Towards Model Checking of Finite State Machines Extended with Memory through Refinement

ELEFTHERAKIS G., KEFALAS P.
Computer Science Department
CITY LIBERAL STUDIES - Affiliated College of the University of Sheffield
13 Tsimiski st., 54624 Thessaloniki
GREECE
{eleftherakis, kefalas}@city.academic.gr http://www.city.academic.gr

Abstract: - Model checking is a successful verification technique for models expressed as finite state machines (FSM). Although FSM are able to describe the control part of a system, they lack the ability to efficiently describe data. As a consequence, FSM are not suitable to model several systems, which include complex data structures. This paper proposes a model checking technique for a type of FSM, namely the X-machine, which extends the FSM in two ways: (a) memory attached to each state, and (b) functions describing transitions between states. These features give the ability to the software engineer to intuitively as well as formally create a model of a system and then automatically check if this model has certain desired properties. The use of this formal method in the development of systems in safety critical domains can assure that the final product is valid with respect to the user requirements by revealing errors during the development life cycle. The proposed model checking technique is accompanied by an example, which demonstrates the use of X-machines in specification and verification.

Key-Words: - Formal Modelling, Specification, Verification

1 Introduction

The wide use of computers in many safety and business critical domains imposes a need for reliable hardware and software systems [1]. The application of formal methods in such systems could add to the confidence in using the system by revealing errors during the development process, in both the systems' modelling (due to verification) and implementation (due to testing).

Formal specification is the procedure of describing a system and its desired properties precisely, by using a language with rigorously defined syntax and semantics. Logic and sets often form the basic theory behind such mathematical languages, in order to avoid ambiguity and allow automated verification techniques to be used on the specification [2]. Specifying a system means to create the appropriate model that describes it, and therefore, in this context we will call this task *modelling*. The most usual general definition of a *model* is a labelled state transition graph, also called *Kripke structure* $<Q, R, L, q_0>$ [3] where:

- Q is a non-empty set of states,
- R is a binary relation on Q, i.e. $R \subseteq Q \times Q$, which shows which states are related to other states.
- $L: Q \rightarrow 2^P$ is a truth assignment function that shows which propositions are true in each state, where P is a set of atomic propositions
- q_0 is the initial state.

Bearing in mind the above definition, *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 (Q, R), checking in which states the property is valid by applying L. The process of stating the properties that the model should satisfy is called *specification*. In this context, the properties, which the model must satisfy are expressed in some formalism, usually *temporal logic*. Through temporal operators, temporal logic formulae [4] express how the behaviour of the system evolves over time. Using temporal logic, it is possible to write a specification as a formula. In order to prove that this specification is valid in the model, the temporal logic formula is checked against this model.

The expressive power of Finite State Machines (FSM) in modelling the control part of systems is demonstrated repeatedly over the last decades. Many model checking techniques have been devised for models expressed as a FSM. These proved to work efficiently in verifying state spaces with a huge

number of states [5]. On the other hand, it is obvious that FSM lack the expressive power for modelling the data part of a system.

In this paper, we present a technique that demonstrates the feasibility of model checking finite state machine models extended with memory. For this reason we propose the use of such a formal method, namely X-machine [6]. A X-machine is a general computational machine that is like a FSM but with a significant difference; transitions are not labelled with simple inputs but with functions that operate on inputs and a memory. The use of memory and functions allows the machine to be more expressive and flexible than a FSM because it is possible to model both the control and the data part of a system. We argue that, system modelling through X-machines provides a significant advantage, that is, the ability to prove that several "safety" properties hold in the final product. This important feature is due to: (a) the proposed model checking technique that proves the validity of the model against a specification expressed in temporal logic, and (b) a testing strategy to check the implementation against the X-machine model [7].

In section 2, X-machines will be formally defined and its advantages will be shortly presented. The main contribution of this work is analytically discussed in section 3, where a methodology for model checking X-machines is presented. A concrete example of a small safety critical system is given in order to accompany the theory and demonstrate the applicability of the approach.

2 Extending FSM with Memory: The X-machine formalism

There are several formal techniques, which may be used either to model the data or the control part of a system [2]. Introduced by Eilenberg [6] the X-machine is a general computational machine [8], that, being a blend of diagrams and simple formalisms, it is capable to model both the static and the dynamic part of a system. It also possesses highly expressive power by providing a convenient and intuitive way to model systems.

X-machines are equipped with a data structure, called the memory, which is accessible from each state. They also employ a diagrammatic approach of modelling the control by extending the expressive power of the FSM. Transitions between states are no longer performed through simple input symbols but through the application of functions. These functions are written in a formal notation and model the processing of the data held in the memory. Functions receive input symbols and memory values, and produce output while modifying the memory values.

Stream X-machines are defined as X-machines with input and output sets of streams of symbols. In short, the idea is that the machine has infinite internal memory and depending on the current state of control and the current state of the memory, an input symbol from the input stream determines the next state, the new memory state and the output symbol, which will be part of the output stream. The formal definition of a deterministic stream X-machine [7] is a 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 set called memory,
- Φ is the type of the machine M, a finite set of partial functions φ that maps a memory state and an input to a new memory state and an output,
$$\varphi: M \times \Sigma \rightarrow \Gamma \times M$$
- 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

In 1988, Holcombe proposed this model as a basis for a possible specification language [9]. In addition, the X-machine model facilitates the testing phase through a testing strategy, which is proved that produces a test-set capable of finding all faults in an implementation [8]. The testing process can be performed automatically by checking whether the output sequences produced by the implementation are identical with the ones expected from the specification. The effectiveness of this method is demonstrated elsewhere [10]. Therefore, X-machines not only provide a model to specify a system intuitively but also offer a strategy to test the implementation against the specification. With the addition of a method to verify whether several desired properties hold in this specification or not, X-machines will provide a formal approach in several important stages of the development life cycle of a critical system.

A simple example of a X-machine will be used as a vehicle of study: "A medical 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

Table 1: Formal Specification of the ray beamer as a X-machine model

The **X-machine** is defined as $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ where:	
Σ	$\Sigma = \{$charge_button, beam_button, reset_button$\}$
Γ	$\Gamma = \{$ MachineCharging, ChargeRejected, BeamRejected, MachineBeaming,ContinueBeaming, MachineReseting $\} \times N_0$
Q	$Q = \{$ ready, charging, beaming $\}$
M	M = (MaxCharge , CurrentCharge) Where MaxCharge is a variable holding the maximum accumulating voltage accepted by the machine, e.g. MaxCharge $\in \{30\}$, and CurrentCharge is a variable holding the current voltage.
q_0	q_0 = ready
m_0	$m_0 = (30, 0)$
F	F: $Q \times \Phi \rightarrow Q$ shown diagrammatically in figure 2.
Φ	The functions defined next using the notation: $\varphi(\sigma,m) = (\gamma,m')$ if condition **charge**(charge_button, (MaxCharge, CurrentCharge)) = ((MachineCharging, CurrentCharge+10), (MaxCharge, CurrentCharge+10)) if CurrentCharge+10 < MaxCharge **charge**(charge_button, (MaxCharge, CurrentCharge))= ((MachineCharging,MaxCharge),(MaxCharge,MaxCharge)) if CurrentCharge+10 ≥ MaxCharge ∧ MaxCharge ≠ CurrentCharge **reject_charge**(charge_button,(MaxCharge,MaxCharge))= ((ChargeRejected,MaxCharge),(MaxCharge,MaxCharge)) **reject_beam**(beam_button,(MaxCharge,CurrentCharge))= ((BeamRejected,MaxCharge),(MaxCharge,CurrentCharge)) if CurrentCharge < MaxCharge **beam**(beam_button, (MaxCharge, MaxCharge))=((MachineBeaming,0),(MaxCharge,0)) **reset**(reset_button,(MaxCharge,CurrentCharge))=((MachineReseting,0), (MaxCharge,0)) **continue_beaming**(button, (MaxCharge,0))= ((ContinueBeaming,0), (MaxCharge,0)) if button ∈ {beam_button, charge_button}

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 system described is formally specified starting with the visual part, which shows the different states of the machine and the transitions between these states in Figure 1.

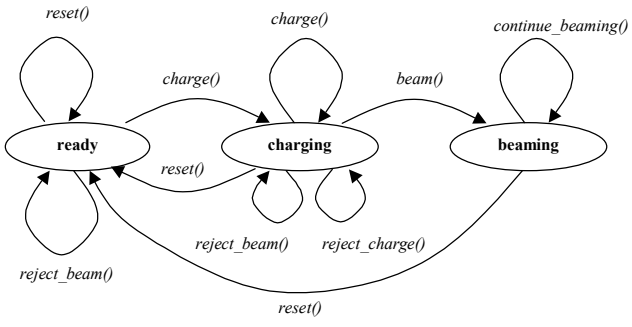

Fig.1: The state transition diagram of the X-machine specification

The formal model M corresponding to the system is formally presented in table 1.

3 Model Checking

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. In order to use model checking, the most efficient way to express a model is any kind of state machine, a CCS agent, a Petri Net, a CSP agent, etc. The most common properties to check are either something will never occur or something will eventually occur. In *Temporal Logic Model Checking* [11] a property is expressed as a formula in a certain temporal logic. The verification can be accomplished using an efficient breadth first search procedure, which views the transition system as a model for the logic and determines if the specifications are satisfied by that model. This approach is simple, completely automated but has one major problem, namely the state explosion. The latter can be handled to some extend by a model checking variant, the *Symbolic Model Checking* [12].

3.1 Model Checking X-machines

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 is valid in some states of the X-machine means whether there are some states in which some memory values satisfy this property. In essence, search should be performed through all instances of the memory of a X-machine model. For example, in order to verify that the current charge of the 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. The appropriate model $<Q,R,L,q_0>$ that facilitates model checking should include:

- Q is the set of all possible states of the X-machines combined with all possible instances of memory in each state,
- R is the set of transitions between states in Q,
- L is the truth assignment function, i.e. given a member in Q (a state with a specific memory instance) which properties are true depending on the values of this memory instance,
- q_0 is the initial state of the X-machine combined with the initial memory.

For example, some members of Q are: `charging (10,30)`, `charging (20,30)`, `charging (30,30)` etc., i.e. the ray beamer is at state charging, the current charge is 10mV and the maximum charge is 30mV, and so on respectively. Then, the property `CurrentCharge < MaxCharge` is true in `charging (10,30)` and `charging (20,30)` but false in `charging (30,30)`.

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 $<Q,R,L,q_0>$. This can be achieved through the process of *refinement*, which is the process of producing an equivalent X-machine with more states but less memory values. More generally, the state space (Q,R) that is produced through exhaustive refinement resembles a FSM $(\Sigma, \Gamma, Q_{fsm}, T, q_{0fsm})$ where:

- Σ is a finite set that is called the input alphabet.
- Γ is a finite set that is called the output alphabet.
- Q_{fsm} is the finite set of states, $Q_{fsm} \subseteq Q$.
- T is the (partial) transition function, $T: Q_{fsm} \times \Sigma \rightarrow Q_{fsm} \times \Gamma$, (T is a labelled R).
- q_{ofsm} is an initial state,

that encapsulates memory values which correspond to properties in each of its states.

The proposed model checking provides a way to prove whether a property expressed within a temporal logic formula is satisfied by the X-machine or not. The atomic propositions are assumed to depend only on memory variables. The methodology can be completely automated. The state explosion can be avoided through the selective refinement of the properties in question and is further discussed in section 3.4.

3.2 Exhaustive Refinement of a X-machine

The sets Γ and Σ are the same in both models. Below, we present a way to derive the set of states Q_{fsm} and the transition function T.

Let Q be the set of states of the X-machine and $M_1 \times M_2 \times \ldots \times M_n$ the memory tuple 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, which results through an exhaustive refinement of the X-machine:

$$S = Q \times M_1 \times M_2 \times \ldots \times M_n$$

The initial state of the equivalent FSM q_{0fsm} is:

$$q_{0fsm} = (q_0, m_{01}, m_{02}, \ldots, m_{0n})$$

where q_0 is the initial state of the X-machine and $m_0 = (m_{01}, m_{02}, \ldots, m_{0n})$ is the initial memory.

Practically, one could imagine that a state $s \in S$ is constructed as a compound name of the name of the states $q \in Q$ and a combination of all possible memory values M_i defined in the X-machine specification. All candidate transitions of the equivalent FSM are defined as:

$$\begin{aligned} T1 = \{&((q, m_1, m_2, \ldots, m_n), \sigma, (q', m_1', m_2', \ldots, m_n'), \gamma) \mid \\ &\sigma \in \Sigma, \gamma \in \Gamma \wedge q, q' \in Q \wedge \\ &(m_1, m_2, \ldots, m_n), (m_1', m_2', \ldots, m_n') \in M \wedge \\ &\exists \varphi \in \Phi \bullet \\ &\varphi(\sigma, (m_1, m_2, \ldots, m_n)) = (\gamma, (m_1', m_2', \ldots, m_n')) \wedge \\ &\exists f \in F \bullet f(q, \varphi) = q' \} \end{aligned}$$

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

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

From the above set, the initial state should be excluded. 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:

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

In the example presented, the equivalent FSM derived from exhaustive refinement of the X-machine model is illustrated in Figure 2.

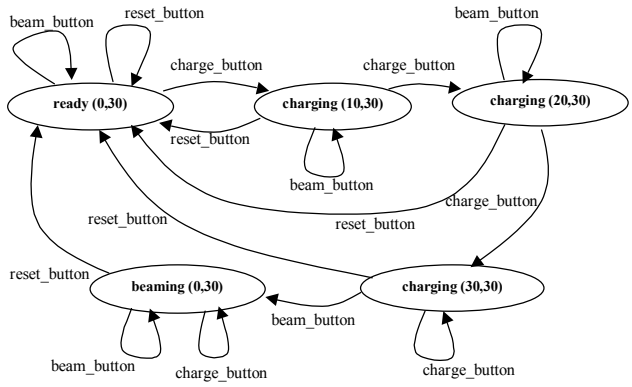

Fig.2: The equivalent FSM

3.3 Checking for Properties

It is now possible to query the model if it has the desired properties by searching the state space defined by the equivalent FSM. Temporal logic operators can be used to express such queries, i.e.:

- necessarily (□),
- possibly (◇) and
- next (○),

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: □(`CurrentCharge ≤ 30`), expresses that "it is impossible the voltage to become greater than the maximum value, which in this case is 30", meaning that the machine will never be charged with more than the permitted value, avoiding the chance to injure the patient. The translation of this temporal formula is that the requirement expressed with that formula holds in the model if in every state of the state space the following is true; the voltage is less or equal to 30 (the property p ⇔ `CurrentCharge ≤ 30`, is true in every state).

Since there is a need for more expressive and flexible operators, a more proper variation of temporal logic is the CTL [11]. A *CTL formula* contains:

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:

- the application of the temporal operators: next (X), eventually (F), or globally (G), to a CTL formula, or
- 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. In a quantified CTL formula the temporal operators always come in pairs of a path quantifier and a state quantifier. CTL formulae are interpreted with respect to an infinite computation tree derived from finite state transition machines. Each path in the tree is a sequence of states. So, for example, if p is a Boolean expression then there are four different cases of CTL formulae that are explained with the aid of the example in table 2.

Table 2. Examples of CTL formulae

Example of Property p	Temporal Operators in CTL	Explanation
CurrentCharge ≤ 30	AG p	For every path and for every state in the path, the property p is valid
CurrentCharge < 30	AF p	For every path, there exists at least one state where p is valid
CurrentCharge = 0	EG p	There are some paths (at least one), where in every state of these paths p is valid
CurrentCharge = 30	EF p	There are some paths (at least one), where in some states of these paths p is valid

3.4 Coping with State explosion with Selective Refinement

Efficiency in model checking a X-machine is still an issue under consideration due to the state explosion if exhaustive refinement is applied. Efficiency also depends on the query, i.e.:

- the CTL operators involved, and
- the number of properties involved.

For the first, there exist appropriate search algorithms, which can be applied in the model checking paradigm. For instance, in AGp, since an exhaustive search is required, Depth-First Search seems rather appropriate because of its memory efficiency in finite search spaces. In contrast, in EFp, Iterative Deepening can be perform better in large search spaces in order to find at least one state where the property holds.

For the latter, selective refinement can be performed. If the properties in the query refer to the whole memory tuple, then nothing can be done and the X-machine should be exhaustively refined into a FSM. If, however, 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 refinement of the X-

machine, thus reducing the number of states in it. Let P= {p_a, p_b, .., p_m} the set of properties in the CTL formula, and $M_1 \times M_2 \times \ldots \times M_n$ the memory of the X-machine. The properties may correspond to the memory elements i, j ,…, k. Then, let the set of states S, which is the candidate set of refined X-machine states can be reduced to $S = Q \times M_i \times M_j \times \ldots \times M_k$. The derivation of the equivalent refined X-machine is based on S, and therefore the resulting X-machine contains exactly the necessary states for model checking.

The actual refinement is possible under certain assumptions concerning infinite domains. The strictest conditions, which can be imposed on the implementation of a model checker are:

- the domain of every element in the memory tuple is both finite and discrete, and
- 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:

- infinite values within a finite range,
- discrete values within infinite range.

It is possible to apply the proposed refinement algorithm to X-machine models with infinite variables in the memory, with the restriction to check only properties relevant to the finite variables. This is feasible by "forgetting" all the infinite variables and using in the algorithm only the discrete and finite ones.

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.

4 Conclusion

We have presented a methodology for model checking X-machines. The methodology is based on selective refinement of X-machines and it is general enough to satisfy all the criteria of a technique that can verify a model that includes states and memory, i.e. it can be fully automated, and can tackle in some cases the state explosion problem. Together with the testing strategy proposed for X-machines elsewhere, it can contribute to safety critical system development. Future work will include the implementation of tools for model checking X-machines, which will be added to the existing tools already built around X-machines [13], thus providing an integrated framework for system development.

References:

[1] Leveson N.G., *Safeware: System Safety and Computers*, Addison Wesley Longman, 1995.

[2] Clarke E., Wing J. M., "Formal Methods: State of the Art and Future Directions", *ACM Computing Surveys*, Vol.28, No.4, December 1996, pp. 626-643.

[3] Hudges GE., Creswell MJ., *Introduction to Modal Logic*, Methuen, 1977.

[4] Pnueli A., "The temporal logic of programs", In Proc. of the 18th Annual Symp. on Foundations of Computer Science, IEEE Computer Society, New York, 1997, pp. 46–57.

[5] Burch J.R., Clarke E.M., McMillan K.L., Dill D.L. and Hwang J., "Symbolic model checking:10^{20} states and beyond", in Symp. on Logic in Computer Science, 1990

[6] Eilenberg S., *Automata Machines and Languages*, Vol. A, Academic Press, 1974.

[7] 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.

[8] Holcombe M. and Ipate F., *Correct Systems: Building a Business Process Solution*, Springer Verlag, London, 1998.

[9] Holcombe M., "X-machines as a basis for dynamic system specification", *Software Engineering Journal*, Vol.3, No.2, 1988, pp. 69-76.

[10] 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, 2000, pp. 163-168.

[11] Clarke E.M., Emerson E.A., and Sistla A.P., "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications", *ACM Trans. Programming Languages and Systems*, vol. 8, no. 2, Apr. 1986, pp. 244–263.

[12] McMillan K.L., *Symbolic Model Checking*, Kluwer Academic Publishers, 1993.

[13] 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.