencing memory area using a pointer. Dereferencing address 0, released area and illegal area.
Array bounds violation	Detection of errors in array access. Upper and lower limit access violations of array bounds. int arr(10): arr(x) = 100: //Alarm if suffix x is out of range 0-9. int *lp = (int *) malloc(size of (int)): ip++: *ip = 100; //Alarm
String operation error	Detection of inconsistency between string length and array size. Buffer overflow, buffer underflow, strong function usage mistake. char arr(10): for (i=0; i<x: i++) strcat(arr;"bc"): //Alarm when x is 6 or more.
Memory management error	Detection of inconsistency between malloc/calloc and free call count. Memory leak and releasing of non-reserved area (including double release).

Table 2 Main specifications of VARVEL.


VARVEL Viewer - test.c - Eclipse SDK
ファイル(F) 編集(E) Refactor ナビゲート(N) 検索(A) プロジェクト(P) 実行(R) ウィンドウ(W) ヘルプ(H)
test.c

```
#define ARRAY_SIZE 5

int data[ARRAY_SIZE];

int set_value(int index, int value);

int set_values(int index, int value) {
    int i;
    if(index > ARRAY_SIZE) {
        return 0;
    }
    for(i = 0; i <= index; i++) {
        set_value(i, value);
    }
    return 1;
}

int set_value(int index, int value) {
    data[index] = value;
}
```

検証結果	エントリ関数	検証タイプ	位置	説明
問題発見	set_values	配列境界	/test.c:19	Property in file test.c at line 19 (block #38): Trying to read or write memory via an out-of-bounds ...
問題なし	set_values	配列境界	/test.c:19	Property in file test.c at line 19 (block #36): Trying to read or write memory via an out-of-bounds ...

Fig. 2 Viewer.

VARVEL detects four types of runtime errors (called the verification types). The type can be specified as an option at the time of command execution (Table 2).

(3) Viewer

The Viewer is provided as an Eclipse plug-in. It makes it possible to display the position where an error is detected and the flow of the process until the error generation (line tracing and the variable values at each line) in a merged display with source codes in order to enable interactive confirmation (**Fig. 2**).

(4) User-Defined Conditions

VARVEL performs exhaustive verification by modeling all of the possible values because, if the target source code includes an argument of a function that is the basis of analysis or there is an external function call without a source code, the information from the given source code is not enough for analyzing the argument and return values. In this case, however, the model becomes larger than necessary, the analysis time is prolonged, and erroneous detections (detections of non-bug states as bugs) would increase. To deal with this, if it is known from the specifications that the argument and/or return value of the function take only specified ranges of values, the user is allowed to define these conditions before executing the tool as the presupposition for the analysis (these conditions are referred to as the user-defined conditions). VARVEL inputs the user-defined conditions specified at the same time as the analysis-target source code and builds the appropriate model.

6. Conclusion

VARVEL as introduced in the above presupposes utilization in the code verification service targeted inside the NEC Group and has already started to be applied in the actual development fields. The code verification service is performed by specialist personnel and consists of checking the quality of developed software assets and reporting on the check results. The service has already been used tentatively in several projects and has achieved results verifying the effectiveness of the tool, by detecting troubles that would not have been detected by ordinary testing.

In the future, we will improve the tool by perfecting its verification accuracy and by reducing the analysis time and will eventually incorporate it in the SystemDirector Developer’s Studio Embedded, which is an integrated development tool for embedded software. In addition to the above improvements, we will also enhance the functions for supporting software quality improvements, for example by adding the Design by Contract (verification of correct packaging by defining the developer’s design intentions such as pre-/post-conditions and assertions.

* Windows is a registered trademark of Microsoft Corporation in the USA and other countries.
* Linux is a registered trademark or trademark of Linus Torvalds in the USA and other countries.
* Fedora is a trademark of Red Hat, Inc.
* Design by Contract is a registered trademark or trademark of Interactive Software Engineering.

References

1) NIKKEI ELECTRONICS, 2005/12/19, pp. 87-121.
2) F. Ivancic et al:Model checking C programs using F-Soft. Invited paper in the Proceedings of the IEEE International Conference on Computer Design (ICCD), October 2005.

Authors' Profiles

TOKUOKA Hiroki
Assistant Manager,
Software Engineering Division,
NEC Corporation

MIYAZAKI Yoshiaki
Manager,
Software Engineering Division,
NEC Corporation

HASHIMOTO Yuusuke
Manager,
Software Engineering Division,
NEC Corporation