Proceedings of the 5th WSEAS International Conference on Telecommunications and Informatics, Istanbul, Turkey, May 27-29, 2006 (pp105-109)

A Purpose-Directed Reachability Analysis Approach

ABDELAZIZ GUERROUAT, HARALD RICHTER
Department of Computer Science
Clausthal University of Technology
Julius-Albert-Str. 4
D-38678 Clausthal-Zellerfeld
GERMANY

Abstract: - The paper presents a new approach for reachability analysis. This is based on analysis purposes that are established by an analyser expert. The analysis purposes address system specification parts that may be 'suspected' of presenting non-reachable states and thus the analysis is especially focused on. Such information about 'suspicious' specification parts to be checked is based on special knowledge about the system code and behaviour. The approach is characterized by its usefulness comparatively to other complex and theoretical approaches, for instance, to those based on petri nets. We describe the *analysis purposes* and explain how reachability analysis can be performed using this concept.

Keyword: - Formal methods, reachability analysis, state/transition-based systems, system verification and validation

1 Introduction

Reachability analysis is very important in verification and synthesis of various systems such as control systems, communicating systems and communication protocols, computer programming (concerning compilers) etc. In computer programming, for instance, unreachable code (*dead code*), consists of one or more statements or entire routines that will never be accessed [1]. These will never be executed regardless of the values of variables and other conditions at run time (s. Fig.1).

```
…
while (condition) {
  do branch_1
  continue;
  do branch_2
}
…
```

unreachable code

Fig.1: Example of an unreachable code

Most languages like C++ allow unreachable code. According to [1] unreachable code may result from many sources including a common programmer practice to temporarily disable code. However, these temporary intended changes often take their way to final and release versions. Other causes of unreachable code consist of redundant checking of exception conditions, and debugging code that in fact should be removed. Thus, whole routines can be unreachable code if they are defined but are no longer called, or are called only from unreachable code.

Unreachable code may be the source of logical errors due to changes in the assumptions and environment of the program. This implies unnecessary work for the compiler and may result in *code bloat* [1].

Reachability analysis has received a lot of attention since decades in almost computing areas. Many reachability methods and approaches have been proposed to deal more efficiently with this problem [2] [3] [4] [5]. Due to the non-decidability character of this problem, these approaches are still applicable only under some assumptions on the given system. In addition, they usually deal with a specific description model.

Reachability analysis has proved to be one of the most effective methods in verifying correctness of communication protocols and distributed systems based on the state transition model [6] [7]. Consequently, many protocol verification tools have been built based on the method of reachability analysis. Nevertheless, it is also well known that state space explosion is the most severe limitation to the applicability of this method. Note that, for state/transition-based models reachability refers to the problem of computing bounds on the set of states that can be reached by the given system.

Our present paper is to serve one main purpose: give a new general way to deal with the reachability analysis problem by targeting given system parts that may be 'vulnerable' for reachability. We are inspired from various methodologies and frameworks developed for protocols and open communicating systems [8]. Because the experience has proven the

practical limit of many theoretical approaches that check at once the system. This is due to many economical and feasibility factors: very high cost, time-consume, lot of computer resources, etc. In addition, such theoretical approaches, refer usually to theoretical concepts that require appropriate knowledge to be understandable and traceable. However, the practitioners do not necessarily dispose of such knowledge.

The paper is organised as follows. In Section 2 we explain what we mean with 'analysis purposes'. Section 3 presents the principle of the analysis approach. An example that is based EFSMs (extended finite state machines) is given in Section 4. Finally, Section 5 concludes the paper and gives an outlook of futures works.

2 Analysis Concepts

The main concepts that relate to the analysis approach are: specification describing a *set of system requirements*, a *set of analysis purposes* and a *set of analysis cases*. The system requirements are included in the specification and indicate the expected behaviour of the *SUA* (system under analysis). Each analysis case is related to a precise analysis purpose.

System requirements are requirements on the behaviour of conforming implementations (*dynamic requirements)*. These can be represented as a set of requirements $R_S=\{r_1, r_2, \ldots, r_n\}$. Thus, $r_i \in R_S$ represents a conformance requirement to which an analysis purpose relates. A *SUA U* is conform (reachable) to the specification *S* if *U* satisfies all conformance requirements in R_S, i.e. $\forall r_i \in R_S$: (U *sat* r_i). In this approach the conformance is defined as a relationship because the specified analysis verdicts are based principally on the satisfaction by *U* of the conformance requirements given in the analysis purposes.

An analysis purpose is defined as a (un)formalised description of a closely defined analysis ground which relates to a single conformance requirement. That is, the set of analysis purposes *AP* usually represents as subset of the set of conformance requirements R_S: $AP \subseteq R_S$.

An *analysis case* relates to a given analysis method and is consisting of the following:

- An analysis purpose
- A preamble
- An analysis body
- A postamble
- Analysis verdicts

The *preamble* is a trace (event sequence) that brings U from the initial sate to the requirement aimed by the analysis purpose. An *analysis body* represents the requirement to be analysed and addressed by the analysis purpose. A *postamble* is a trace that brings back *U* from the actual (un)reached state after performing the analysis on the *analysis body* to the initial state. *Analysis verdicts* are statements of 'reached' or 'unreached' that have to be specified for the analysis events of the *analysis case* and that assess the conformance of *U* regarding the analysis purpose. For an analysis event, the analysis verdict is associated '*pass*' if it is conform to the analysis purpose and '*fail*' if it doesn't.

3 The Approach

First, we explain how one does deduce the analysis components introduced in the previous section.

3.1 Analysis case

An *analysis case AC* is a 5-Tupel $AC=<A, P, B, M, V>$ that is derived from the expected (reachable) behaviour *S* and the set of analysis purposes *AP*.

3.2 Analysis purposes

As indicated above, an analysis purpose *A* designates a single requirement r_i of the system under analysis *SUA U*. The specification of the set of analysis purposes *AP* depends on the nature of the used model for the specification of *U* and *S*. *U* is the system to be analysed that may presents unreachable parts and *S* is the specification providing the initially expected system behaviour, i.e. whose all specification parts are reachable. It is supposed that *U* and *S* are using the same specification model, e.g. state/transition based model or C++-Code etc.

3.2.1 Example

"*after the sequence of transitions* σ *U must reach state s*" etc. It is supposed again that the same state/transition model is used.

3.3 Preamble

A preamble *B* is a trace (an event sequence) that brings *U* from its initial state to a given state or part. It is supposed that the complete traversed trace staring at the initial is executable and the intermediate parts or states until the edge are reachable.

3.3.1 Example

Assuming a state/transition model for *U* and *S*. We denote $s \xrightarrow{*} s'$ as sequence of transitions starting at the state *s* an ending at state *s'*. A transition t_i is a 5-tuple of the form $t_i = <s_{1i}, i_i, p_i, o_i, s_{2i}>$ where s_{1i} is the starting state (the edge) and s_{2i} the next state (the tail) of the transition, i_i the input event, p_i the enabling condition (predicate) of the transition and o_i the output event after performing the transition. Note that all these components may depend on a context c_i, i.e. a subset of variables and parameters. Thus, the state *s'* of the preamble given above designates the state from which the analysis starts, i.e. the conformance requirement targeted by the analysis purpose. In this case all enabling predicates p_i of the state transitions that trace the preamble must 'fire' (be fulfilled).

3.4 Analysis body

An analysis body *B* consists of a single conformance requirement that is addressed by an analysis purpose. This presents a 'suspicious' part (behaviour) of U for unreachability. Thus, the analysis body should start at the edge part reached by the preamble. While the behaviour of *U* regarding reachability is assumed to be correct for preamble, this is still open for the analysis body *B*. Thus, U may or may not reach the part aimed by the analysis purpose.

3.4.1 Example

Assuming again a state/transition-based *U* and *S*, the analysis body may consist of a single transition whose 'firing' must be proven. This means, regarding the analysis purpose, one must state whether a given next state can be reached starting from the state to which the preamble leads. This can be performed by proving the satisfaction of the enabling predicate of the transition with the same context as for the preamble. Note the edge of this transition represents the state reached by the preamble.

3.5 Postamble

A postamble *M* is trace that brings *U* from the reached edge after performing the analysis on the analysis body to the initial state. This assumes that the initial state is always reachable to perform following analysis cases on *U*, .i.e. all enabling predicates of the trace bringing back *U* to its initial state fire. In both cases reachability or unreachabilty this should be possible.

3.5.1 Example

Given a reached[1] state s after performing the analysis body for *U*. We assume here also a state/transition model for both *U* and *S*. It always possible to bring *U* back to its initial state s0 through a trace consisting of one or more transitions whose edge is s and tail s_0. It is assumed that the enabling predicates of all these transitions fire.

3.6 Analysis Verdicts

The analysis verdicts *V* are assignments for the analysis case depending on the analysis purpose. For analysed event of the analysis body, it is indicated whether *U* reaches the required part or state (specified with 'reached') or not (specified with 'unreached'. Depending on these specifications, an analysis verdict will be assigned to the given analysis case: 'pass' if performing the analysis case is successful, else 'fail'. The verdicts will be used in the statement of the reachability or not for the given system under analysis *U*.

4 Example

We assume in this example a state/transition-based model EFSM (extended finite state machines) for U and S. The problem of reachability / unreachability analysis consists of the detection of non-executable parts. Thus, deciding whether a given EFSM modelling a function for a given system component contains non-executable transitions and detecting them if they exist.

In addition to the statement of the reachability for a given *U*, the detection of non-executable branches allows also to deduce a specification whose all transitions are executable in addition. This is obtained by eliminating all non-executable transitions and their descendants. We can find all non-executable branches in a EFSM as follows. A branch $s \rightarrow s'$ in the EFSM is a non-executable and thus, the system part with edge *s'* is unreachable if the two following conditions are fulfilled:

- $\exists\, x_1, \ldots, x_k\, [\delta_s(x_1,\ldots,x_k)]$
- $\neg(\exists\, x_1, \ldots, x_k\, [\delta_{s'}\, (x_1,\ldots,x_k)])$

where

- $\delta_s(x_1,\ldots,x_k)$ represents the conjunction of all enabling predicates for the context $x_1, \ldots, x_k$ from the initial state s_0 until the state *s*.

[1] This may be the same state in the case of non-reachable state, i.e. tail is the same edge since no transition is possible.

These represent the enabling predicates of the preamble.

- ¬() denotes the negation of the expression between the parentheses.

In other words, the branch $s \rightarrow s'$ given above represents the analysis body aimed by the analysis purpose.

The problem for deciding, whether a given branch of a system under analysis EFSM is non-executable (the tail is unreachable) is resolvable under certain limiting assumptions.

If the domains of state variables are finite and countable it is always possible to solve the problem analytically or by simulation. In fact, for each state of the EFSM it is possible to assess whether there is a context (variables assignment) for which the predicates are not satisfied.

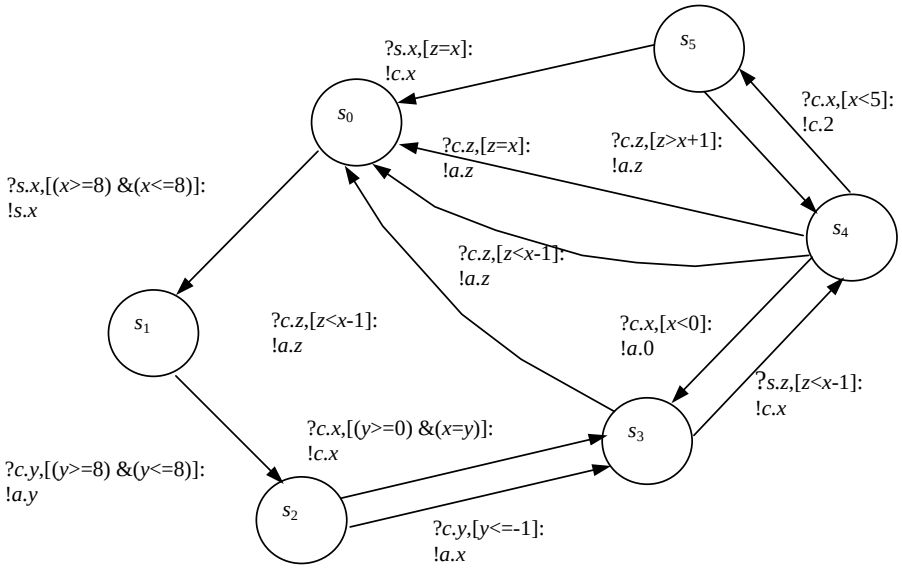

Fig.2: Example

For example, for the state s_3 and s_4 in Fig.2, we can show that

- $\exists\, x, y, z\, [\psi_{s3}(x, y, z)]$
  $= x, y, z\, [((x>=8)$ *or* $(x<=8))$ *and* $((y>=8)$ *or* $(y<=8))$ *and* $(y<=1)$ *and* $(z<x-1)]$
  $=$*true*
- $\exists\, x, y, z\, [\psi_{s4}(x,y,z)]$
  $= x, y, z\, [((x>=8)$ or $(x<=8))$ *and* $((y>=8)$ *or* $(y<=8))$ *and* $(y<=1)$ *and* $(z<x-1)$ *and* $(x=z)]$
  $=$*false*

In this simple example, we suppose that the variables domains are of type integer, finite and countable. We can easily verify that s_3 is reachable from the initial state, but the branch $s_3 \rightarrow s_4$ is non-executable and the thus the system part with edge s_4 is unreachable.

For convenience, an event of the EFSM (s. Fig.2) is represented as $\$g.e$. The symbol \$ denotes either the input symbol '?' or the output symbol '!'. The letter *g* represents the gate name, that is, the name of the component sending the event or receiving it. For example, the letters *s*, *c* and *a* designate sensors, the controller and actuators, respectively.

5 Conclusion

In this paper we presented a new approach for reachability analysis that in contrast common approaches is purpose-directed. In addition to its general aim, it needs less analysis efforts and resource requirements and thus lower costs. Indeed, the analysis is focused only on system parts assumed to be 'vulnerable' for reachability.

The principle of 'purpose-directed' adopted in this approach can be also extended and used to check other system properties like deadlock, livelock, safety, partial validation and correctness etc.

In future works, we are planning to deal in more details with the principle presented here, especially the formalisation of a relationship between the analysis purposes and the system parts addressed by it depending on the used model. In addition, we intend to apply the principle for concrete cases, in particular for compilers as mentioned in Section 1.