Extending CTL to Facilitate Model Checking of X-machines

George Eleftherakis, Petros Kefalas, Anna Sotiriadou
Computer Science Department
City Liberal Studies
Affiliated College of the University of Sheffield
13 Tsimiski Str., 54624 Thessaloniki, Greece
{eleftherakis, kefalas, sotiriadou}@city.academic.gr

Abstract

Temporal logic has been demonstrated to be a powerful specification language suitable for describing properties of a system modelled as a finite state machine. Model Checking is a formal verification technique which determines whether a given property expressed as a temporal logic formula is satisfied by such a system model. Finite state machines lack the ability to model non-trivial data structures, in contrast to X-machines which can model both the data and the control part of a system. It would be desirable to apply model checking techniques to X-machines. However, with existing logic it is obscure how one can describe properties that refer to the memory data structure of a X-machine model. This paper describes the syntax and the semantics of a new logic, namely XmCTL, which extends CTL with memory quantifiers, thus facilitating model checking of X-machines.

1 Introduction

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 a 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 early 80s two different teams by Quielle and Sifakis [QuSi81] and by Clarke and Emerson [ClEm81], working in parallel, proposed temporal logic model checking algorithms. In *Temporal Logic Model Checking* [CES86] a property is expressed as a formula in a certain temporal logic, usually the *Computation Tree Logic* (CTL). In this context the most usual definition of a model is as a labelled state transition graph, also called *Kripke* structure $K = \langle Q, R, L \rangle$ [Kri63] where: i) Q is a non-empty set of states, ii) R is a binary relation on Q, i.e. $R \subseteq Q \times Q$, which shows which states are related to other states, iii) $L : Q \to 2^{Prop}$ is a truth assignment function that shows which propositions are true in each state, where $Prop$ is the set of atomic propositions. The semantics of CTL are defined with respect to a Kripke structure K.

Temporal Logic (TL) is an extension of logic through operators that handle the notion of time. Using these operators TL gives the ability to argue about "when" logical expressions are true. In CTL [CES86] each of the temporal operators must be preceded by either **A** (for all paths) or **E** (there exists path) path quantifiers. The five basic temporal operators of CTL are:

- **X** (next time) requires that a property holds at the following state,
- **F** (in the Future, eventually) a property will hold at some state on a path,
- **G** (Globally, always) a property holds at every state on a path,
- **U** (Until) combines two properties, and $p\mathbf{U}q$ holds in the model if there is a state in a path that the second property (q) holds and also the first property (p) holds in every preceding state on the path, thus p holds in a path until q holds,
- **R** (Release) also requires two properties and is the dual operator of **U**. $p\mathbf{R}q$ holds in the model if the second property (q) holds along a path up to and including the initial state where the first property (p) holds, however without requiring p to hold eventually.

CTL facilitates model checking in formal models that resemble a Kripke structure, like the Finite State Machines (FSM). However, CTL is not expressive enough to facilitate model checking of FSM extended with memory. In this paper, we argue that there is a need for appropriate extensions of CTL so that temporal logic formulae are able to describe specifications suitable to refer to the memory structure of the model. A X-machine is a general computational machine that is like a FSM extended with a memory structure which models the data of a system. Unlike FSM, in a X-machine transitions are not labeled with simple inputs but with functions that operate on inputs and memory, allowing the machine to be more expressive and flexible than the FSM and able to model both the control and the data part of a system [HoIp98].

2 X-machines

X-machine is a formal method introduced by Eilenberg [Eil74], which is capable of modelling both the data and the control of a system. X-machines employ a diagrammatic approach 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. In contrast to FSM, X-machines are capable of modelling non-trivial data structures by employing a 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 [Hol88]. Stream X-machines are defined as X-machines with input and output sets of streams of symbols [HoIp98].

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.

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 $p \in P$ is valid in some states of the X-machine means whether there are some states in which some memory values satisfy the property p. For example, in order to verify that a memory variable will never exceed a maximum value 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 $\langle Q, R, L\rangle$ should include: i) Q is the set of all possible states of the X-machine combined with all possible instances of memory in each state, ii) R is the set of transitions between states in Q, iii) L is the truth assignment function, i.e. given a member in Q shows 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 $\langle Q, R, L\rangle$. The resulting state space (Q, 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. 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. Thus with the use of CTL formulas to express a desired specification the above Kripke structure could be verified [ElKe01]. But although this process demonstrates the feasibility of model checking X-machine models, it creates two problems; i) the loss of expressiveness that the X-machine possesses, e.g., CTL is not expressive enough to describe that a properly p holds in some memory instances of all states of the X-machine, and ii) the combinatorial explosion.

3 XmCTL

Bearing in mind the above, in order to overcome the lack of expressiveness, there is a need for an appropriate formalism for expressing specifications of state transition systems extended with memory e.g. X-machines. The proposed logic (XmCTL) is an extension of CTL that will facilitate effective model checking of X-machines through the use of operators that quantify memory instances within a single state. In the following sections several preliminary definitions are given, while the syntax and the semantics of the proposed logic are presented.

3.1 Definitions

Definition 3.1 *$MProp$ is the set of all predicates composed of instances of memory variables and/or atoms.*

Definition 3.2 *A state q of the X-machine $\mathcal{M}$ is called x-state.*

Definition 3.3 *The set of all memory instances of all x-states q is denoted as Q_{fsm} and it is the set of states of the equivalent finite state machine derived from the exhaustive refinement of the X-machine $\mathcal{M}$. A memory instance of a x-state is denoted as qm ($qm \in Q_{fsm}$). The notation qm^i is also used to denote the i-th memory instance of the q x-state.*

Definition 3.4 *$\tau : Q_{fsm} \rightarrow 2^{MProp}$ is the truth assignment function that given a x-state memory instance qm^i, returns a set of all the propositions $p \in MProp$ that are true in the specific x-state q for the specific i-th memory instance.*

Definition 3.5 *A x-path π in a X-machine $\mathcal{M}$ is defined as an infinite sequence of states of $\mathcal{M}$, $\pi = q_0, q_1, \ldots$ such that for every $i \geq 0, \exists \phi \in \Phi : (q_i, \phi, q_{i+1}) \in F$ (an infinite branch in the corresponding to $\mathcal{M}$ computation tree). Also the notation π^i is used to denote the suffix of π starting at state q_i.*

3.2 Syntax

In this section the syntax of XmCTL formulas is defined. The temporal operators used in XmCTL are the operators of CTL described in section 1 with the addition of two new memory quantifiers; $\mathbf{M_x}$ and $\mathbf{m_x}$. $\mathbf{M_x}$ (for all memory instances) requires that a property holds at all possible memory instances of a x-state. $\mathbf{m_x}$ (there exists memory instance) requires that a property holds at some memory instances of a x-state. A class of x-state memory formulas, x-state formulas and x-path formulas are defined inductively:

XSM1. if $p \in MProp$, then p is a *x-state memory formula*,

XSM2. if a and b are x-state memory formulas, then $\neg a$, $a \vee b$, $a \wedge b$, are *x-state memory formulas*,

XS1. if a is x-state memory formula, then $\mathbf{M_x}a$ and $\mathbf{m_x}a$ are *x-state formulas*,

XS2. if f_1 and f_2 are x-state formulas, then $\neg f_1$, $f_1 \vee f_2$, $f_1 \wedge f_2$, are *x-state formulas*,

XP1. if f_1 and f_2 are x-state formulas, then $\mathbf{X}f_1$, $\mathbf{F}f_1$, $\mathbf{G}f_1$, $f_1\mathbf{U}f_2$, and $f_1\mathbf{R}f_2$ are *x-path formulas.*

XS3. if g is a x-path formula, then $\mathbf{A}g$, $\mathbf{E}g$ are *x-state formulas*,

A valid XmCTL formula is any x-state formula.

3.3 Semantics

In CTL, the semantics are defined with respect to a Kripke structure. Here, the semantics of XmCTL will be defined with respect to the X-machine model.

Definition 3.6 *The notation $\mathcal{M}, q \models f$, means that f (which is a x-state formula) holds at x-state q in the model $\mathcal{M}$. If p is a x-state memory formula, then the notation $\mathcal{M}, qm^i \models p$ means that p holds in the x-state memory instance qm^i in the model $\mathcal{M}$. In the case that g is a path formula, then the notation $\mathcal{M}, \pi \models g$ means that g holds along path π in the model $\mathcal{M}$. Usually if the X-machine structure $\mathcal{M}$ is clear from the context, it is omitted. Assuming that a, b are x-state memory formulas, f, f_1 and f_2 are x-state formulas, and g_1 and g_2 are x-path formulas, the relation $\models$ is defined inductively below:*

XSM1. $\mathcal{M}, qm^i \models p \Leftrightarrow p \in \tau(qm^i)$, where $p \in MProp$

XSM2. $\mathcal{M}, qm^i \models \neg a \Leftrightarrow \mathcal{M}, qm^i \not\models a$,
$\mathcal{M}, qm^i \models a \wedge b \Leftrightarrow \mathcal{M}, qm^i \models a$ and $\mathcal{M}, qm^i \models b$,
$\mathcal{M}, qm^i \models a \vee b \Leftrightarrow \mathcal{M}, qm^i \models a$ or $\mathcal{M}, qm^i \models b$,

XS1. $\mathcal{M}, q \models \mathbf{M_x} a \Leftrightarrow$ for all $i \geq 0$, $\mathcal{M}, qm^i \models a$,
$\mathcal{M}, q \models \mathbf{m_x} a \Leftrightarrow$ there exists $i \geq 0$, $\mathcal{M}, qm^i \models a$,

XS2. $\mathcal{M}, q \models \neg f \Leftrightarrow \mathcal{M}, q \not\models f$,
$\mathcal{M}, q \models f_1 \wedge f_2 \Leftrightarrow \mathcal{M}, q \models f_1$ and $\mathcal{M}, q \models f_2$,
$\mathcal{M}, q \models f_1 \vee f_2 \Leftrightarrow \mathcal{M}, q \models f_1$ or $\mathcal{M}, q \models f_2$,

XP1. $\mathcal{M}, \pi \models \mathbf{X} f \Leftrightarrow \mathcal{M}, first(\pi^1) \models f$,
$\mathcal{M}, \pi \models \mathbf{F} f \Leftrightarrow \mathcal{M}, first(\pi^i) \models f$, there exists $i \geq 0$,
$\mathcal{M}, \pi \models \mathbf{G} f \Leftrightarrow \mathcal{M}, first(\pi^i) \models f$, for all $i \geq 0$,
$\mathcal{M}, \pi \models f_1 \mathbf{U} f_2 \Leftrightarrow$ there exists $k \geq 0$: $\mathcal{M}, first(\pi^k) \models f_2$ and for all $0 < j < k$ $\mathcal{M}, first(\pi^j) \models f_1$,
$\mathcal{M}, \pi \models f_1 \mathbf{R} f_2 \Leftrightarrow$ for all $j \geq 0$, if for every $i \leq j$ $\mathcal{M}, first(\pi^i) \not\models f_1$ then $\mathcal{M}, first(\pi^j) \models f_2$,

XS3. $\mathcal{M}, q \models \mathbf{A} g \Leftrightarrow \mathcal{M}, \pi \models g$ for all x-paths π with $first(\pi) = q$,
$\mathcal{M}, q \models \mathbf{E} g \Leftrightarrow$ there exists x-path π with $first(\pi) = q$ such that $\mathcal{M}, \pi \models g$.

4 Discussion

X-machines are able to model both the control and the data part of a system and therefore it possesses valuable characteristics that are desirable to software engineering. A framework for formal development of systems proposed in [Ele01] uses X-machines as a formal modelling language [KEK00], a testing strategy to check the implementation against the X-machine model [IpHo97] and a verification technique to prove the validity of the model [ElKe00]. It is argued that, by applying the proposed framework based around the X-machine formal method, to safety critical systems it is possible to assure that several "safety" properties hold in the final product.

XmCTL is defined as an extension to CTL in order to improve expressiveness with respect to the specific model. However, efficiency in model checking a X-machine is still an issue under consideration due to the state explosion if exhaustive refinement is applied. Efficiency may be substantially improved depending on the number of properties involved. If the properties in the XmCTL formula refer to the whole memory tuple then the X-machine should be exhaustively refined into a FSM. This is inevitable, but it would have happened anyway if the system was modelled as a FSM from scratch. But if, for example, some of the elements in the memory tuple do not correspond to a given property in a XmCTL formula then the memory values of this element should not participate in the refinement of the X-machine. Thus, selective refinement can be performed [ElKe01] which will reduce the number of states in it and therefore the refined X-machine will contain exactly the necessary states for the model checking process.

References

[CES86] 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.

[ClEm81] Clarke E.M. and Emerson E.A., "Synthesis of synchronization skeletons for branching time temporal logic", In Proc. IBM Workshop on Logics of Programs, Lecture Notes in Computer Science, vol. 131, Springer-Verlag, 1981, pp.52-71.

[Eil74] Eilenberg S., Automata Machines and Languages, Vol. A, Academic Press, 1974.

[Ele01] Eleftherakis G., "A Formal Framework for Modelling and Validating Medical Systems", to appear in proc. MEDINFO 2001, Sep. 2001.

[ElKe00] Eleftherakis G, and Kefalas P., "Model Checking Safety-Critical Systems Specified as X-machines", *An. Univ. Bucur. Mat. Inform.*, Vol. 49, No.1, 2000, pp. 59-70.

[ElKe01] Eleftherakis G, and Kefalas P., "Towards Model Checking of Finite State Machines Extended with Memory through Refinement", to appear in proc. 5th WSES/IEEE WORLD MULTICONFERENCE ON Circuits, Systems, Communications & Computers (CSCC 2001), July 2001.

[EmHa86] Emerson E.A. and Halpern J.Y., "'Sometimes' and 'not never' revisited: On branching time versus linear time", *J. ACM*, 33:151-178.

[Hol88] Holcombe M., "X-machines as a basis for dynamic system specification", *Software Engineering Journal*, Vol.3, No.2, 1988, pp. 69-76.

[HoIp98] Holcombe M. and Ipate F., *Correct Systems: Building a Business Process Solution*, Springer Verlag, London, 1998.

[IpHo97] 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.

[KEK00] 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.

[Kri63] Kripke S.A.,"Semantical analysis of modal logic I: normal propositional calculi", *Z. Math. Logik Grund. Math.*, 9:67-96, 1963.

[QuSi81] Quielle J.P. and Sifakis J., "Specification and verification of concurrent systems in CESAR", In Proc. 5th Int. Symp. on Programming, pp. 337-350.