Invariants in symbolic modeling and verification of requirements

Alexander Letichevsky[1], Alexander Godlevsky[1], Anton Guba[1], Alexander Kolchin[1], Oleksandr Letychevskyi[1], Vladimir Peschanenko[2]
[1] V.M. Glushkov Institute of cybernetics, Kiev, Ukraine
let@cyfra.net, godl@iss.org.ua, antonguba@ukr.net, kolchin_av@yahoo.com, lit@iss.org.ua
[2] Kherson state university, Kherson, Ukraine
vladimirius@gmail.com

Abstract

The paper presents the usage of invariants technique for symbolic modeling verification of requirements for reactive systems. It includes checking of the safety, incompleteness, liveness, consistency properties, and livelock detection. The paper describes the iterative method of double approximation for invariants generation and illustrated by examples of invariants technique usage for symbolic verification.

1. Introduction

The initial step in the modern software development process is the stage of gathering and verification of requirements. During this stage the specifications of the product are created on some level of abstraction that is understandable and useful for the customer. Traditionally, the original requirements created by the customer and, possibly, with the participation of the developer, are the technical requirements for the coding, as well as for creation of a product design model, if such is required. Moreover, original requirements are the basis for creation of tests that are developed by a team of test engineers on a testing stage.

We consider the verification of requirements for reactive systems, i.e. systems that repeatedly receive input from the external environment as a stimulus and react on it. The observed reaction is in the form of a message exchange.

The main properties that are considered at the stage of requirements gathering are:

1. Completeness property that guarantees that a scenario, leading to a state that is not represented by the requirements, is not possible. In other words, this property guarantees the absence of deadlocks or states in which the behavior of the system is not specified by requirements.
2. Consistency property that guarantees the absence of a scenario, leading to a state where an ambiguous response on the same external stimulus is possible. In other words, this property specifies the absence of non-determinism.
3. Safety property that should not be violated for states reachable in all possible scenarios. This property is defined by the user as a control condition of the adequacy of the initial requirements to the customer's objectives.
4. Livelock property that means the existence of a scenario, leading to a state where only the repetition of the same scenario is possible.
5. Liveness property that means the existence of a scenario, leading to a state where a certain property of system is reachable.

We use dynamic methods which are based on the modeling of requirements presented in a formal way. However, for systems with infinite number of states due to inability to reach all of the states, it is needed to abstract from some entities to cover all scenarios or consider incomplete set that does not guarantee reachability of a property.

Static methods are based on the proof of the properties after the analysis of the formal requirements. However, these methods are also not so efficient due to false negatives.

The method that allows reducing a set of checked scenarios in reactive systems is a method of invariants which includes some of the features of proving and modeling approaches.

2. Formal Presentation of Requirements by Basic Protocols Specifications

The *model* of system is defined as a transition system which has the formulas of some typed logic language **L** as states. The language **L** used in our implemented system so far contains integer, real, enumerated, Boolean, and symbolic types with linear arithmetic operations and inequalities for arithmetic types, logical connectors for Booleans, and equalities for all types. Arrays are considered as functions with restricted domains for indices. Formulae allow quantifiers with some restrictions.

The transitions of a system over the language **L** are defined by means of a set of basic protocols of the system. Each basic protocol has the form

$$d = \forall x(\alpha(x) \rightarrow < A(x) > \beta(x))$$

where $x = (x_1 : \tau_1, x_2 : \tau_2, ...)$ is a set of typed variables of the language **L**, $\alpha(x)$ and $\beta(x)$ are formulas of **L**, $A(x)$ is a symbolic term which presents the event in the system. A(x) could be omitted. It shall be written as <>. The formulas $\alpha(x)$ and $\beta(x)$ are called pre- and postconditions of a basic protocol and the list *x* is called the parameter list of a *d*. Basic protocol can be associated with Hoare triple, describing the semantics of transition of the system or with logic form of production rule in expert system, or simple temporal logic formula describing the property of the next state of the system.

The general theory of basic protocols is presented in [2]. As the transition transforms one formula to another in [3] a term *predicate transformer* was used. Thus, to compute transitions between the states of such models, basic protocols are interpreted as predicate transformers: for a given symbolic state of the system and a given basic protocol the direct predicate transformer generates the next symbolic state as its strongest postcondition, and the backward predicate transformer generates the previous symbolic state as its weakest precondition. These concepts have been implemented in VRS (Verification of Requirement Specifications) system [4] and IMS (Insertion Modeling System) system [5].

3. Invariants in Basic Protocols

Research of problems in automatic program invariants generation for a variety of data algebras was performed starting from the 70s years in the Institute of Cybernetics of NAS of Ukraine. Their main results are presented in [6, 7].

In this section the dynamic iterative method of invariants generation is considered. This method is based on the mentioned above work and is an adaptation of it for requirements verification.

Pre- and postinvariants of basic protocols are considered as invariants. Formula φ is a *preinvariant* of a requirement *d* if it is valid each time before the **successful** application of this requirement. Formula φ is a *postinvariant* of a requirement *d* if it is valid each time after the **successful** application of requirement *d*.

The strongest invariant is such that all other invariants are the consequences of this one.
For verification purposes we are talking about finding the strongest invariants and their use in testing the properties of systems. Hereinafter the term “invariant” means postinvariant.

To determine the reachability of non-determinism property we consider the formula: PreInv (di) & PreInv (dj) for each pair of basic protocols di and dj in the existing formalization. If the formula is satisfiable, then the corresponding pair of basic protocols represents reachable non-determinism.

In the proof of completeness of the basic protocols we consider the disjunction of preinvariants ~∨PreInvariant(di). If the formula is satisfiable, it represents the incompleteness.

If we check the safety violations of formula S, we check the satisfiability of the formula S&PreInvariant(d)). If it is satisfiable, we have a safety violation, and it is possible to construct a path leading to the initial state from preinvariant.

Similarly we check the validity of liveness properties by determining the satisfiability of conjunction of the invariant and desired property. Scenario demonstrates that the reachability property is determined as a path in the graph. Livelock is considered on the set of cycles, which can be built simultaneously with the generation of invariants. The research on livelock detection and its reachability using invariants technique is in progress.

Invariants are presented in the context of the checking of reachability problem of goal state from *initial*, i.e. the aim is not only to construct the invariant itself, and as soon as possible, but to return a verdict about unreachability of the *goal* or, conversely, about its reachability.
Let us introduce the following notation:

M – model, which consists of: environment description *env*, initial environment state *initial* and a set of basic protocols *{B};*

bp_i – i-th basic protocol (for simplicity, let us assume that all the basic protocols are numbered from 1 to k), $bp_i \in \{B\}$; also we introduce the additional basic protocol bp_0 *1-> <> initial;*

n – iteration number. Initial value is *0;*

$ModWeakPostInv^{(n)}$ – invariant formula for the whole model *M*, which is obtained by a weak approximation method, on the *n-th* iteration;

$ModStrongPostInv^{(n)}$ – invariant formula for the whole model *M*, which is obtained by a strong approximation method, on *n-th* iteration*;*

$pt(state, bp_i) \rightarrow state'$ – symbolic environment state, which is obtained after applicability of bp_i to the state, which is defined by the formula state. Expression $pt(1,bp_i)$ means that bp_i is applicable the to state, which is always true. If a protocol is not applicable to the state *state*, then the result is *0.*

WeakComplete – an indicator if the invariants are equal on the last two iterations by the weak approximation method.

$succ(bp_i, bp_j)$ – succession relation of bp_j after bpi on current iteration. If the protocol bp_j *is* applicable after bp_i on current iteration, then $succ(bp_i, bp_j) = 1$, otherwise *0.* Initial value is *1* for any bp_i, bp_j. If protocol bp_j is applicable after *initial* , we notate it as $succ(bp_0, bp_j)$.

$WeakPostInv^{(n)}(bp_i)$ – postinvariant bp_i on *n*-th iteration by a weak approximation method.

$StrongPostInv^{(n)}(bp_i)$ – postinvariant bp_i on *n-th* iteration by a strong approximation method.

property – *f*ormula of property wanted to be checked, for example, goal state.

WeakCompleteSet – set of protocols for which *WeakPostInv* are equal.

WeakUncompleteSet – set of protocols for which *WeakPostInv* are not equal.

StrongCompleteSet – set of protocols for which *StrongPostInv* are equal.

StrongUncompleteSet – set of protocols for which *StrongPostInv* are not equal.

Let R be a set of all reachable states in M.

Invariants of the algorithm are the formulas that satisfy the following relations:

$ModUpPostInv^{(0)} \supseteq ModUpPostInv^{(1)} \ldots \supseteq ModUpPostInv^{(n)} \supseteq R$

$ModLowPostInv^{(0)} \subseteq ModLowPostInv^{(1)} \ldots \subseteq ModLowPostInv^{(n)} \subseteq R$

To solve the problem of reachability properties, invariants can be used as follows:

1) intersect *initial* with the property *goal*. If the formula is satisfiable, the property is reachable, and the reachability problem is solved. Otherwise, we need to perform one more iteration go to step 2.

2) *ModWeakPostInv* shall be intersected with *goal*. If formula *ModWeakPostInv & goal* is not satisfiable, the property is unreachable, and the reachability problem is solved. Otherwise, go to step 3)

3) *ModStrongPostInv* shall be intersected with *goal*. If formula *ModStrongPostInv & goal* is satisfiable, the property is reachable, and the reachability problem is solved. Otherwise, we need to perform one more iteration go to step 2).

If the reachability problem for all properties is solved, it is the additional condition of the end of loop iterations.

Let us briefly describe the essence of the algorithm. All protocols are applied to everywhere true state 1. Protocol, which could not be applied after state 1 are contradictory protocols. This is the zero iteration of weak approximation. Then, all noncontradictory protocols are applied to initial state. This is the zero iteration of the strong approximation. In the end of 0-th iteration a model invariant is constructed. The succession relation succ, which is an indicator if we need to apply the basic protocol after another or not, is used and precised on each iteration of weak approximation. Initially we shall apply each basic protocol after all basic protocols. On the next iterations the protocols are applied to the previous obtained approximations. The algorithm is iterative. It finishes when the verdict of the reachability or unreachability of all properties is given or a custom filter of maximum iterations is reached.

Let us consider an example. The initial state x=0. The basic protocol adds 2 to x if x is even. It is necessary to check the reachability of property x = 7.

We have an infinite loop, correspondingly, the modeling will not give a result. Using the above algorithm we will obtain on the 0-iteration that the weak approximation is equal to Exist (n:int) (x=2*n+2). x=7 does not satisfy this formula, so, this property will be always unreachable and the weak approximation will be refined on the next iterations.

Now we will check the reachability of state x=8. It satisfies the formula of the weak approximation but it includes unreachable states. So, we should build a strong approximation. It will be x=2 on the 0-iteration. We will continue the

iterations and obtain that x=8 is reachable. In a general case we can do the following conclusion that the result will be obtained sometimes on an initial iteration. Sometimes a large number of iterations is needed, and the complexity will be compared with searching of all the state space of the given model.

4. Conclusions

The paper describes the use of invariants generation methods for symbolic verification of requirements. A new iterative method of double approximation has been described and implemented in the IMS (Insertion Modeling System) [1]. The nearest plans are to use the idea of usage of undetermined coefficients method. The authors continue their research to find suitable algorithms of static invariants generation. This technique may be good for system with infinite number of states, especially, for those systems where we can observe the changing of such physical values as time, distance, velocity, temperature, etc. Checking of reachability of properties of systems defined by basic protocols for which the invariants are generated could be performed efficiently by finding of satisfiability of formula that presents the system state.

References

1. APS and IMS Systems. http://apsystem.org.ua, last viewed May 2013.
2. Letichevsky, J. Kapitonova, V. Volkov, A. Letichevsky Jr., S. Baranov, V. Kotlyarov, T. Weigert. System Specification with Basic Protocols. Cybernetics and System Analyses, vol. 4, 2005, p. 3-21.
3. Letichevsky, A. Godlevsky, A. Letichevsky, S. Potienko, V. Peschanenko. Properties of Predicate Transformer of VRS System. Cybernetics and System Analyses, vol. 4, 2010, p. 3-16.
4. Verification for Requirement Specification (VRS). http://iss.org.ua ISS/VRS/tool.htm}, last viewed May 2013.
5. Letichevsky. Algebra of behavior transformations and its applications. In V.B.Kudryavtsev and I.G.Rosenberg eds. Structural theory of Automata, Semigroups, and Universal Algebra, NATO Science Series II. Mathematics, Physics and Chemistry, vol. 207, 2005, p. 241-272.
6. Letichevsky A.A. About one approach to program analysis. Cybernetics, vol. 6, 1979, p.1-8.
7. Letichevsky A., Lvov M. Discovery of Invariant Equalities in Programs over Data Fields. Applicable Algebra in Engineering, Communication and Computing, vol. 4, 1993, p.21-29.