A Combined Approach for Reachability Analysis

Abdelaziz Guerrouat
Clausthal University of Technology
Department of Computer Science
Julius-Albert-Str. 4
D-38678 Clausthal
aguerrou@in.tu-clausthal.de

Harald Richter
Clausthal University of Technology
Department of Computer Science
Julius-Albert-Str. 4
D-38678 Clausthal
richter@in.tu-clausthal.de

***Abstract*—This paper presents the principle of an approach that allows to analyse most important system properties like reachability, executability, deadlock-freeness etc. The approach is based on two concepts: analysis purpose-directed analysis and specification unfolding. The first concept drives the analysis to the aimed specification part whereas the second allows to present the specification behaviour in a suitable form for the analysis. In contrast to the common analysis methods, the present approach considerably alleviates the state-explosion problem.**

Keywords—Formal methods; reachability analysis; formal specifications; testing and validation

I. INTRODUCTION

The analysis of specification properties like reachability, executability, deadlock-freeness, determinism, completeness is very a important task in verification and synthesis of systems such as control systems, communicating systems and communication protocols, computer programming (concerning compilers) etc. It allows to improve the correctness and the performance of a system. Indeed, 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. The presence of unreachable code may be the source of logical errors due to alterations in the program or significant changes in the assumptions and environment of the program. This causes unnecessary work for the compiler and may result in *code bloat*. In addition, unreachable code has an important impact on the system performance and specification readability.

Many reachability methods have been proposed to deal with this problem [2] [3] [4] [5] [13] [14]. Nevertheless, the state space explosion problem particularly due to interleaving is generally still unsolved. This fact makes the reachability analysis a non-decidable problem and considerably limits the use of such methods.

In our approach the state explosion problem is well alleviated because the analysis concerns only those specification parts that are aimed by analysis purposes and not the whole specification. The targeted system parts are considered to be the most vulnerable ones for reachability. This implies lower costs, less time-consumes, lower amount of computer resources, etc.

The paper is organised as follows. In Section 2 we explain the principle of the purpose-directed analysis. Section 3 presents the specification unfolding principle and shows how this is integrated into the purpose-directed analysis. Finally, Section 4 concludes the paper and gives an outlook of future works.

II. THE PURPOSE-DIRECTED REACHABILITY 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 is related. A *SUA U* is conform 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 principally based 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 a 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*
- A preamble *P*
- An analysis body *B*
- A postamble *M*
- Analysis verdicts *V*

The *preamble* is a trace (event sequence) that brings U from the initial sate to the requirement aimed by the analysis

Proceedings of the International Conference
on Software Engineering Advances (ICSEA'06)
0-7695-2703-5/06 $20.00 © 2006 IEEE

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.

Now, we explain how the analysis components introduced above can be deduced.

A. 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 where A is the *analysis purpose* associated to the analysis case AC, P the *preamble*, B the *analysis body*, M the *postamble* and V the *analysis verdicts*.

B. 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 specification of U and S. U is the system under analysis that may include unreachable parts, and S is the specification providing the expected system behaviour, i.e. all specification parts of S are reachable. We assume that U and S are using the same specification model, e.g. state/transition-based model or C++-Code etc.

1) Example

"*after the sequence of transitions* σ *U must reach state s*" etc. In this example, we assume that we are using a state/transition model.

C. 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 assumed that the complete traversed trace staring at the initial state is executable and thus the intermediate parts or states until the edge are reachable.

1) Example

We consider the same specification model for both U and S that is state/transition-based. 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).

D. 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.

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.

E. Postamble

A postamble M (similar to reset) is a trace that brings U from the reached edge after performing 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 that brings back U to its initial state 'fire'. In both cases, i.e. reachability or unreachabilty, it is assumed that bringing back U to its initial sate is always possible.

1) Example

Let s be a reached[1] state after performing an analysis body B for U. We consider again a state/transition model for specifying U and S. It always possible to bring back U to its initial state s_0 by means of a trace consisting of one or more transitions beginning at s and ending at s_0. It is supposed that the enabling predicates of all these transitions are satisfied ('fire').

F. Analysis Verdicts

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

III. Specification Unfolding

The main question that may arise from the purpose-directed reachability analysis presented above is how is it possible to present the specification in an appropriate form for which this principle become applicable. In other words, how one can express the system behaviour in terms of traces. Because the

[1] This may be the same state in the case of non-reachable state, since no transition was possible.

above introduced concepts of analysis purposes are given in term of traces, i.e. event sequences. This is done by unfolding the specification behaviour. The process of the reachability analysis based on analyse purposes and behaviour unfolding is depicted by Fig. 1.

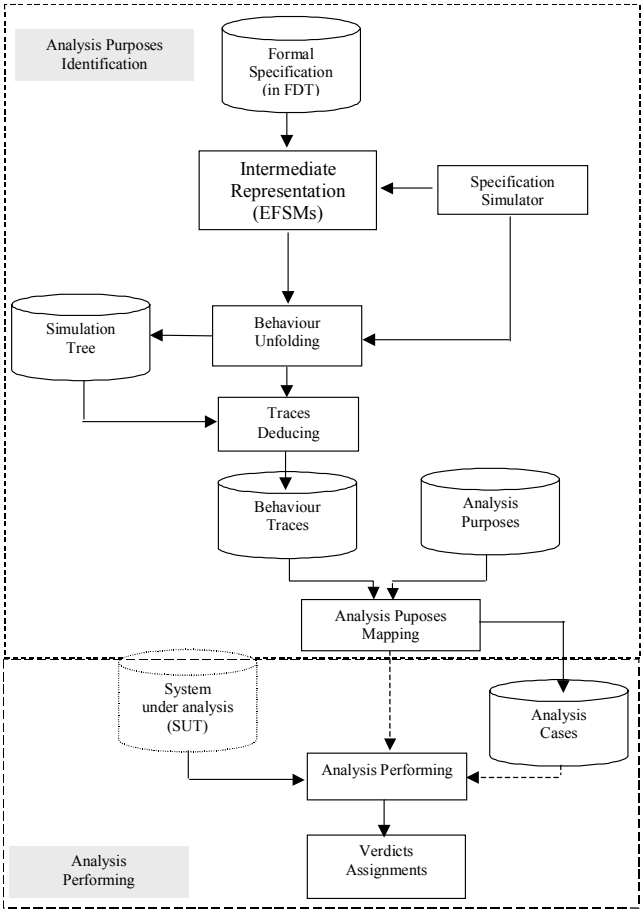

Figure 1. The analysis process based on analysis purposes and unfolding

The main analysis process is composed of two parts: "Analysis purposes identification" and "Analysis performing". The fist one allows to identify the set of analysis purposes based on the unfolded behaviour of the specification. The system specification is based on *formal description techniques* (FDTs) like Estelle, SDL or LOTOS [6] [7] [8] [9] [10].

Fig. 2 shows an example of a portion of a LOTOS specification. This portion consists of one process, called *prozess*.

```
...
process prozess[g1,g2,g3,g4](y:nat):noexit:=
              g1?x:nat[(x ge 0) and (x le 3)];g2!y[y ge 1];
             (g1!x[x gt 3];(g1!x[x le 2];stop []
              g1!x[(x gt 2) and (y lt 1)];stop) []
             (i;i;g1!x[x le 1];stop) []
             (g1!x[x ge 1];g4!y[(y ge x) and (x le 3)];stop))
endproc (*prozess*)
...
```

Figure 2. A part (process) of a LOTOS specification

The formal semantics of FDTs are supported by EFSMs (Extended Finite State Machines). Many simulation tools have been already developed to unfold a specification behaviour [11] [12]. Since the behaviour may be infinite such tools usually allow the user to restrict the extent of the resulted behaviour by initially specifying the maximal depth length the behaviour may include. In addition, the user has the possibility to select modularly, partly or completely the specification for simulation. For readability, the resulted behaviour may also be provided in an intermediate form, for instance, in form of EFSMs or SLTSs (Structured Labelled Transition Systems). The later support the formal semantics of FDTs. The result of the simulation is a simulation tree that can be easily seen as a set of traces (event sequences).

Fig. 3 gives an example of this intermediate form (EFSMs) that is resulted from the simulation of behaviour of process in Fig. 2.

```
specification prozess[g2,g3,g1,g4](y:nat):noexit
behaviour
state1[g2,g3,g1,g4](y)
where
      process state1[g2,g3,g1,g4](y:nat):noexit:=
                  g1?x:nat[(x ge 0) and (x le 3)];
            state2[g2,g3,g1,g4](y,x)
      endproc
      process state2[g2,g3,g1,g4](y:nat,x:nat):noexit:=
                  [(y ge 1)]-> g2!y;
            state3[g3,g1,g4](y,x)
      endproc
      process state3[g3,g1,g4](y:nat,x:nat):noexit:=
                  [(x ge 1)]-> g1!x;
            state4[g4](y,x)
                  [] i;
            state5[g1](x)
                  [] [(x gt 3)]-> g3!x;
            state6[g1](x,y)
      endproc
      process state4[g4](y:nat,x:nat):noexit:=
                  [(y ge x) and (x le 3)]-> g4!y;
            stop
      endproc
      process state5[g1](x:nat):noexit:=
                  i;
            state7[g1](x)
      endproc
      process state6[g1](x:nat,y:nat):noexit:=
                  [(x gt 2) and (y le 1)]-> g1!x;
            stop
                  [] [(x le 2)]-> g1!x;
            stop
      endproc
      process state7[g1](x:nat):noexit
                  [(x le 1)]-> g1!x;
            stop
      endproc
endspec
```

Figure 3. Behaviour in an intermediate form

Now, it is possible to identify the set of analysis purposes and the set of the analysis cases on the basis of the behaviour traces (or shortly traces) w.r.t. to the concept of the analysis purposes defined in Section 2.

Fig. 4 shows an example of the unfolded behaviour that represents a simulation tree and thus, interpreted in terms of traces (σ_i, $1 \le i \le 4$). These are presented in a suitable form for applying the principle of the purpose-directed analysis approach as previously explained.

Note that s_i in Fig 4 represent states whereas t_i transitions.

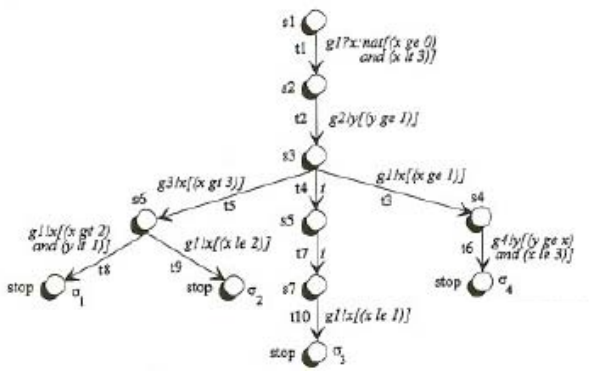

Figure 4. Unfolded behaviour of the example

The second part of the analysis process consists of performing the set of the analysis cases corresponding to the set of the analysis purposes on the system under analysis. This results in the assignments of verdicts to each applied analysis case. Depending on the given analysis purpose and on the fact whether the analysed state has been reached or not, a verdict of "pass" or "fail", respectively, is associated to the given analysis purpose.

In addition to the statement of the reachability for a given system under analysis, the detection of non-executable branches allows also to deduce a specification whose all transitions are executable. This is obtained by eliminating all non-executable transitions and their descendants.

IV. Conclusion

In this paper we present a new approach for reachability analysis that, in contrast to the common reachabilty analysis approach, is purpose-directed. This approach implies particularly less analysis efforts and resource requirements and lower costs. Indeed, in this approach the analysis is focused only on the system parts assumed to be vulnerable for reachability.

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

In future works, we intend to apply the principle for concrete cases, in particular for compilers.

References

[1] H. Xi, Dead Code Elimination through Dependen Types, *Lecture Notes in Computer Science*, v.1551, p228-242, 1999.

[2] J. Espraza and C. Schröter, Reachability Analysis Using Net Unfolding, *Fundamenta Informaticae*, 2002.

[3] K. Heljanko, Using Logic Programs with Stable Model Semantics to Solve Deadlock and Reachability Problems for 1-Safe Petri Nets, *TACAS'99*, 1999.

[4] V. Khomenko and M. Kounty, LP Deadlock Checking Using Partial Order Dependencies, *CoCur'2000*, LNCS 1877.

[5] K. Varpaaniemi, K. Heljanko and J. Lilius, PROD 3.2 – An Advanced Tool for Efficient Reachability Analysis, *Lecture Notes In Computer Science*, v.1254, p. 472 – 475, 1997, ISBN:3-540-63166-6.

[6] B. Sarikaya et al. Method of analysing extended finite-sate machine specifications. Computer Communications, v.13 n.2, p83-92, March 1990.

[7] K. C. Tai and P. V. Koppol. Hierarchy-Based Incremental Analysis of Communication Protocols. Proc. Of Intl. Conf. On Network Protocols, 1993.

[8] Specification and Description Language SDL'92, *ITU-T Recommendation Z.100*, 1992.

[9] Information processing systems – Open Systems Interconnection – Estelle: A formal description technique based on an extended state transition model, *International Standard ISO 9074*, 1989.

[10] ISO TC97/SC21: Guidelines for the application of Estelle, LOTOS and SDL (ISO TR 10167, 1990)

[11] B. Sarikaya, Principles of protocol engineering and conformance testing, *Ellis Horwood*, London, 1993.

[12] P.H.J.v. Eijk, Tools for LOTOS – a Lotosphere overview, *Memoranda informatica 91-25*, University of Twente, 1991.

[13] A. Groce, W. Visser, Heuristics for Model Checking Java Programs, *International Journal on Software Tools for Technology Transfer (STTT)*, v. 6, n. 4, p.260-276, Springer Berlin/Heidlberg, 2004, ISSN: 1433-2779.

[14] W. Damm and D. Harel, LSCs: Breathing Life into Message Sequence Charts, *Formal Methods in System Design,* 19(1):45-80, 2001.