Dependability of Relational Safety-Critical Programs

Farokh B. Bastani, Victor L. Winter, and I-Ling Yen*

1 Introduction

Software for safety-critical systems must be highly reliable since failures can have catastrophic consequences. While existing methods, such as formal techniques and testing, can significantly enhance software reliability, they have some limitations in achieving ultra-high reliability requirements.

One approach that works for hardware systems is to decompose the system into independent components, evaluate each component separately, and then infer the dependability of the system from the properties of its components. One of the earliest works related to software decomposition is [5] where a requirements specification is decomposed into multiple views, each of which captures some behavior of the system. The concept of multiple views has also been used in State-Charts [2], separation using rely-guarantee assertions [4], behavioral inheritance [1], and Aspect-Oriented Programming [3]. These decompositions reduce the complexity of the system, but two different views are not necessarily independent. This complicates the dependability analysis. For example, if the reliabilities of components \( f \) and \( g \) are 1.0 and 0.9999, respectively, then the reliability of a system consisting of \( f \) and \( g \) can range from 1.0 to 0.0 depending on how \( f \) and \( g \) interact. This uncertainty means that considerable effort has to be expended to reason about global properties of the system rather than by simple deductions from the component properties.

2 Relational Control Programs

Consider a process-control program \( P \). Let \( S(t) \) denote the time-varying state space of the system and \( s(t) \) denote the actual state of the system at time \( t \). \( s(t) \) describes a trajectory of the system through its state space which can be divided into goal states, constraint states, and free states. The purpose of a process-control program is to determine a trajectory that passes through the free states and reaches a goal state.

The top level specification, \( \gamma \), can be decomposed into a conjunction of predicates, \( \gamma = \gamma_1 \land \gamma_2 \land \cdots \land \gamma_n \). This includes decomposition of the system constraints as well as the goals. The individual predicates, \( \gamma_i \), can be further decomposed into a disjunction of predicates, i.e., \( \gamma_i = \gamma_{i1} \lor \gamma_{i2} \lor \cdots \lor \gamma_{in_i} \). Each predicate \( \gamma_{ij} \) represents one way of achieving \( \gamma_i \). Note that, conjunctive and disjunctive decompositions can be applied to the specification iteratively as necessary.

Let \( S_{ij}(t) \) denote the view of the state space that only reflects the goal or constraint specified by predicate \( \gamma_{ij} \), i.e., \( S_{ij}(t) \) has the same state space as \( S(t) \), but all its states other than those in \( \{ x \in S(t) \mid \gamma(x) \} \) are free states. Let \( P_{ij} \) be a program that solves the limited control problem expressed in \( S_{ij}(t) \). In conventional programs, \( P_{ij} \) is viewed as a function. There is no obvious mathematical model for merging independently developed \( P_{ij} \)'s into the overall system program \( P \) since the output of the \( P_{ij} \)'s may be incompatible with each other. In our approach, \( P_{ij} \) is viewed as a general relation and, hence, it returns the set of all output values for each input. \( P \) can be obtained by simply forming the intersection of the output sets of \( P_{ij} \)'s, \( P \equiv P_1 \cap P_2 \cap \cdots \cap P_n \), where \( P_i \) is the program for achieving \( \gamma_i \). Similarly, each \( P_i \) can be obtained via a systematic union operation from its components, i.e., \( P_i \equiv P_{i1} \cup P_{i2} \cup \cdots \cup P_{in_i} \).

3 Relational Hybrid Finite State Machines

It is difficult to automatically convert goal-oriented specifications into code without a substantial knowledge-base (and associated inference engine) for the application domain. There are various formalisms that one can use to express system knowledge. One practical approach is to use a Finite State Machine paradigm to describe the state space of a system.

**Definition 2.1.** A Finite State Machine (FSM) model for a process-control system consists of a set of states, \( S' = \{ s'_1, s'_2, \ldots, s'_k \} \) and a set of transitions, \( T' = \{ t'_{ij} \mid 1 \leq i, j \leq k \} \). Associated with each transition, \( t'_{ij} \), is a control command vector, \( ctl(t'_{ij}) \).

The meaning of \( t'_{ij} \) is that if the state of the machine is \( s'_i \) and \( ctl(s'_ij) \) has the given value, then the new...
state will be \( s_i^j \).

Given the presence of continuous variables in many control systems, it is necessary to extend the FSM model to a Hybrid FSM (HFSM) model where continuous variables are associated with each state.

**Definition 2.2.** A Hybrid FSM (HFSM) is an FSM where a vector of continuous variables is associated with each state. A transition \( t_{ij} \) is an instantaneous transition if the state changes to \( s_i^j \) whenever \( c(t) \in \mathcal{T}(t) \) has the given value. A transition \( t_{ij} \) is an eventually transition if the state will eventually change to \( s_i^j \) and \( c(t) \in \mathcal{T}(t) \) has the given value continuously in state \( s_i^j \). A state is a boundary state if all transitions are instantaneous; otherwise, it is an interior state.

In order to fit the notion of an abstract algorithm within our relation-based model of computation, we further extend the HFSM and introduce relational HFSMs defined as follows.

**Definition 2.3.** A Relational HFSM (RHFSM) is an HFSM where \( c(t) \in \mathcal{T}(t) \) can be a set of values rather than a single value. Given an RHFSM, it is possible to automatically generate the code through relational composition.

## 4 Dependability Analysis

### 4.1 Reliability Analysis

Let \( s_i, 1 \leq i \leq n \), be the states in an RHFSM and let \( t_{ij}, 1 \leq i, j \leq n \), denote the transitions. To analyze the reliability, the following inputs are needed:

1. \( \pi_i \), the probability of starting in state \( s_i \); and \( p_{ij} \), the conditional probability of selecting transition \( t_{ij} \) given that the state is \( s_i \); these data must be provided by the domain expert;
2. \( r_{ij} \), the reliability of each transition in the RHFSM, and \( c_i \), the probability that state \( s_i \) has been correctly classified; these data are obtained from the analysis of the RHFSM.

The reliability of the RHFSM is the probability that the selected trajectory (a) is achievable, (b) does not pass through any constraints, and (c) ends in a goal state. This is computed in the following way. Let \( T_i \) denote the set of trajectories to a goal state from state \( s_i \).

1. Reliability of trajectory \( x \in T_i \),
   \[ R(x) = \prod_{1 \leq j \leq |\mathcal{T}(x)|} r_{ij}(x_{i+1}) \]
2. Probability of selecting trajectory \( x \in T_i \),
   \[ p(x) = \prod_{1 \leq j \leq |\mathcal{T}(x)|} p_{ij}(x_{i+1}) \]
3. Reliability of the RHFSM, \( P_i \),
   \[ R(P_i) = \sum_{x \in T_i} \pi_i p(x) R(x) \]

The system level reliability follows from the component level reliability in a simple way.

**Theorem 4.1.** If \( P_1 \) and \( P_2 \) are two relational programs, then \( R(P_1 \cap P_2) = R(P_1) \times R(P_2) \), provided \( P_1 \) and \( P_2 \) have independent failure processes.

**Theorem 4.2.** If \( P_1 \) and \( P_2 \) are two relational programs, then \( \max\{R(P_1), R(P_2)\} \geq R(P_1 \cup P_2) \geq R(P_1) \times R(P_2) \), provided \( P_1 \) and \( P_2 \) have independent failure processes.

### 4.2 Safety Assurance

The safety of each component is assured independently, i.e., by finding the set of unsafe states and ensuring that there are no transitions from any reachable states to the unsafe states.

**Theorem 4.3.** The safety of a disjunctive decomposition of \( S_i(t) \) into \( S_j(t) \), \( 1 \leq j \leq n \), follows from the safety of its components if the set of unsafe states in \( S_j \), \( 1 \leq j \leq n \), is identical to the set of unsafe states in \( S_i(t) \).

**Theorem 4.4.** The safety of a conjunctive decomposition of \( S_i(t) \) into \( S_j(t) \), \( 1 \leq i \leq n \), follows from the safety of its components if the components are ranked in a priority order and, in the intersection operation, the more safety-critical version overrides a less critical version if the intersection is \( \phi \).

## 4.3 Stability Analysis

The stability of the individual components is analyzed by determining whether the goal states are reachable from all possible states, including failure states. Unlike other attributes, the stability of the system is more difficult to show in general.

**Theorem 4.5.** If \( P_1 \) and \( P_2 \) are stable relational programs, then \( P_1 \cap P_2 \) is stable provided that there are no \( \phi \) outputs.

## References