Available online at www.sciencedirect.com

SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science 141 (2005) 91–106

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

A Formal Approach for Analysis and Testing of Reliable Embedded Systems

Abdelaziz Guerrouat[1]

*Department of Computer Science
Clausthal University of Technology
38678 Clausthal, Germany*

Harald Richter[2]

*Department of Computer Science
Clausthal University of Technology
38678 Clausthal, Germany*

Abstract

In this paper, a framework for the specification of embedded systems described as 'predicated' extended finite state machines (p-EFSMs) is proposed. Compared to simple FSMs, p-EFSMs allow the control flow and the data flow description of hardware modules or software processes. We introduce a new variant of the EFSM model, a so-called 'predicated' EFSM that extends the usual EFSM. This extension offers a more convenient mean to specify constraints on the system's transitions. Secondly, it provides an easy mapping onto formal description techniques. Thirdly, it allows the development of an embedded system independently from the implementation, i.e. without favoring a hardware or a software. Crucial tasks in the design of reliable embedded systems are analysis and testing. These allow the system developer to detect bugs that may be very costly to do in subsequent phases of the system development. We identify the different testing issues and demonstrate how the bugs can be detected by means of p-EFSMs. Failure detection and elimination improve the likelihood of the well-functioning and the reliability of the embedded system.

Keywords: Embedded systems, analysis and testing, extended finite state machines, formal description techniques.

[1] Email: aguerrou@informatik.tu-clausthal.de
[2] Email: richter@informatik.tu-clausthal.de

doi:10.1016/j.entcs.2005.02.050

1 Introduction

An embedded system represents a part of a product with which an end user does not directly interact. Nowadays, there exists a lot number of typical applications of industrial embedded systems that are supported by software, for instance adaptive cruise control systems, automotive systems and nuclear power plants. These are mostly characterized by safety-critical properties.

Because of the growing complexity of embedded systems the software development process becomes a costly and error-prone activity. The cost factor plays a central role in today's industrial competition, for instance between car manufacturers. The development of competitive and efficient products is imposing more and more constraints to the design of embedded systems. One of the means to reach this goal are formal methods to support the different phases of system development, i.e. specification, synthesis and testing. There are several requirements for those methods that should be among others qualities abstract, understandable, analyzable, scalable and unambiguous specification formalisms.

Much work should be done for introducing standardized formal description techniques (FDTs) [6] [10] and related formalisms at a high level of abstraction in the life-cycle development of embedded systems. On the other side, formal descriptions techniques have demonstrated their effectiveness in the analysis of complex requirements like those for communicating systems [5] [6] [10]. Furthermore, they provide a solid mean for unambiguous specification and rigorous analysis. They are based on formal methods such (E)FSMs ('extended' finite state machines) and differ from conventional programming languages by providing not only a formal syntax but also a formal semantic. Moreover, the application of formal specification increases the confidence in the software and the system. Especially in the area of safety-critical systems, the use of formal techniques is highly recommended [2], e.g. steer-by-wire or brake-by-wire in cars [9].

Statecharts as a semi-formal model is actually the mostly used formalism to specify requirements for embedded systems [7]. Although Statecharts provide graphical facilities, they might lack formal and unambiguous semantics. Therefore, detecting bugs, incompleteness and inconsistencies becomes a difficult task. Furthermore, they are only used to describe behavioral requirements. To alleviate these lacks many authors try to combine formal notations like Z with state-transition models [8]. Z is based on set theory and first order predicate logic and used for data structuring and abstracting. Petri Nets have been also used to verify the correctness of embedded systems, e.g. [3]. However, approaches developed around this model do not directly deal with the here addressed standardized FDTs and related test generation methods.

Formal methods have been widely and successfully used in other areas for both, software and hardware design and testing. We think that they may be adapted for embedded systems too.

The finite state machine model is very popular in the control flow specification of state/transition-based systems and many related analysis methods have been developed [1] [4]. These support a formal test derivation which can be used for validation and testing purposes. However, finite state machines lack to deal with the data flow. This shortcoming can be alleviated by using the extended finite state machine model. Moreover, the test generation cannot be easily applied in this case.

In this paper, we discuss the testing problems for embedded systems modeled as extended finite state machines and propose a framework to deal with them. We first present the main components of an embedded system as well as their interactions. Furthermore, the testing issues based on so-called p-EFSM (predicated EFSM). The later, represents a variant of the usual EFSM are identified and the appropriate solutions will be discussed.

The paper is organized as follows. Section 2 reviews the conventional finite state machine and the extended finite state machine models and gives a short comparison. Modeling embedded systems based on p-EFSM are presented in Section 3. Section 4 identifies the main properties of an embedded system to be tested. After that, the testing principle is explained. Finally, Section 5 concludes the paper.

2 Preliminaries

In this section, we give the definitions of the used formalisms. We first review the finite state machine model and its extension. After that, we propose a definition of the p-EFSM ('predicated' EFSM).

Definition 2.1 A *finite state machine* (FSM) is a 5-tuple $\langle S, I, O, T, s_0 \rangle$, where S is a non-empty finite set of states, I a non-empty set of inputs, O a non-empty finite set of outputs, $T \subseteq S \times I \times O \times S$ the set of transition relations, and $s_0 \in S$ the initial state of the FSM.

Note, in some literature the set of outputs may be considered as not a part of an FSM. However, the definition of an FSM given here appears to be more convenient in FDTs.

A transition $t \in T$ of an FSM is a 4-tuple $\langle s, i, o, s' \rangle$, where $s \in S$ is a current state (the edge), $i \in I$ an input, $o \in O$ an output related to s and I, and $s' \in S$ the next state (a tail state) related to s and i.

The FSM model is often less appropriate for specifying most parts of a system, for instance, regarding the data flow. Therefore, FSMs were extended to improve their description capability by using additional state variables and interaction parameters. Such variables are used in programming languages specifying conditions on transitions and calculations carried out during transitions.

Definition 2.2 An *extended finite state machine* (EFSM) is a 7-tuple $\langle S, C, I, O, T, s_0, c_0\rangle$ where S is a non-empty set of main states, $C = dom(v_1) \times \ldots \times dom(v_n)$ a non empty countable set of contexts with $v_i \in V$, V the non-empty finite set of variables and $dom(v_i)$ a non-empty countable set referred to as the domain of v_i, I a non-empty finite set of inputs, O a non-empty set of outputs, $T \subseteq S \times C \times I \times O \times S \times C$ the set of transition relations, $s_0 \in S$ the initial main state, and $c_0 \in C$ the initial context of the EFSM.

A main state may consist of sub-states. A context is a specific assignment of values to the variables. A transition $t \in T$ of an EFSM is a 6-tuple $\langle s, c, i, o, s', c'\rangle$ where $s \in S$ is a current main state, $c \in C$ a current context, $i \in I$ an input, $o \in O$ an output, $s' \in S$ a next main state, and $c' \in C$ a next context.

A transition may be characterized, in addition to its current and next state and input and output interactions and context, by a so-called enabling predicate. This represents a condition on a state transition and the related output to be carried out, once the predicate 'fires'. All usual logical and comparative operators `and, or, =, >` etc. are allowed in a predicate. Thus, a transition takes place only if its enabling predicate fires. It depends on the current FSM state with additional variables and the concrete variables values (context) of the input. Therefore, we introduce a so-called p-EFSM in which the enabling predicates on transitions are explicitly represented.

Definition 2.3 A *predicated extended finite state machine* (p-EFSM) is an 8-tuple $\langle S, C, I, P, O, T, s_0, c_0\rangle$ where S is a non-empty set of main states, $C = dom(v_1) \times \cdots \times dom(v_n)$ a non-empty countable set of contexts with $v_i \in V$, V a non-empty finite set of variables, and $dom(v_i)$ a non-empty countable set referred to as the domain of v_i, P a countable set of predicates (possibly empty), I a non-empty finite set of inputs, O a non-empty finite set of outputs, $T \subseteq S \times C \times I \times P \times O \times S \times C$ a set of transition relations, $s_0 \in S$ the initial main state, and $c_0 \in C$ the initial context of the p-EFSM.

A transition $t \in T$ of a p-EFSM is a 7-tuple $\langle s, c, I, p, o, s', c'\rangle$ where $s \in S$ is a current main state, $c \in C$ a current context, $i \in I$ an input, $p \in P$ an enabling predicate which depends on the context c, $o \in O$ an output, $s' \in S$ a next main state, and $c' \in C$ a next context.

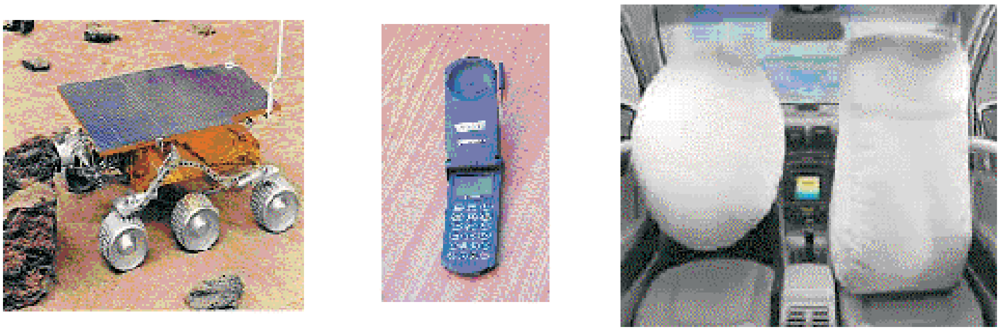

Fig. 1. Examples of objects including embedded systems

Note if $P = \emptyset$ then the p-EFSM is equal to the conventional EFSM. Therefore, the p-EFSM provides a more generalized specification mean. The system specification using a p-EFSM is more suitable for using formal description techniques like Estelle or SDL [6] [10].

Estelle is a standardized formal description technique (International Standard ISO 9074) based on concepts of structured communicating extended state automata and Pascal. It is oriented towards the specification of complex distributed systems, in particular communicating systems. A specified system is presented as a tree of tasks where each task has a fixed number of input/output access points (interaction points). Within a specified system it exists a fixed structure of subsystems (sub-trees of tasks) and communication links between subsystems.

SDL (Specification and Description Language) is an object-oriented, formal language defined by The International Telecommunications Union Telecommunications Standardization Sector (ITU) (formerly Comité Consultatif International Télégraphique et Téléphonique [CCITT]) as recommendation Z.100. The language is intended for the specification of complex event-driven real-time, and interactive applications involving many concurrent activities that communicate using discrete signals.

Indeed, p-EFSMs can functionally describe system components that may be blocks or modules depending on the used formal description technique[3].

3 Characteristcs and Structure of Embedded Systems

An embedded system is any computer system or computing device that performs a dedicated function or is designed for use with a specific embedded software application, e.g. Car, PDA (Personal Data Assistant), Mobile Phone,

[3] SDL uses the 'block' concept whereas Estelle 'module'

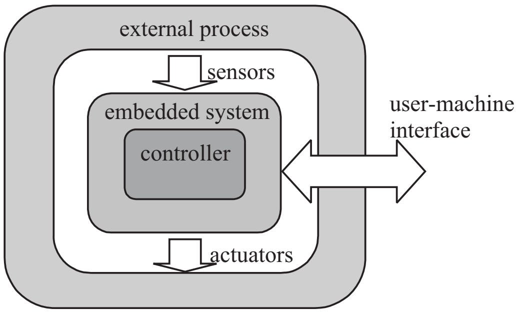

Fig. 2. Basic structure of an embedded system

E-Book (Electronic Book), Robot, etc. (Figure 1). That is, an embedded system is a special-purpose system built into a larger device. It is embedded as a subsystem in a larger system which may or may not be a computer system. An embedded system is typically required to meet specific requirements. In an embedded mechatronic system, a microcontroller or computer system performs a dedicated function for an appliance or a gadget such as a car's brake or a steering wheel [9].

Embedded systems must usually be dependable, efficient and must meet real-time constraints. Be 'dependable' means that an embedded system must be reliable, available and safe. The efficiency mostly concerns properties like energy, code-size, run-time, weight and cost. An embedded system is dedicated for a certain application and characterized also by a dedicated user interface. Thus, knowledge about future behavior at design time can be used to minimize resources and to maximize robustness. Many embedded systems must meet real-time constraints. A real-time system must react to stimuli from the controlled object (or the operator) within the time interval dedicated by the environment.

Embedded systems are frequently connected to a physical environment through sensors and actuators. They are typically reactive systems. A reactive system is in continuous interaction with its environment and executes at a pace determined by that environment. The behavior depends on input and current state for which the automata model is often most appropriate.

Figure 2 illustrates the basic structure of an embedded system comprising an *external process*, *sensors*, *actuators*, and a *controller*:

- The *external process* is a process that can be of physical, mechanical, or electrical nature.
- *Sensors* provide information about the current state of the *external process* by means of so-called *monitoring events*. They are communicated to the *controller*. For the *controller*, they represent input events. They are

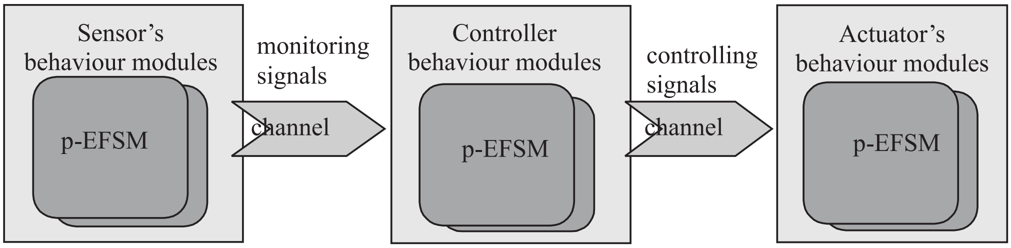
Sensor's behaviour modules
p-EFSM
monitoring signals
channel
Controller behaviour modules
p-EFSM
controlling signals
channel
Actuator's behaviour modules
p-EFSM

Fig. 3. Embedded system based on p-EFSM modeling

considered as stimuli for the *controller*.

- The *controller* must react to each received event, i.e. input event. Events originate usually from *sensors*. Depending on the received events from *sensors*, corresponding states of the *external process* will be determined.
- *Actuators* receive the results determined by the *controller* which are communicated to the *external process* by means of so-called *controlling events*.

The external process is usually given in advance. In contrast, the controller is often implemented by real-time hardware and software. This should allow each modification of the controller algorithm in a straightforward way each time this is needed. The controller's behavior is depending on that of the external process. The controller commands the behavior of the external process taking into consideration requirements on the process and its characteristics, such as physical laws, real time and other constraints.

Formal Description

The embedded system specification consists of the specification of its environment and its controller. We assume that the embedded system is state-transition based because the automata model is efficiently more appropriate. Thus, its behavior description will be based on the proposed p-EFSM model. This consists of a set of modules where each module describing a given function is modeled as a one or many p-EFSM (Figure 3). These modules interact with each other via broadcasting events. However, a sequence has to be respected in this communication. For instance, the direct communication of a module of an actuator with a sensor is not allowed.

The most important component of an embedded system consists of the controller which communicates with its environment, i.e. sensors and actors, via signals (i.e. events). To be recognized by all components, these events have to be declared as global variables for adjacent p-EFSMs. The events output from sensors represents input events for the controller. The events from the controller to the actuators are output events and represent input events for the actuators. They result from new computations performed by the controller that is triggered by the received input events.

Depending on the nature of sensor events (e.g. indicating the power on/of state for an electrical unit, the speed of a mobile object such as a car, etc.) the corresponding p-EFSM of this component is triggered and the concerning transition(s) are performed. This triggers the p-EFSMs of the controller whose states change. Depending on the received events, transitions in the p-EFSMs are executed. Note, that transitions in the controller can spontaneously be triggered by other events, e.g. time out. The modeled subsequent state of the external process is computed and communicated as output events via the actuators.

Now we use the p-EFSM to model a given embedded system. We consider a single p-EFSM for each component of the system and denote them with indices s, c and a for sensors, controller, and actuators.

Interdependencies between these components are described as follows:

- Let be given a transition $t_s \in T_s : t_s = \langle s_s, c_s, i_s, p_s, o_s, s'_s, c'_s \rangle$ with
  $s_s \in S_s, c_s \in C_s, i_s \in I_s, p_s \in P_s, o_s \in O_s, s'_s \in S_s, c'_s \in C_s$
  $\Rightarrow \exists t_c \in T_c | o_s \equiv i_c$

  That is, each output event generated by sensors must trigger a transition of the controller. This event represents an input event for the triggered transition. We assume here that the predicates related to the transitions are satisfied by the actual context.

- Let be given a transition $t_c \in T_c$ with
  $s_c \in S_c, c_c \in C_c, i_c \in I_c, p_c \in P_c, o_c \in O_c, s'_c \in S_c, c'_c \in C_c$
  if $i_c \in O_s \Rightarrow t_s \in T_s$ and $i_c \equiv o_s$

  This means that, if there exists a transition of the controller whose input event belongs to the set of output events of the sensors then it must exist a transition of the sensors whose output event is identified with the given event.

- Let be given a transition $t_a \in T_a : t_a = \langle s_a, c_a, i_a, p_a, o_a, s'_a, c'_a \rangle$
  with $s_a \in S_a, c_a \in C_a, i_a \in I_a, p_a \in P_a, o_a \in O_a, s'_a \in S_a, c'_a \in C_a$
  $\Longrightarrow t_c \in T_c : t_c = \langle s_c, c_c, i_c, p_c, o_c, s'_c, c'_c \rangle$ and $o_c \equiv i_a$

  Each transition of actuators must be only triggered by the controller and must match the output event of the triggering transition of the controller.

It is easy to map a p-EFSM specification on the behavioral part (transition part) of an Estelle module. The later has the following structure which is composed of two parts - condition clauses and actions:

```
WHEN clause
```

```
        when interaction_point_id.interaction_name
    FROM clause
        from state
    PROVIDED clause
        provided Boolean expression
    DELAY clause
        delay (integer_expression)
    TO clause
        to state
        output
```

The `when` corresponds to input events in p-EFSM, `from` to edge state, `provided` to predicate, `delay` is a timing special input event, `to` to the tail state and `output` to output event.

4 Analysis and Testing Issues

Analysis and testing of embedded systems have to prove correctness, completeness and consistency in early phases of system development. Correctness means the fulfillment of the required services and its providing within a given time period. Completeness is its act of reaction to all possible events and carrying out all services. Consistency relates to the interior contradiction freeness of the specification.

There are two kinds of testing, general and special. The first one consists of testing of properties that must be held independently of special semantics of the developed system (consistency), such as livelock and deadlock-freeness, limitedness and resynchronization. The second aims at properties that are determined by the semantics of the designed system.

Properties that are addressed by analysis and testing are summarized as follows:

- *The non-existence of non-executable actions:* The system comprises no actions that cannot be executable under normal conditions.
- *Liveliness:* Each state of the system is reachable from the initial state.
- *Deadlock-freeness:* The system reaches no state that does not allow to interact with the environment and never leaves it.
- *Livelock-freeness:* The system comprises no non-productive cycles.
- *Error tolerance and resynchronization:* The system reaches a normal state within a limited time period after an error leading to an abnormal state has been occurred.

- *Safety:* The system comprises no unspecified events.
- *Partial correctness:* The system provides a special service when it terminates.
- *Termination:* The system reaches each time the final state(s), or the initial state for cyclic systems.

Precise specifications are essential to allow the analysis of embedded systems. The use of formal methods enables the automation of most aspects. We are particularly interested in the following testing problems:

- The non-executability of parts of the system
- Deadlock situations
- Inconsistencies of the system, i.e. whether the system contains non- deterministic behaviors
- Prohibited types of communication
- Incompleteness
- Checking of erroneous behaviors

The first five problems relate mainly to the system design phase, whereas the last question is more relevant for testing. These problems are usually very complex for models like extended finite state machines because of the inclusion of state variables and interaction parameters. Furthermore, the derivation of appropriate test cases is also an important issue for embedded systems testing and for software testing in general. In the context of embedded system testing, to decide whether a given part of the specification is executable is difficult. Therefore, the most work on verification and test development for embedded systems assumes that the system is specified in a simple state transition model without considering the data flow.

Detecting of Non-executable Parts

Lemma 4.1 *The detection of non-executable parts is reducible to the problem of deciding whether a given p-EFSM modeling a function for a given component contains non-executable transitions.*

The detection of non-executable transitions allows to deduce a specification whose all transitions are executable. This specification is obtained by eliminating all non-executable transitions as well as their descendants.

We can find all non-executable transitions in a p-EFSM as follows: A branch $s \rightarrow s'$ (i.e. a sequence of transitions starting at state s and ending at s') in the p-EFSM is non-executable if the two following conditions are fulfilled:

- $\exists x_1, \cdots, x_k[\delta_s(x_1, \cdots, x_k)]$
- $\neg(\exists x_1, \cdots, x_k[\delta'_s(x_1, \cdots, x_k)])$

  where

- s and s' are two given states of the p-EFSM.
- $\delta_s(x_1, \cdots, x_k)$ represents the intersection of the (for the given context $x_1, \cdots, x_k$ fulfilled) predicates of all transitions starting at the initial state s_0 and ending at state s.

The problem for deciding, whether a given transition (or a branch) of the p-EFSM is non-executable, is resolvable under certain assumptions. Indeed, if the domains of state variables are finite and countable it is always possible to solve the problem analytically or by simulation. For each state of the p-EFSM it is possible to assess whether there is a context for which the predicates are not satisfied.

Note 1 *An event e of the p-EFSM is represented as follows:* ***g.e****. The symbol* ***\$*** *denotes either the input symbol* ***?*** *or the output symbol* ***!****. The letter* ***g*** *represents the name of the component sending the event or receiving it. We use the letters* ***s****,* ***c*** *and* ***a*** *to designate sensors, the controller and actuators, respectively. The condition for a transition is written in brackets [], the output event follows the bracket by* ***:****. The whole sequence of input, condition and output is denoted as* ***?***$g_i.i, [c]$ $:!g_o.o$.

An example of an p-EFSM is given in Figure 4 that comprises three global variables x, y and z.

In Figure 4, we can show by means of the transition conditions for state s_3 and s_4:

- $\exists x, y, z[\delta_{s3} = true]$ and $\neg(x, y, z[\delta_{s4} = true])$ where
- $\delta_{s3}(x, y, z) \equiv [((x >= -8)$ and $(x <= 8))$ and $((y >= -8)$ and $(y <= 8))$ and $(y <= -1)$ and $(z < x - 1)]$, and
- $\delta_{s4}(x, y, z) \equiv [((x >= -8)$ and $(x <= 8))$ and $((y >= -8)$ and $(y <= 8))$ and $(y <= -1)$ and $(z < x - 1)$ and $(x = z)]$

In this example, we suppose that all the variable domains are of type integer, finite and countable. s_3 is reachable from the initial state, but the branch $s_3 \rightarrow s_4$ is non-executable.

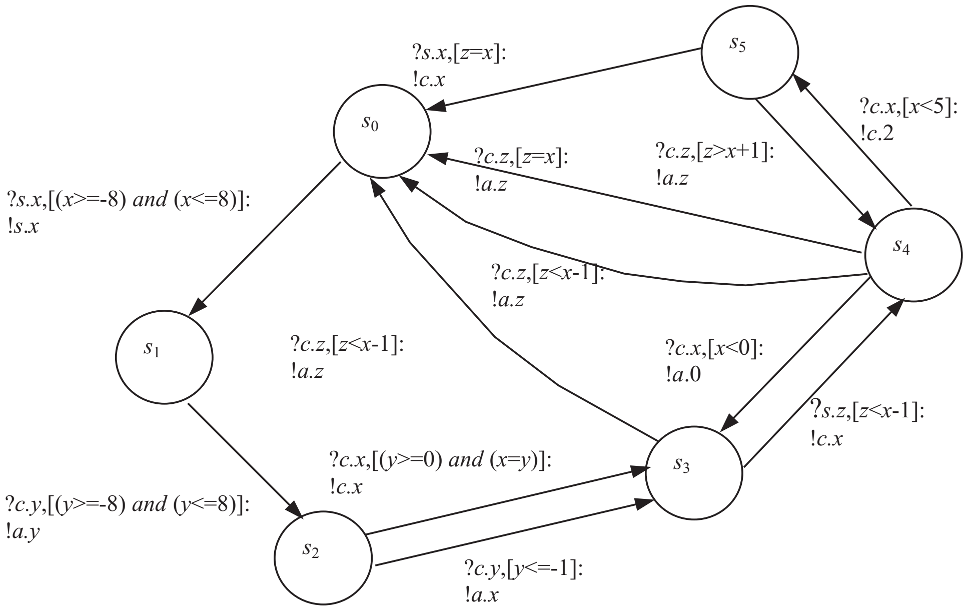

Fig. 4. A p-EFSM example for the controller

Detecting Deadlocks

Lemma 4.2 *Deadlock detection is reducible to decide whether a given p-EFSM modeling a function of a component contains deadlocks, and generating a specific test sequence that leads to the deadlock states.*

Let be given a p-EFSM, a state s, and $s'_1, s'_2, , s'_m$ all the tail states of s. We assume that for the following transitions: $\langle s, c_1, i_1, p_1, o_1, s'_1, c'_1 \rangle$, $\langle s, c_2, i_2, p_2, o_2, s'_2, c'_2 \rangle$, $\cdots$, $\langle s, c_m, i_m, p_m, o_m, s'_m, c'_m \rangle$ the predicates $p_1, p_2, \cdots, p_m$ are satisfied. The p-EFSM contains a deadlock state if there exists a state s in the p-EFSM such that the following condition is satisfied:

- $\exists x_1, x_2, \cdots, x_k[\delta_s(x_1, x_2, \cdots, x_k)$ and $\neg p_1$ and $\neg p_2 \cdots$ and $\neg p_m]$

p_i denote the enabling predicates for leaving the state s. If the condition is satisfied, then there is no executable input event at the state s, which means that the state s is a deadlock state.

In the p-EFSM of Figure 4 the state s_5, for instance, represents a deadlock state. The condition for checking the presence of deadlocks for this state is:

- $\exists x, y, z[\delta_{s5}(x, y, z)$ and $\neg p_5]$
  $\Leftrightarrow \exists x, y, z[((x >= -8)$ and $(x <= 8))$ and $((y >= -8)$ and $(y <= 8))$ and $(y <= -1)$ and $(z < x - 1)$ and $(x < 5)$ and $\neg(z > x + 1)$ and $\neg(z = x)]$

This condition is true, for instance, for the context $x = 2$, $y = 1$ and $z = 0$ noted $(x/1, y/1, z/0)$. Therefore, the state s_5 represents a deadlock state for the given context. The input sequence leading to the deadlock state and the corresponding generated output sequence are $\rho[s_0, s_5](x/2, y/1, z/0) =$ "$?s.2; ?c.1; ?s.0; ?c.2$" and $\beta[s_0, s_5](x/2, y/1, z/0) =$ "$!s.2; !a.1; !a.2; !c.2; !c.2$", respectively. $\rho[s_0, s](x_1/\lambda_1, \cdots, x_n/\lambda_n)$ and $\beta[s_0, s](x_1/\lambda_1, \cdots, x_n/\lambda_n)$ denote functions that provide an input sequence or an output sequence, respectively, for the context $(x_1/\lambda_1, \cdots, x_n/\lambda_n)$ from the initial state s_0 until the state s. The semi-colon ; is used as an event sequencing operator.

Lemma 4.3 *For a given state s in the p-EFSM, it is decidable whether the state s is a deadlock state. If s is a deadlock state, then a test sequence leading to the deadlock state can be derived.*

Detecting Non-Determinism

Lemma 4.4 *Non-determinism can be reduced to decide whether a p-EFSM describes non-determinism and to determine test sequences that lead to non-deterministic behaviors.*

Let s denote a state of a p-EFSM. Suppose that we have transitions $t = \langle s, c_1, i_1, p_1, o_1, s'_1, c'_1 \rangle$ and $t' = \langle s, c_2, i_2, p_2, o_2, s'_2, c'_2 \rangle$ with the same edge state s and satisfying conditions (i) and (ii):

(i) The predicate p_1 holds. The input event i_1 has the form $?g.e_1$.

(ii) The predicate p_2 holds. The input event i_2 has the form $?g.e_2$.

We also assume that the following condition

- $x_1, x_2, \cdots, x_k[\delta_s(x_1, \cdots, x_k)$ and p_1 and p_2 and $(e_1 = e_2)]$ (*)

is satisfied and the values $\lambda_1, \cdots, \lambda_k$ are the solutions for the variables $x_1, \cdots, x_k$ to meet predicates p_i. Since $\delta_s(x_1, \cdots, x_k)$ is true, the input event sequence $\rho[s_0, s](x_1/\lambda_1, \cdots, x_k/\lambda_k)$ traces the path from the initial state s_0 of the p-EFSM to the state s.

If the condition above holds, then

"$\rho[s_0, s](x_1/\lambda_1, \cdots, x_k/\lambda_k); ?g.e_1(x_1/\lambda_1, \cdots, x_k/\lambda_k)$"

is an event sequence to trace the path from the initial state s_0 to the state s'_1. Here, $?g.e_1(x_1/\lambda_1, \cdots, x_k/\lambda_k)$ is obtained by assigning the values $\lambda_1, \cdots, \lambda_k$ to variables $x_1, \cdots, x_k$. Since the values $\lambda_1, \cdots, \lambda_k$ also satisfy the condition p_2, "$\rho[s_0, s](x_1/\lambda_1, \cdots, x_k/\lambda_k); ?g.e_2(x_1/\lambda_1, \cdots, x_k/\lambda_k)$" is a test sequence to trace the path from the initial state s_0 to the state s'_2. The value of the expression $e_1(x_1/\lambda_1, \cdots, x_k/\lambda_k)$ is the same as that of $e_2(x_1/\lambda_1, \cdots, x_k/\lambda_k)$. This means that the p-EFSM contains non-determinisms and that the sequence

"$\rho[s_0, s](x_1/n_1, \cdots, x_k/n_k); !g.e_1(x_1/n_1, \cdots, x_k/n_k)$" is an event sequence representing non-deterministic behaviors. The state s is called the state starting non-determinism. Since it is decidable whether the condition **(*)** is satisfied, the following lemma holds.

Lemma 4.5 *For a given state s in the p-EFSM, it is decidable whether s is a state starting non-determinism. If s is such a state, then an event sequence representing non-determinism can be derived.*

For the p-EFSM in Figure 4, for instance, there are two paths from s_2 to s_3, i.e. transitions $t_3 = \langle s_2, c_3, i_3, p_3, o_3, s_2, c_3' \rangle$ and $t_4 = \langle s_2, c_4, i_4, p_4, o_4, s_3, c_4' \rangle$. We can show that the input event i_3 of t_3 is "$?c.x$" and the input event of t_4 is "$?c.y$", and that

- $\exists x, y[\delta_{s2}(x, y, z)$ and p_3 and p_4 and $(x = y)]$
  $\Leftrightarrow \exists x, y[((x >= -8)$ and $(x <= 8))$ and $((y >= -8)$ and $(y <= 8))$ and $((y >= 0)$ and $(y = x))$ and $(y <= -1)$ and $(x = y)]$

which can be satisfied for the context $(x/1, y/1)$. Therefore, we conclude that the p-EFSM of the example is non-deterministic.

Detecting Prohibited Communications

Lemma 4.6 *Prohibited communications is reducible to decide whether a p-EFSM describing a function of a given component contains prohibited events and to determine them if they exists.*

Let be given a $p-EFSM_s$, $p-EFSM_c$ and $p-EFSM_a$ that model functions of the components: *sensors*, *controller* and *actuators*, respectively. Specification relations between these components can functionally be expressed as follows:

- Let be given a transition $t_s \in T_s : ts = \langle s_s, c_s, i_s, p_s, o_s, s_s', c_s' \rangle$ with
  $s_s \in S_s, c_s \in C_s, i_s \in I_s, p_s \in P_s, o_s \in O_s, s_s' \in S_s, c' \in C_s$
  $\Rightarrow \exists t_c \in T_c : t_c = \langle s_c, c_c, i_c, p_c, o_c, s_c, c_c' \rangle | o_s \equiv i_c$
- Let be given a transition $t_c \in T_c : t_c = \langle s_c, c_c, i_c, p_c, o_c, s_c', c_c' \rangle$ with
  $s_c \in S_c, c_c \in C_c, i_c \in I_c, p_c \in P_c, o_c \in O_c, s_c' \in S_c, c_c' \in C_c$,
  if $i_c \in O_s \Rightarrow \exists t_s \in T_s : t_s = \langle s_s, c_s, i_s, p_s, o_s, s_s', c_s' \rangle$ and $i_c \equiv o_s$.
- Let be given a transition $t_a \in T_a : t_a = \langle s_a, c_a, i_a, p_a, o_a, s_a', c_a' \rangle$ with
  $s_a \in S_a, c_a \in C_a, i_a \in I_a, p_a \in P_a, o_a \in O_a, s_a' \in S_a, c_a' \in C_a$
  $\Rightarrow \exists t_c \in T_c : t_c = \langle s_c, c_c, i_c, p_c, o_c, s_c', c_c' \rangle$ and $o_c \equiv i_a$.
  For actuators, we assume that:
  $\forall t_a \in T_a : t_a = \langle s_a, c_a, i_a, p_a, o_a, s_a', c_a' \rangle \Rightarrow o_a \equiv \varepsilon$ (ε empty event)

Considering the facts above, we can conclude that an embedded system contains a prohibited communication if particularly one of the following conditions holds:

- $\exists t_s \in T_s : t_s = \langle s_s, c_s, i_s, p_s, o_s, s'_s, c'_s \rangle \mid$ $\forall t_c \in T_c : t_c = \langle s_c, c_c, i_c, p_c, o_c, s'_c, c'_c \rangle | o_s \equiv i_c$
- $\exists t_a \in T_a : t_a = \langle s_a, c_a, i_a, p_a, o_a, s'_a, c'_a \rangle \mid$ $\forall t_c \in T_c : t_c = \langle s_c, c_c, i_c, p_c, o_c, s'_c, c'_c \rangle | o_s \equiv i_c$
- $\exists t_a \in T_a : t_a = \langle s_a, c_a, i_a, p_a, o_a, s'_a, c'_a \rangle \mid o_a \equiv \varepsilon$

These predicates assume that first, it exists at least one input event generated by the sensors that is never used in the controller. Secondly, the actuator contains a transition whose input event is not specified in the controller. Thirdly, it exists at least a transition in actuators whose output event is not empty.

Lemma 4.7 *Let be given all the p-EFSMs of all components, it is decidable whether the whole system contains prohibited behavior as shown above.*

Detecting Incompleteness

Lemma: Incompleteness is reduced to decide whether a given p-EFSM of a component is incomplete and to determine the states presenting incompleteness if they exist.

Let be given a $p - EFSM_s$, $p - EFSM_c$ and $p - EFSM_a$ that denote p-EFSMs of *sensors*, *controller* and *actuators*, respectively. If one of the following conditions holds, the corresponding p-EFSM is *incomplete*:

- $(\exists s_s \in S_s, s_c \in S_c, s_a \in S_a)$ and $(\exists i_s \in I_s, i_c \in I_c, i_a \in I_a)$ such that $t_s = \langle s_s, c_s, i_s, p_s, o_s, s'_s, c'_s \rangle \notin T_s$, $t_c = \langle s_c, c_c, i_c, p_c, o_c, s'_c, c'_c \rangle \notin T_c$, $t_a = \langle s_a, c_a, i_a, p_a, o_a, s'_a, c'_a > \notin T_a)$

For a given p-EFSM, it is decidable whether this p-EFSM is incomplete. The states in which the p-EFSM presents incompleteness can be then detected. This can be carried out by checking all the states of the specified behavior of the given p-EFSM. Each time the specified input events have to be checked against the potential input events at the given state.

The p-EFSM given in Figure 4, for instance, is incomplete because the initial state s_0, for example, presents incompleteness.

5 Conclusion

In this work we have presented an analysis approach dedicated to reliable embedded systems which can be described as (predicated) extended finite state machines (p-EFSMs). The mapping of this formalism on formal description techniques like Estelle or SDL has been addressed. We identified the different analysis and testing issues based on p-EFSMs. The solutions of these problems depending on the ranges of the state variables and interaction parameters have been demonstrated. The main advantage of the approach is the use of already standardized specifications languages (FDTs). These are characterized not only by a formal syntax but also a formal semantic and have been successfully used in the formal design of many communication protocols and communicating systems [9]. In addition, many test generation methods (e.g. transition tour method, distinguishing sequences' method, characterising sequences' method, unique input/output sequences' method, W-method, etc. [6] [7) have been developed for state/transition models which build the background of these FDTs.

We are now developing a knowledge-based diagnosis system to explain the reasons of errors in faulty implementations after execution test suites (sets of test cases). In addition, we plan to investigate real-life embedded systems especially from the automotive area to study the extent of the application of the analysis and testing approach.

References

[1] Aho, A. V. *et al*, *An optimisation technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours*, In S. Aggarwal and K. Sabnani, editors, Protocol Specification, Testing, and Verification, New Jersey, 1988.

[2] Buessow, R., R. Geisler, and M. Klar, *Specifying safety-critical embedded systems with statecharts and Z: A case study*, In Proceedings of Fundamental Approaches to Software Engineering (FASE'98), Lisbon, 1998.

[3] Corts, L. A., P. Eles, and Z. Peng, *Verification of embedded systems using a petri net based representation*, In Proceedings of the 13th international symposium on System synthesis, Madrid, 2000.

[4] Fujiwara *et al*, *Test selection based on finite state models*, IEEE transaction on Software Engineering 17(6): 591-603, 1991.

[5] Henniger, O., A. Ulrich, and H. Koenig, *Transformation of Estelle modules aiming at test case derivation*, Chapmann & Hall, 1995.

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

[7] Mendler, M. , and G. Luettgen, *Statecharts: From Visual Syntax to Model-Theoretic Semantics*, In K. Bauknecht, W. Brauer, and Th. Mck (editors), Workshop on Integrating Diagrammatic and Formal Specification Techniques (IDFST 2001), pages 615-621, Vienna, 2001.

[8] Potter, B., J. Sinclair, and D. Till, *Introduction to Formal Specification and Z 2ndEdition*, Prentice Hall PTR, 1996.

[9] Richter, H. *et al*, *A Concept For a Reliable, Cost-Effective, Real-Time Local-Area Network for Automobiles*, In Proceedings of Joint conference Embedded in Munich and Embedded Systems, Munich, 2004.

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