

Model checking safety-critical systems specified as X- machines

George Eleftherakis and Petros Kefalas

Computer Science Department, City Liberal Studies,
Affiliated College of the University of Sheffield
13 Tsimiski Str., 54624 Thessaloniki, Greece
{ `eleftherakis, kefalas` }@ `city.academic.gr`

Abstract

Misleading user requirements, inappropriate specification and errors in the implementation of a system, are the usual reasons responsible for the creation of non-safe systems. The use of formal methods in the development of safety critical systems is demonstrated in the past. In this paper a formal technique is proposed in order to assist software engineers to find mistakes in the specification of the system, thus providing the ability to prove that certain desired properties exist in the final product. It is argued that the X-machine as a specification tool combined with the proposed model checking verification technique, gives the ability to the software engineer to formally and intuitively specify a system and then automatically check if this model has all the desired properties. Since complete testing of X-Machine specifications has been demonstrated elsewhere, integration of these techniques built around X-Machine theory leads towards an integrated framework for system development, ranging from validity of model properties to implementation correctness. The proposed ideas in this paper are illustrated through an example describing the specification of a medical safety-critical system.

1 Introduction

In safety critical domains, the traditional development life cycle failed several times to produce a reliable and correct system. The reasons for this are: i) Misleading user requirements may lead to a potential “correct” specification

and implementation, which however do not meet the actual requirements. ii) Inappropriate specification producing a model different from the one needed. iii) Errors in the implementation lead to a different product from the one specified.

The application of formal methods in safety critical systems could reveal errors during the development process, in both the system's specification and its implementation. *Formal specification* is the procedure of describing a system and its desired properties precisely, by using a language with rigorously defined syntax and semantics. Specifying a system means to create the appropriate *model*, that is a triple $\langle W, R, \pi \rangle$ where: i) W is a non-empty set of states, ii) $R \subseteq W \times W$, i.e. which states are related to other states iii) π is a truth assignment function, i.e. which propositions are true in each state, so $\pi : W \times P \rightarrow \{\text{true}, \text{false}\}$, where P is the set of all properties in this model. *Model checking* can be defined as the process of proving if a given property is valid in any, some or all states of the model. This can be achieved by searching the state space of the model (W, R), checking whether the property is valid through the π function. *Testing* has an essential role to play in system development. Testing may identify errors in the implementation without being able to prove its correctness. Testing based on formal specifications provides a structured and rigorous approach to the development of the test set [1].

In this paper, we propose the use of a formal method, namely X-Machines, which can accommodate all three above-mentioned activities. A X-machine is a general computational machine similar to a Finite State Machine (FSM) but with a significant difference; transitions are labelled with functions that operate on inputs and a memory, allowing the machine to be more expressive and flexible than the FSM. X-Machines are able to model both the control and the data part of a system. This integrated method (Figure 1) uses X-machines as a specification language, a testing strategy to check the implementation against the X-machine specification and a model checking technique to prove the validity of the specification. In safety critical systems such a formal methodology can be used to prove that several "safety" properties hold in the final product.

2 X-Machines as specification method

X-machine is a specification formalism introduced by Eilenberg [2], which is capable of modelling both the data and the control by integrating various specification methods. X-machines employ a diagrammatic approach

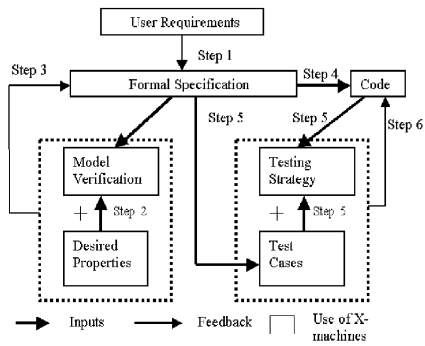


Figure 1: The methodology of building safety critical systems with X-Machines

of modelling the control by extending the expressive power of FSM. Transitions between states are no longer performed through simple input symbols but through the application of functions. Data is held in memory, which is attached to the X-machine. Functions receive input symbols and memory values, and produce output while modifying the memory values. Holcombe proposed X-machines as a basis for a possible specification language [3]. Stream X-machines are defined as X-machines with input and output sets of streams of symbols [4]. A *stream X-machine* is an 8-tuple $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ where:

- Σ, Γ is the input and output finite alphabet respectively,
- Q is the finite set of states,
- M is the (possibly) infinite set called memory,
- Φ is the type of the machine \mathcal{M} , a finite set of partial functions ϕ that map an input and a memory state to an output and a new memory state, $\phi : \Sigma \times M \rightarrow \Gamma \times M$
- F is the next state partial function that given a state and a function from the type Φ , denotes the next state. F is often described as a transition state diagram. $F : Q \times \Phi \rightarrow Q$
- q_0 and m_0 are the initial state and memory respectively.

X-machine is a general computational machine [4] that, being a blend of diagrams and simple formalisms, it is capable to model both the static and the dynamic part of a system. A simple example of a X-machine will be used as a vehicle of study: “ a medical x-ray beaming system is controlled using three buttons: i) one for charging the machine (a single button press increases the voltage by a 10 mV step), ii) one for the beam activation and iii) one for resetting the machine at any time. The system will only beam if the charge in mV has reached a preset maximum, e.g. 30mV. Any attempts to increase the charge of the machine should be rejected, since there is a danger to seriously injure the patient”.

The state transition diagram of the stream X-machine $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$

corresponding to that system is shown in figure 2. The X-machine’s input set is: $\Sigma = \{\text{charge_button}, \text{beam_button}, \text{reset_button}\}$

The output of the system consists of a set of messages that are displayed on a screen together with the current charge of the machine.

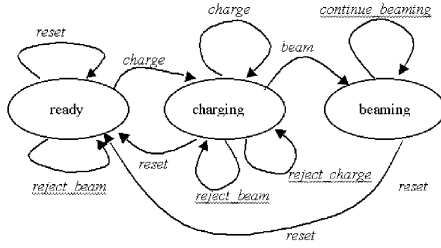


Figure 2: State transition diagram of the stream X-machine specification.

The set of states is: $Q = \{\text{ready}, \text{charging}, \text{beaming}\}$ The X-machine's memory is: $M = (\text{MaxCharge}, \text{CurrentCharge})$ where: i) MaxCharge is a variable holding the maximum accumulating voltage accepted by the machine and takes values from a set, e.g. $\text{MaxCharge} \in \{30\}$. ii) CurrentCharge is a variable holding the current voltage. Initially, the machine is in the state **ready**, i.e. $q_0 = \text{ready}$, while its initial memory is $m_0 = (30, 0)$ The next state function F is shown diagrammatically in figure 2. The functions are defined below. The notation used: $\phi(\sigma, m) = (\gamma, m')$ if condition, is very close to the X-Machine Description Language [5], which is intended to be an interchange language between X-Machine tools.

```

function charge(charge_button, (MaxCharge, CurrentCharge))=
  ((MachineCharging, CurrentCharge + 10),
  (MaxCharge, CurrentCharge + 10)),
  if CurrentCharge + 10 < MaxCharge.
function charge(charge_button, (MaxCharge, CurrentCharge))=
  ((MachineCharging, MaxCharge), (MaxCharge, MaxCharge)),
  if CurrentCharge + 10 ≥ MaxCharge ∧ MaxCharge ≠ CurrentCharge.
function reject_charge(charge_button, (MaxCharge, MaxCharge))=
  ((ChargeRejected, MaxCharge), (MaxCharge, MaxCharge)).
function reject_beam(beam_button, (MaxCharge, CurrentCharge))=
  ((BeamRejected, MaxCharge), (MaxCharge, CurrentCharge))
  if CurrentCharge < MaxCharge

```

```

function beam(button, (MaxCharge, MaxCharge))=
  ((MachineBeaming, 0), (MaxCharge, 0))
function reset(reset_button, (MaxCharge, CurrentCharge))=
  ((MachineResetting, 0), (MaxCharge, 0)).
function continue_beaming(button, (MaxCharge, 0))=
  ((ContinueBeaming, 0), (MaxCharge, 0))
  if button ∈ {beam_button, charge_button}

```

Ipate and Holcombe [6] presented a testing method, which is a generalisation of Chow’s W-method [7] for FSM testing. It is proved that this testing method finds all faults in an implementation [8], as long as the specification satisfies some design for test conditions (completeness and output distinguishability), and its associated automaton is minimal [4]. Concluding, X-machines not only provide a model to specify a system but also offer a strategy to test the implementation against the specification [9]. With the addition of a method to verify whether several desired properties hold in this specification or not, X-machines will provide a complete methodology to develop a safety critical system.

3 Model Checking X-Machine Specifications

Model Checking is a formal verification technique, which is based on the exhaustive exploration of a given state space trying to determine whether a given property is satisfied by the system. A model checker takes a model and a property as inputs and outputs either a claim that the property is true or a counterexample falsifying the property. The property is usually expressed as a Modal-Mu formula or as a Computational Tree Logic (CTL) formula. The most common properties to check are that “something” will never occur or “something” that will eventually occur. This approach is simple, completely automated but has one major problem, namely the state explosion [10].

In X-Machines, the search of some properties P of the model being true or false cannot be applied in a straightforward manner, since these properties are implicitly expressed in the X-Machine memory values. Thus, checking whether a property p_i is valid in some states of the X-Machine means whether there are some states in which some memory values satisfy the property p_i . For example, in order to verify that the current charge of the x-ray beaming machine will never exceed the maximum charge in any of the states, a model checker needs to search through all possible states as well as all possible instances of memory. Therefore, the appropriate model that facilitates model checking $\langle W, R, \pi \rangle$ should include: i) W is the set of

all possible states of the X-Machines combined with all possible instances of memory in each state, ii) R is the set of transitions between states in W , iii) π is the truth assignment function, i.e. given a member in W (a state with a specific memory instance) which properties are true depending on the values of this memory instance.

Bearing the above, model checking of a X-Machine model for specific properties can be achieved through the transformation of the X-Machine into the form $\langle W, R, \pi \rangle$. The resulting state space (W, R) resembles a FSM $(\Sigma, \Gamma, Q_{fsm}, T, q_{0fsm})$ where: i) Σ is a finite set that is called the input alphabet. ii) Γ is a finite set that is called the output alphabet. iii) Q_{fsm} is the finite set of states, $Q_{fsm} \subseteq W$. iv) T is the (partial) transition function, $T : Q_{fsm} \times \Sigma \rightarrow Q_{fsm} \times \Gamma$, (T is a labelled R). v) q_{0fsm} is an initial state, that encapsulates memory values which correspond to properties in each of its states.

The sets Γ and Σ are the same in both models. The initial state of the equivalent FSM q_{0fsm} is: $q_{0fsm} = (q_0, m_{01}, m_{02}, \dots, m_{0n})$, where q_0 is the initial state of the X-Machine and $m_0 = (m_{01}, m_{02}, \dots, m_{0n})$ is the initial memory. Let Q be the set of states of the X-machine and the memory tuple $M_1 \times M_2 \times \dots \times M_n$ with n memory variables, where M_i is the set of all the possible values. Then, let the set of states S be the candidate set of FSM states: $S = Q \times M_1 \times M_2 \times \dots \times M_n$. All candidate transitions of the equivalent FSM are defined as:

$$T1 = \{((q, m_1, m_2, \dots, m_n), \sigma, (q', m'_1, m'_2, \dots, m'_n), \gamma) \mid \\ \sigma \in \Sigma, \gamma \in \Gamma \wedge q, q' \in Q \wedge (m_1, m_2, \dots, m_n), (m'_1, m'_2, \dots, m'_n) \in \\ M \wedge \\ \exists \phi \in \Phi \cdot \phi(\sigma, (m_1, m_2, \dots, m_n)) = (\gamma, (m'_1, m'_2, \dots, m'_n)) \wedge \exists f \in \\ F \cdot f(q, \phi) = q'\}$$

Some of the states in these candidate transitions are unreachable from the initial state q_{0fsm} , i.e. there is no path from leading to those states. Therefore, the transitions, which should be excluded from the set of transitions, are:

$$T2 = \{(s_1, \sigma_1, s'_1, \gamma_1) \mid (s_1, \sigma_1, s'_1, \gamma_1) \in T1 \wedge (s, \sigma_2, s_1, \gamma_2) \notin T1 \wedge \\ \sigma_1, \sigma_2 \in \Sigma \wedge \gamma_1, \gamma_2 \in \Gamma \wedge s, s_1, s_2 \in S \wedge s_1 \neq q_{0fsm}\}$$

The set of transitions of the equivalent FSM becomes: $T = T1 - T2$. Finally, since the set S contains some redundant states, i.e. states without any transition or unreachable states from the initial state, the set Q_{fsm} becomes:

$$Q_{fsm} = \{s \mid s, s' \in S \wedge ((s, \sigma, s', \gamma) \in T \vee (s', \sigma, s, \gamma) \in T) \wedge \sigma \in \Sigma, \gamma \in \Gamma\}$$

In the example presented in the previous section, the equivalent FSM

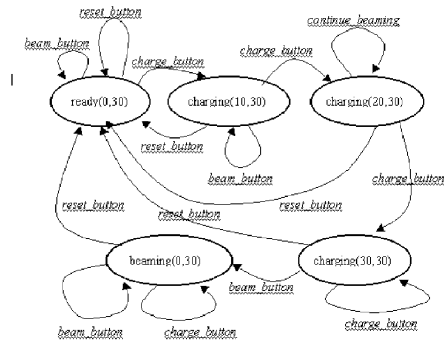


Figure 3: The equivalent FSM of the x-ray beamer X-Machine specification.

derived from the X-Machine specification is illustrated in Figure 3.

It is now possible to query the model if it has the desired properties by searching the state space defined by the equivalent FSM. In order to express such queries, there is a need for a mathematical language with built-in notion of time. Temporal logic seems to be the most appropriate. With the use of modal operators, i.e.: i) necessarily (\Box), ii) possibly (\Diamond) and iii) next (\circ), it is possible to express if a desired property is valid in the whole model or in part of it, starting from the initial state. In the example used previously in this paper the logic proposition: $\Box(\text{CurrentCharge} \leq 30)$, expresses that “it is impossible the voltage to become greater than the maximum value, which in this case is 30”.

A CTL formula contains: i) a boolean expression, ii) an existential (E) path formula, iii) a universal (A) path formula, or iv) the application of standard Boolean operators to CTL formulae. A path formula contains: i) the application of the temporal operators: next (X), eventually (F), or globally (G), to a CTL formula; or ii) the application of until (U) to a pair of CTL formulae. The operators A and E are called path quantifiers, and the operators X, F, G are called state quantifiers. There are four different cases and all are explained with the aid of the example in the following table:

Example of Property p	Temporal Operators	Explanation
$(\text{CurrentCharge} \leq 30)$	AGp	For every path and for every state in the path, the property p is be valid
$(\text{CurrentCharge} < 30)$	AFp	For every path, there exists at least one state where p is valid
$(\text{CurrentCharge} = 30)$	EGp	There are paths (at least one), where in every state p is valid
$(\text{MaxCharge} = \text{CurrentCharge})$	EFp	There are paths (at least one), where in some states p is valid

4 Implementation Issues

Efficiency in the model checking a X-Machine specification is still an issue under consideration due to the state explosion. Efficiency also depends on query, i.e.: i) the CTL operators involved, and ii) the number of properties involved.

For the first, there exist appropriate search algorithms, which can be applied in the model checking paradigm, according to the given query. For the latter, optimisation in deriving the equivalent FSM can be performed. If some of the elements in memory tuple do not correspond to a given property in the query, then the memory values of this element should not participate in the construction of the equivalent FSM, thus reducing the number of states in it. Let $P = \{p_a, p_b, \dots, p_m\}$ the set of properties in the CTL formula, and $M_1 \times M_2 \times \dots \times M_n$ the memory of the X-Machine. The properties may correspond to the memory elements i, j, \dots, k . Then, let the set of states S , which is the candidate set of FSM states can be reduced to $S = Q \times M_i \times M_j \times \dots \times M_k$.

The strictest conditions, which can be imposed on the implementation of a model checker are: i) the domain of every element in the memory tuple

is both finite and discrete, and ii) the input set is finite and discrete. There are, however, ways to relax those requirements. Infinite memory values and infinite input set refer either to: i) infinite values within a finite range, ii) discrete values within infinite range

It is possible to apply the proposed algorithm to X-machine models with infinite variables in the memory, with the restriction to check only properties relevant to the finite variables. In the case of infinity between a range, the values should be changed to discrete with an accepted step (something acceptable in computer systems). For discrete but infinite values an assumption should be made about the maximum value this variable could take.

5 Conclusions

We have presented a methodology for model checking X-Machine specifications. The methodology satisfies all the criteria of a technique that can verify the specification of a model, i.e. it can be fully automated, it can tackle in some cases the state explosion problem and it can act as the complementary part in the system development. The effectiveness of X-Machines theory in specification and testing has been already demonstrated elsewhere [6, 9]. Together with the proposed model checking method, X-Machines consists an integrated formal method, which can be used in all stages of safety critical system development.

References

- [1] Goodenough J.B., Gerhart S.L., "Toward a Theory of Test Data Selection", IEEE Trans. Software Eng., Vol. 1, No. 2, June 1975, pp.156-173.
- [2] Eilenberg S., Automata Machines and Languages, Vol. A, Academic Press, 1974.
- [3] Holcombe M., "X-machines as a basis for dynamic system specification", Software Engineering Journal, Vol.3, No.2, 1988, pp. 69-76.
- [4] Holcombe M. and Ipaté F., Correct Systems: Building a Business Process Solution, Springer Verlag, London, 1998.
- [5] Kefalas P. and Kapeti E., "A Design Language and Tool for X-Machines Specification", In Advances in Informatics, D.I. Fotiadis, S.D. Nikolopoulos (eds.), World Scientific Publishing Company, April 2000, pp. 134-145.

- [6] Ipate F. and Holcombe M., "Specification and testing using generalised machines: a presentation and a case study", *Software Testing, Verification and Reliability*, Vol.8, 1998, pp. 61-81.
- [7] Chow T.S., "Testing Software Design Modeled by Finite-State Machines," *IEEE Transactions on Software Engineering*, Vol.SE-4, No.3, 1978, pp.178-187.
- [8] Ipate F. and Holcombe M., "An integration testing method that is proved to find all faults", *International Journal of Computer Mathematics*, Vol.63, No.3, 1997, pp. 159-178.
- [9] Kehris E., Eleftherakis G., and Kefalas P., "Using X-Machines to Model and Test Discrete Event Simulation Programs", In *Systems and Control: Theory and Applications*, N. Mastorakis (ed.), World Scientific and Engineering Society Press, July 2000, pp. 163-168.
- [10] McMillan K.L., "Symbolic Model Checking," Kluwer Academic Publishers, 1993.

Received 24-th October 2000