A Deductive Approach to Verify e-commerce Protocols

Mohamed Larbi Rebaiaia
Computer Sciences Department
University of Batna (05000), Algeria
Email: www.univ-batna.dz
Tel: +213-33-80-51-06

Abstract

We present an automatic and terminating approach for verifying e-commerce protocols. It is based on a powerful formal declarative programming language based on rewriting logic. The model checking is an endeavor to the axiomatization of computational tree logic operators and offer a new deductive ability to transform Kripke structure formula to normal form and thus to check easily its validity. A computation and deduction scheme will be introduced to prove the feasibility of rewriting logic to deal with verification systems. Experimental results are obtained from the application of the developed tool to an e-business procedure based on an IEC (International Electrotechnical Commission) standard 625-1 1979 interface protocol for programmable measuring instruments.

Keywords : Verification, Simulation, Rewriting Logic, Temporal Logic, e-commerce.

1. Introduction

The large demand for electronic access to information and electronic transactions in internet and World Wide Web have demonstrated that providers and consumers need a reasonable protection for their transactions. Thus, numerous protocols have been proposed that claim to guaranty the integrity of transaction between interveners. However, many of such protocol procedures fail to prevent transaction errors. Nowadays, specifications and verification of reactive systems such as digital hardware circuits are conveniently given in modal logics. A failure in the computer software system could have serious consequences and cause the loss of human lives and hundred's of millions (eg. Ariane rocket flight 501 and Intel's Pentium processor).

The system that is to be checked can be directly verified using three main categories which are formal techniques as the theorem provers such as ACL2 [1], Model-Checking as the symbolic model checker of McMillan (SMV) [2], and the well-known simulation techniques. Unhappily, simulation techniques are not capable to trace out all the behavior expectations and the proof systems (provers) are hard to be used by inexperienced practitioner.

During the last decades, a lot of techniques and theoretical formalisms have been proposed to abstract human thinking to resolve complex problems. Among such formal techniques, Rewriting Logic due to Meseguer et al. [3], constitutes a universal framework for unifying

different computational models as Turing computable numbers, Petri Nets [4], linear logic [5], and has been extended to real-time concepts of concurrent and hybrid system and so on. Maude system [3], developed by the team of Meseguer at the SRI Laboratory, which is similar to ELAN [6] and Cafe-Obj is essentially based on the semantic of the rewriting logic.

In this paper we give an overview of the rewriting logic and its application to obtain solution of critical systems based-constraints. Experimental results are obtained from the application of the developed tool to verify a part of the trade procedure of an e-business protocol based on an IEC (International Electrotechnical Commission) standard 625-1 1979 interface protocol for programmable measuring instruments [7]. We present two attempts to deal with verification based on rewriting logic. The first one is a deductive method modulo some axioms of CTL operators, and the second try to compute CTL fixpoints using a SAT procedure written in Maude. Both the two methods give a good results, but in case of complex systems they fail in a reasonable time.

The paper is organised as follows. In section 2 we introduce the theory of rewriting logic. Section 3 presents the Maude System and its language and section 4 deals with the verification of Specification using Rewriting Logic. Section 5, presents computational tree logic as a model of verification and we give all the theoretical aspects of model checking and the rewriting logic approach as a model of deductive computation. Finally, in section 5 we demonstrate the feasibility of our approach on a benchmark.

2. The Theory of Rewriting Logic

The specification language is based on Rewriting logic as developed by Meseguer et al. [3]. Rewriting logic is a unified and reflexive logic based on both rewriting theory and equational logic extended to membership equational logic which gives a rigorous expressiveness and a good integration of logics . A signature in this logic is a pair (Σ, E) with Σ a ranked alphabet of function symbols and E a set of Σ-axioms (equations). The sentences are rules of the forms $[t_A]_E \rightarrow [t_B]_E$ with t_A and t_B are Σ-terms and a theory is a rewrite system $S = (\Sigma, E, L, R)$, *where* L is a set of labels and R is a collection of labelled and possibly conditional rules. This generalises the usual notion of theory, which is typically defined as a pair consisting of a signature and a set of sentences. When specifying real-time systems, a time domain satisfying general axioms is needed. We then define a parameter called *time*. The time is considered as a particular action, acting on states of the system. The result of time action is considered as stimuli and is specified using a rewrite rule.

3. The Maude System and its Language

The Maude language specifies rewriting logic theories. Data types are defined algebraically by equations and the dynamic behavior of a system is defined by rewrite rules which describe how a part of the state can change in one step.
A Maude program (at the meta-level) defines different strategies guiding the execution of another Maude program. In this way Maude provides a flexible execution strategy and parameterized programming language where the user can define all the strategies he needs for analyzing his specification. The current version of the interpreter can perform up to 1.3 million rewrites per second, making Maude a very sophisticated tool which uses all kinds of techniques to improve performance and competitive with general- purpose programming languages in terms of efficiency.

The Maude system is a rewrite based interpreter for order-sorted logic [8]. The basis for the executable program is a theory. Theories can be defined using the notion of sorts, subsorts, variables, operators or operations and properties on these operators. The properties on the basic objects are axiomatized by equations and rewrite rules.
To better explain how Maude could perform a computation step, we present a classical example known as natural numbers in Peano form. The data structure of Peano form deal with the non-negative integers built up using the 0 and S function symbols. Every element of this structure can be represented by a variable-free (ground) term that evolves 0 and S only. Suppose now we define the addition operation + by the following axioms:

$$x + 0 = x \quad (1)$$
$$x + S(y) = S(x + y) \quad (2)$$

It is clear that adding two integers using the above two equations, yields a non negative integer. For instance, S(S(0)) + S(S(0)) equates to S(S(S(S(0)))) by deductive reasoning in term of the application twice the second axioms and once the first. Using such reasoning way, we can easily demonstrate by induction the property of associativity as follow :

$$(x + y) + z = x + (y + z) \Rightarrow (x + y) + S(z) = x + (y + S(z)). \quad (3)$$

Proof: We reason using the left and the right sides of the equation. We deduce from the left hand side (LHS),

$$\text{LHS : } (x + y) + S(z) \Rightarrow S((x + y) + z) \quad \text{--- by rule 2}$$
$$S((x + y) + z) \Rightarrow S(x + (y + z)) \quad \text{--- by induction hypothesis}$$

The right hand side (RHS) gives :

$$\text{RHS : } x + (y + S(z)) \Rightarrow x + S(y + z) \quad \text{--- by rule 2}$$
$$x + S(y + z) \Rightarrow S(x + (y + z)) \quad \text{--- by rule 2}$$

4. Verification using Rewri ting Logic

Maude language and its respective tool may be used for verifying assumption of critical systems. Maude is able to combine synchronous and asynchronous modelling of the rewriting logic and the robustness and the flexibility of object-oriented construction (for architecture and data design). Modeling critical systems one need to represent the relative behaviour using what we call a Boolean Automaton. Such representation is basically a kind of finite state machine encoded by a set of Boolean variables and equations. Boolean variables are used to represent program signals (lines of communication) and states. Each Boolean variable may be true or false, it depends of the presence of such signal. Input and output signals (variables) are joined to form Boolean equations. Thus, CTL (Computation Tree Logic) [9], a branching time temporal logic with future operation, can be used for specifying properties. Using the Symbolic Model Checker [2] and the relations in term of equations furnished by the Boolean Automaton, one can verify for example the Safety and the Fairness specification. In the sequel we introduce Propositionnal Calculus Logic presented as a module in Maude, which will be used by the system to obtain a reduction

form in term of normal form. Such presentation is followed by a description of the CTL logic and a symbolic model checking algorithm.

5. Computational Tree Logic as a Model of Verification

In recent years, there has been much effort to extend the wide range of theory and methods for verification of critical real time systems. The result of such intensive work was the ability to make difference between on modal-temporal of finite and infinite verification models and others like simulation, theorems proving compilers and equivalence checking. Temporal logic models called model checking are divided into two well known methods-- a linear temporal logic (PTL, LTL, PLTL) [10-11] introduced by Pnueli as a formal language used for modelling and checking critical systems. Such temporal logics are called linear, because the qualitative notion of time is linear, because at each moment of time there is only one possible successor state and thus only one possible future. The other kind of temporal logic introduced by Clarke & Emerson in [9-12] and independently by Quielle & Sifakis [11], takes into account non deterministic and branching time notions of concurrent systems. Such logic is called Computational tree logic (CTL), which was extended some time after to CTL* the union of both LTL and CTL, and TCTL (Timed CTL) to verify real time systems. The family of CTL logic is not based on a linear notion of time, but on a branching notion of time. Branching time refers to the fact that at each moment there may be several different possible futures. Temporal logics are interpreted using the satisfaction relation defined in terms of computation, noted $\kappa, s \models A$, which means that the temporal formula A is verified (holds) on a path (a sequence of states) of a model κ also called Kripke structure and takes s as the first state of the path. The semantics of CTL is based on tree of states rather than a sequence of states as in linear temporal logics.
The temporal operators in branching temporal logic allow the expression of properties of (some or all) computations of a system. For example the CTL operators EF ϕ (there exist a computation along which Fϕ holds), EG ϕ, AF ϕ and AG ϕ.

The most elementary statements about systems that one can make in CTL are atomic propositions, as in the definition of PLTL. The set of atomic propositions is denoted by AP, with typical elements p, q, and r.
A decision procedure for the propositionnal calculus provides the usual truth constants (true, false) and the following usual connectives (operators): $\wedge$ (and), $\vee$ (or), nor (exclusive or), $\Rightarrow$ (implication), $\Leftrightarrow$ (equivalence) and the negation operator $\neg$. The procedure reduce the tautology formulae to the constant true and all the others to the canonical form modulo associativity and commutativity.

Thus the logic of CTL consists of Propositional Logic augmented with path quantifiers and temporal operators are defined as follows:

- Path quantifiers - describe branching structure of the tree,
  - *A* (for *all* computation paths)
  - *E* (for *some* computation path = there *exists* a path)
- Temporal operators - describe properties of a path through the tree
  - *X* (next time, next state)
  - *F* (future state, eventually, finally)

Top

– *G* (always, globally)
– *U* (until)
– *R* (release, dual of U)

Definition 1. (Syntax of CTL) The syntax of CTL is made out of state formulae and path formulae (a path is a sequence of states connected via transitions) and is the smallest set of all state formulae as introduced by the following assumptions:

(i) p is a state-formula
(ii) If ϕ is a state-formula, then $\neg\phi$ is a state-formula
(iii) If ϕ and Ψ are state-formulae, then $\phi \vee \Psi$ is a state-formula
(iv) If ϕ is a path-formula, then E ϕ and A ϕ are state-formulae
(v) If ϕ is a state-formula, then X ϕ is a path-formula
(vi) If ϕ and ψ are state-formulae, then ϕ U ψ are state-formulae
(vii) Anything else is neither a state nor a path-formula

5.1 Semantic of Computational Tree Logic (CTL)

The semantics of CTL is defined over a Kripke structure *K*

Definition 2. (Kripke structure) A Kripke structure κ is a tuple *(S,T,S₀, AP, L)* where

- *S* is a countable set of states,
- $T \subseteq S \times S$ is the transition relation,
- $S_0 \subseteq S$ is the set of initial states,
- *AP* is the set of atomic propositions,
- $L : S \rightarrow 2^{AP}$ is the labelling function

A formula is said to hold in κ if it is satisfied by every initial state of κ.

Definition 3. (Path) A path in Kripke structure κ*(S,T,S₀, AP, L)* is an infinite sequence of states $s_0\ s_1\ s_2 \ldots$ such that $(s_i, s_{i+1}) \in T$ for all $i \geq 0$.

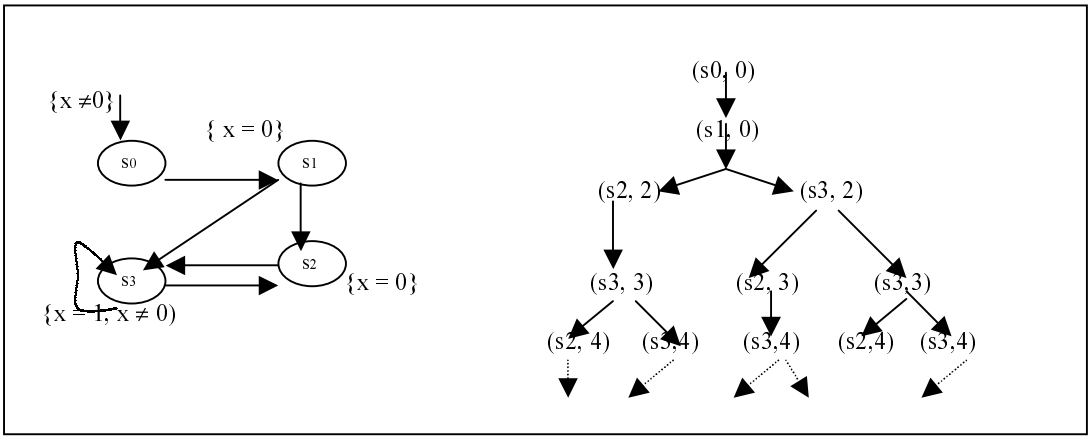

Figure 1 : A Kripke structure and its infinite tree

Top

Using the CTL operators EX, EG and EU, it is possible to find equivalent forms of CTL formulae.

(i) $AX\ \varphi = \neg\ EX(\neg\ \varphi)$
(ii) $EF\ \varphi = E(True\ U\ \varphi)$
(iii) $AG\ \varphi = \neg\ EF(\neg\ \varphi)$
(iv) $AF\varphi = \neg\ EG(\neg\ \varphi)$
(v) $A[\varphi\ U\ \psi] = \neg\ E\ [\neg\ \psi U(\neg\varphi \wedge \neg\ \psi)]\ \wedge \neg EG(\neg\ \psi)$
(vi) $A[\varphi\ R\ \psi] = \neg\ E\ [\neg\ \varphi\ U\ \neg\ \psi]$
(vii) $E[\varphi\ R\ \psi] = \neg\ A\ [\neg\ \varphi\ U\ \neg\psi]$

Example of such formulae equivalence, the *safety condition,*

- $AG(\neg\ (G1 \wedge G2)) = \neg\ EF(G1 \wedge G2)$

of the mutual exclusion (protocol), which signify that no both processes at the same time be in the critical section, and the *Fairness Condition,*

- $EF\ (G1 \vee G2\ ) = E(true\ U\ (G1 \vee G2\ )\ )$

which means, that eventually one process enter the critical section.

Definition 4. (Semantics of CTL)
Let $p \in AP$ be an atomic proposition, κ a Kripke structure, a state $s \in S$, Φ and Ψ a state formulae of CTL, and φ a path-formula of CTL. The satisfaction relation $\models$ is defined for the state-formulae as follows :

- $s \models p$ iff $p \in L(s)$.
- $s \models \neg\ \Phi$ iff not $(s \models \Phi)$.
- $s \models \Phi \vee \Psi$ iff $(\sigma \models \Phi)$ or $(\sigma \models \Psi)$.
- $s \models E\ \varphi$ iff $\sigma \models \varphi$ for some paths of κ : $\sigma \in Paths(s)$.
- $s \models A\ \varphi$ iff $\sigma \models \varphi$ for all paths of κ : $\sigma \in Paths(s)$.

While, the path-formula relation is defined as follow :

- $\sigma \models X\ \Phi$ iff $\sigma\ [1] \models \Phi$.
- $\sigma \models \Phi\ U\ \Psi$ iff $(\exists j \geq 0.\ (\sigma[j] \models \psi \wedge (\forall 0 \leq k < j.\ \sigma[k] \models \Phi))$.

The interpretation of atomic propositions as the negation and the conjunction operators are as usual.

Example of a deductive way to verify a specification in Computational Tree Logic.

Derivation of : EG Φ,

$s \models EG\ \Phi \Leftrightarrow$
{definition of EG}
$s \models \neg AF\ \neg\Phi \Leftrightarrow$

Top

{definition of F}
$s \models \neg A(true\ U \neg\Phi)) \Leftrightarrow$
{semantic of $\neg$}
$\neg(s \models A(true\ U \neg\Phi)) \Leftrightarrow$
{semantic of AU }
$\neg(\forall\sigma \in Paths(s).\ (\exists j \geq 0.\ (\sigma[j] \models \neg\Phi \wedge\ (\forall 0 \leq k < j.\ \sigma[k] \models true)\ )) \Leftrightarrow$
{$s \models$ true for all state s; predicate calculus}
$\exists\sigma \in Paths(s).(\forall j \geq 0.\ \sigma[j] \models \Phi\ )$

In CTL model checking, there are two approach's, one based on the axiomatisation of the model formulae and the second is based on the fixpoints calculus. In the sequel of this presentation the two methods will be developed and proved to be well-suited for the rewriting logic.

5.1 Formulae Axiomatization version rewriting rules

An interesting research direction consists of the application of rewriting logic in combining a rewrite system R with a set of simplification rules valid in Kripke structure.
We have seen that rather than a reasoning on the basis of the semantics, using the syntactic form of CTL formulae, axioms can be deduced from atomic formulae and used to prove the equivalence of CTL formulas. An important axiom for LTL is the expansion axiom for until:

$$\Phi\ U\ \psi \Leftrightarrow \psi \vee (\Phi \wedge X(\Phi\ U\ \psi))$$

For CTL, axioms similar to the expansion axiom exist. We note that the linear temporal operator U can be prefixed with either an existential or a universal path quantifier, we have axioms for E (Φ U ψ) and A (Φ U ψ). These axioms are listed in the last two rows of Table 1. The soundness of these axioms can be proved using the semantics of CTL.

Table 1. Expansion axioms for CTL

$EF\ \Phi \equiv\ \Phi \vee\ EX(EF\ \Phi)$	(axiom 1)
$AF\ \Phi \equiv\ \Phi \vee\ AX(AF\ \Phi)$	(axiom 2)
$EG\ \Phi \equiv\ \Phi \wedge\ EX(EG\ \Phi)$	(axiom 3)
$AG\ \Phi \equiv\ \Phi \wedge\ AX(AG\ \Phi)$	(axiom 4)
$E(\Phi\ U\ \psi) \equiv\ \psi \vee (\Phi \wedge EX\ E(\Phi\ U\ \psi))$	(axiom 5)
$A(\Phi\ U\ \psi) \equiv\ \psi \vee (\Phi \wedge AX\ A(\Phi\ U\ \psi))$	(axiom 6)

In the following proof we can show that AF Φ and $\Phi \vee\ AX(AF\ \Phi)$ are equivalent, and so it is for EG Φ and $\Phi \wedge\ EX(EG\ \Phi)$, and thus they could be substituted to each other.

Proof of : {$AF\ \Phi \equiv\ \Phi \vee\ AX(AF\ \Phi)$}

AF $\Phi \Leftrightarrow$
{definition of AF}
$A(true\ U\ \Phi) \Leftrightarrow$
{axiom for $A(\Phi\ U\ \psi)$}
$\Phi \vee (true \wedge AX\ A(true\ U\ \Phi))) \Leftrightarrow$

Top

{predicate calculus, definition of AF}
$\Phi \vee \mathbf{AX(AF\ \Phi)}$.

Proof of : $EG\ \Phi \equiv \Phi \wedge EX(EG\ \Phi)$.

EG Φ $\Leftrightarrow$
{definition of EG}
$\neg AF \neg\Phi \Leftrightarrow$
{results after the derivation}
$\neg (\neg\Phi \vee AX(AF \neg\Phi)) \Leftrightarrow$
{predicate calculus}
$\Phi \wedge AX((AF \neg\Phi)) \Leftrightarrow$
{definition of AX}
$\Phi \wedge EX(\neg(AF \neg\Phi)) \Leftrightarrow$
{definition of EG}
$\Phi \wedge$ **EX((EG Φ)**.

The following simplification rules could be used to reduce the number of CTL temporal operators and thus to find the normal form of a formula. We distinguish in table 2, two sets of rules. The first set consists of rules built over temporal and Boolean operators, and the second set consists of only temporal rules. The system of such rules decrease the depth of releases, the number of operators, and thus they (the rules) could be verified to be terminating, but not confluent.

Table 2. A rewriting CTL axioms

(i) $AG\ \Phi \wedge AG\ \psi \rightarrow AG(\Phi \wedge \psi)$
(ii) $EF\ \Phi \vee EF\ \psi \rightarrow EF(\Phi \vee \psi)$
(iii) $EG((\psi \vee EG\Phi) \vee (\Phi \vee EG\psi)) \rightarrow EG\Phi \vee EG\psi$
(iv) $AF((\psi \wedge AF\Phi) \vee (\Phi \wedge AF\psi)) \rightarrow EG\Phi \wedge EG\psi$
(v) $A((\Phi \wedge \Phi') U (A(\Phi U\psi) \wedge \psi' \vee A(\Phi' U\psi') \wedge \psi)) \rightarrow A(\Phi U \psi) \wedge A(\Phi' U\psi')$
$AGAG\ \Phi \rightarrow AG\ \Phi$
$EFEF\ \Phi \rightarrow EF\ \Phi$
$AGEFAGEF\ \Phi \rightarrow EFAGEF\ \Phi$
$EFAGEFAG\ \Phi \rightarrow AGEFAG\ \Phi$
$EGEG\ \Phi \rightarrow EG\ \Phi$
$AFAF\ \Phi \rightarrow AF\ \Phi$
$AFEGAF\ \Phi \rightarrow EGAF\ \Phi$
$EGAFEG\ \Phi \rightarrow AFEG\ \Phi$
$AGEF\ \Phi \rightarrow AG\ \Phi$
$AFAF\ \Phi \rightarrow AF\ \Phi$
$AFAGAF\ \Phi \rightarrow AGAF\ \Phi$
$EGEFEG\ \Phi \rightarrow EFEG\ \Phi$
$AGAFAG\ \Phi \rightarrow AFAG\ \Phi$
$EFEGEF\ \Phi \rightarrow EGEF\ \Phi$
$A(\Phi UA(\Phi U\psi)) \rightarrow A(\Phi U\psi)$
$AFEGAGAF\ \Phi \rightarrow EGAGAF\ \Phi$
$EGAFEFEG\ \Phi \rightarrow AFEFEG\ \Phi$
$AGAFEFAGEF\ \Phi \rightarrow AFEFAGEF\ \Phi$
$EFEGAGEFAG\ \Phi \rightarrow EGAGEFAG\ \Phi$

Top

5.2 Fixpoints model checking

The CTL model-checking is a procedure based on the theorem of Clarke and Emerson [12], which is defined as : given a model (Kripke structure) and a CTL formula P, the algorithm will return a set of those states of the model that satisfy P. Temporal logic formulas are evaluated to a state in the model as a b*asic* CTL formulae :

EX (f)	true in state *s* if *f* is true in some successors of *s* (there *exists* a next state of s for which *f* holds)
AX (f)	true in state *s* if *f* is true for all successors of *s* (for *all* next states of *s f* is true)
EG (f)	true in *s* if *f* holds in *every* state along *some* path emanating from *s (there exists a path …)*
AG (f)	true in *s* if *f* holds in every state along *all* paths emanating from *s (for all* paths, *globally*)
EF (g)	there *exists* a path which *eventually* contains a state in which *g* is true
AF (g)	for *all* paths, eventually there is state in which *g* holds
f U g	true if there is a state in the path where *g* holds, and at every previous state *f* holds

Typical of CTL formulae (for example for a sender and receiver of ABP protocol) are:

EF (*start* $\wedge$ $\neg$ *ready*) : eventually a state is reached where *start* holds and *ready* does not hold

AG (*req* $\wedge$ AF *ack*) : any time *request* occurs, it will be eventually *ac*knowledged

We remark that EGφ and E[φ U ψ] are computed respectively as the greatest fixpoint using the recursive function $\tau(Z) = \varphi \wedge EX\ Z$ and the least fixpoint function $\tau(Z) = \varphi \vee (\psi \wedge EX\ Z)$. The fixpoints are determined iteratively by approximations as defined by Theorem 1 and Theorem 2. A computation in such case lead to seek the states that satisfy the required property and reachable in one more step.
The following theorems (1 and 2), introduce the fixpoint approximation theory.

Theorem 1. (Tarski & Knaster, 1955)

- *If f is a monotonic function, then it has a least fixpoint,* $\mu Z.[f(Z)] = \cap \{Z \mid f(Z) = Z\}$, *and a greatest fixpoint,* $\upsilon Z.[f(Z)] = \cup \{Z \mid f(Z) = Z\}$.
- *If f is monotonic, f has the least (greatest) fixpoint which is the intersection (union) of all the fixpoints.*

Theorem 2. (Tarski & Knaster, 1955)

- *If f is* $\cup$*- continuous ,* $\mu Z.[f(Z)] = \cup_i f^i(\varnothing)$, $i = 1; \infty$ *and*
- *if f is* $\cap$ *- continuous ,* $\upsilon Z.[f(Z)] = \cap_i f^i(S)$, $i = 1; \infty$.

Each fixpoint can be computed as the limit of a series of approximations. The following theorem due to Clarke & Emerson determines CTL operators fixpoints.

Top

Theorem 3. (Clarke and Emerson, 1981) [12].

$$
\begin{array}{llll}
AFp & = & p \vee AX\,AFp & = & \mu Z.[\,p \vee AX\,Z] \\
EFp & = & p \vee EX\,EFp & = & = \mu Z.[\,p \vee EX\,Z] \\
AGp & = & p \wedge AX\,AGp & = & \upsilon Z.[\,p \wedge AX\,Z] \\
EGp & = & p \wedge EX\,EGp & = & \upsilon Z.[\,p \wedge EX\,Z] \\
A(\,pUq) & = & q \vee (p \wedge AX\,A(\,pUq)) & = & \mu Z.[\,q \vee (p \wedge AX\,Z)] \\
E(\,pUq) & = & q \vee (p \wedge EX\,E(\,pUq)) & = & \mu Z.[\,q \vee (p \wedge EX\,Z)]
\end{array}
$$

Model checking algorithms in the simple form proceed as follows:

Given a model κ and a temporal logic formula φ, find a set of states that satisfy φ $\{s \in S: M,s \models \varphi\}$ is performed using the following algorithms :

Algorithm 1.

Step 1: label each state s with the set *label(s)* of sub-formulas of φ which are true in s.
Step 2: $i = 0$; label(s) = L(s)
Step 3: $i = i + 1$; Process formulas with *(i -1)* nested CTL operators
Step 4: Add the processed formulae to the labelling of each state in which it is true
Step 5: Continue until closure. Result: $M,s \models \varphi$ iff $\varphi \in$ *label*

Algorithm 2

• ***eval*** takes a CTL formula as its argument and returns the rewriting form after transformation for the set of states that satisfy the formula,

• function *eval(f)*
  case
    f an atomic proposition*:* return *f;*
    $f = \neg p$: return $\neg$ *eval(p);*
    $f = p \vee q$: return *eval(p)* $\vee$ *eval(q);*
    f = EXp : return *evalEX*(eval(p));
    f = E(pUq) : return *evalEU(eval(p), eval(q), False);*
    f = EGp : return *evalEG(eval(p), True)*
  end case
 end function

• function *evalEX(p)* = $\exists v'(R \wedge p')$

• function *evalEU(p, q, y)*
    y' = $q \vee (p \wedge$ *evalEX(y))* ;
    if y' = *y* then return *y;*
    else return *evalEU(p, q, y');*
  end function;

Top

6. Experiment results

A contractual scenario is an e-business procedure based on an IEC (International Electrotechnical Commission) standard 625-1 1979 interface protocol for programmable measuring instruments. Such procedure which is a protocol, includes a 3-wire handshake intended to synchronise the transmission of data bytes over a bus to allow a single source to synchronise with a number of acceptors [7]. The trade procedure operates as follows: the seller can indicate that he has goods available for delivery (GAV). The purchasers can indicate that they are ready to take goods—that is, they request goods—(RFG) and can also indicate when they have received goods (GAC). The required sequence of events for the transfer of goods from the seller to the purchasers is as follows: GAV, RFG, GAC are initially set to false (to say that the seller is not ready for transaction, because for whatever reason some problems prevent him, for example because he has no goods available or because for some other private reasons), and remain false until all purchasers are ready to receive goods, at which time RFG becomes true. The seller may then assert GAV to true and RFG become false. When all purchasers have accepted a specified quantity of goods, GAC becomes true and only then, GAV may become false. Finally the purchaser's set GAC back to false and the cycle may repeat. To illustrate the verification of the trading protocol, one must to describe the behaviour using states diagram. First of all, the requirement items identified are transformed using specification parameters considered as Boolean variables. We note that such problem has been approached in [15] using a composition technique based on Integer Programming and Genetic Algorithms.
Consider the following trading protocol specification. A request can be made for goods, and the goods can be sent. The following automaton graph (Kripke structure) models the trading communication for setting the transactions. The states of the Kripke structure model are explained as follows:

- The initial state PNR (Purchaser Not Ready), labelled by the predicate RFG_F $\wedge$ GAC_F,
- The second state PR (Purchaser Ready), labelled by the predicate RFG_T $\wedge$ GAC_F,
- The third state PS (Purchaser Start transactions), labelled by the predicate RFG_T $\wedge$ GAC_F,
- The last state PF (Purchaser Finish transactions) labelled by the predicate RFG_F $\wedge$ GAC_T.

For simplicity we suppose that the prepositional variable used to model the communication relations between purchaser's and seller's-- RFG_F, RFG_T, GAC_F and GAC_T are replaced respectively by ~Req, Req, ~Sent and Sent. Following the definition of Kripke structure model, we obtain the graph sketched in figure 3.

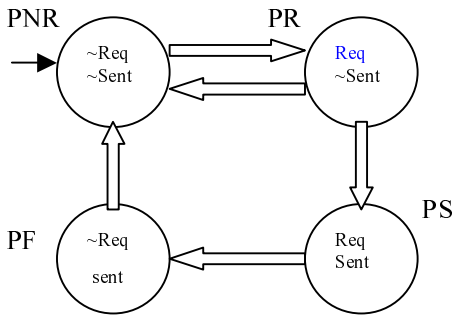

Figure 2 : Purchaser view of trade procedure Kripke model

Top

In the same way, we model the symmetric part of the global system (trade protocol), which is the Seller's automaton. The states of the automaton will be labelled respectively by SNR (seller's not ready for transaction), SR (seller's ready for transaction), SS (seller send the product) and SF (seller's finished transaction).

To model the global system, we have to obtain the simplified composition graph of the seller's and the purchaser's automata. In such operation the labels are used in providing synchronisation information in the composition, in the sense that a transition, t, from one automaton synchronises with a transition, t', from another if the labels of the two transitions are both equivalently the same. To be effective we proceed by the juxtaposition of the nodes (states) of the two automata and to construct the transition relations as discussed above.
The state diagram in the following figure (figure 3) represents the composition graph of the Purchaser automaton graph with those of the Seller automaton graph.

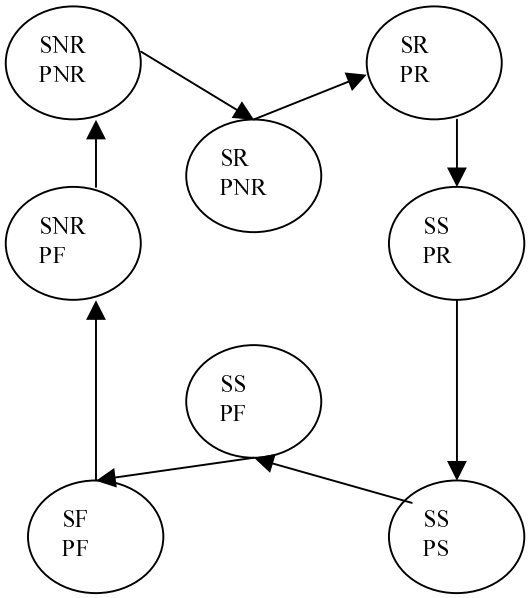

Figure 3 : Automaton of the Seller and Purchaser models Composition

We try now to be more exact. The above graphs are a graphical specification of the system. If we consider the model of the purchaser (figure 2), such specification is translated into a Boolean expression. The model uses four Boolean variables: Req, Sent, Req', and Sent'. Req and Sent represent variables in the present state, while Req' and Sent' represent variables in a next state. Each transition (arrow) in the state space is modelled as a conjunction of these variables. For example the bottom arrow is modelled as: ($\neg$Req $\wedge$ $\neg$Sent) and (Req' $\wedge$ $\neg$Sent'). The whole graph is then a disjunction of these conjunctions. Therefore, in this example the Boolean expression (transition relation) for the state space is:
R = ($\neg$Req $\wedge$ $\neg$Sent $\wedge$ Req' $\wedge$ $\neg$Sent')$\vee$(Req$\wedge$ $\neg$Sent$\wedge$ $\neg$ Req'$\wedge$ $\neg$Sent')$\vee$(Req $\wedge$ $\neg$Sent$\wedge$Req'$\wedge$ Sent')$\vee$(Req $\wedge$ Sent$\wedge$ $\neg$Req'$\wedge$Sent')$\vee$ ($\neg$Req $\wedge$ Sent $\wedge$ $\neg$Req' $\wedge$ $\neg$Sent')

The following table (table 3) gives the results (satisfiability) of the trading protocol.. The results show that the problem is satisfiable in the following cases:

Top

Table 3. Solutions of the trading protocol

Solution n° I	Req	Sent	Req'	Sent'
1	*false*	*true*	*false*	*false*
2	*false*	*false*	*true*	*false*
3	*true*	*false*	*false*	*false*
4	*true*	*false*	*true*	*true*
5	*true*	*true*	*false*	*True*

7. Checking some Properties of the E -business Trade Protocol

The ideal trade procedure (business protocol) must covers all the aspects of the exchange mode between sellers and purchasers and to prevent conflict situations to provide safety property (nothing bad will happen), liveness property ("something good will happen") and ill-defined contract. In the following, we give some conditions which must be checked to avoid system disturbance.

We wish to establish that the transaction between seller and any of the *n* purchaser's will eventually terminate.

The condition is expressed below (S_0 is the initial state).

$$S_0 \models G\,(EF\,(SNR \wedge A\,(PNR(1) \wedge PNR(2) \wedge \ldots \wedge PNR(n)))).$$

Where :

SNR : seller not ready.
PNR : purchase not ready.
SR : seller ready.
PR : purchase ready .

For the second property which we have to verify is to establish that there is no situation where a purchaser's obligation to pay for goods is activated before such goods available and delivered to him:

$$S0 \models AG((\neg PS(i) \wedge PS(i)) \Rightarrow SellersGAV\_T)$$

Where :

PS : purchase starts,

and :

SellersGav_T : good are available of the sellers

In the following assumption, we show how to verify specification.

• $EX(\ Req \wedge Sent\ )$

$= \exists\ v'.\ (R \wedge p')$

$= \exists\ (Req\ ',\ Sent\ ').\ (R \wedge (Req\ ' \wedge Sent\ '))$

Top

= ∃ (Req ', Sent '). ((¬Req ∧ ¬Sent ∧ Req' ∧ ¬Sent') ∨ (Req ∧ ¬Sent ∧ ¬ Req' ∧ ¬Sent') ∨ (Req ∧ ¬Sent ∧ Req' ∧ Sent') ∨ (Req ∧ Sent ∧ ¬Req' ∧Sent') ∨ (¬Req ∧ Sent ∧ ¬Req' ∧ ¬Sent')) ∧ (Req ' ∧ Sent '))

The solution using Maude system of such problem is : Req ∧ ¬ Sent.
Meaning that state (1, 0) satisfies EX (Req ∧ Sent) which corresponds to the state S2 of the figure 2.

EF(Req ∧ Sent) = μ **y. ((**Req ∧ Sent **)** ∨ EX**y)**

We proceed iteratively until to reach the fixpoint

τ 1 [0] = (Req ∧ Sent) ∨ EX0 = (Req ∧ Sent)
τ 2 [0] = (Req ∧ Sent) ∨ EX(Req ∧ Sent)
= (Req ∧ Sent) ∨ (Req ∧ ¬ Sent) {from the result of EX(v 0 ∧ v 1)}
= Req
τ 3 [0] = (Req ∧ Sent) ∨ EX(Req)
= (Req ∧ Sent) ∨ [∃ (Req', Sent'). (**R** ∧ Req)]
= (Req ∧ Sent) ∨ [∃ (Req', Sent'). ((¬Req ∧ ¬Sent ∧ Req' ∧ ¬Sent') ∨ (Req ∧ ¬Sent ∧ ¬ Req' ∧ ¬Sent') ∨ (Req ∧ ¬Sent ∧ Req' ∧ Sent') ∨ (Req ∧ Sent ∧ ¬Req' ∧Sent') ∨ (¬Req ∧ Sent ∧ ¬Req' ∧ ¬Sent')) ∧ Req')]

We iterate until $\tau_i = \tau_{i+1}$.

7.1 Some Programming Maude Codes

In the following, we introduce just a little part of the main Maude program defined as follows:

- The functional MU module : present the abstract data type used to define MU-calculus as an equivalence approach to the CTL operator according to the theorem of Clarke & Emerson (1981) [Theorem 3]. The general form is,

  Nu Y (<> (Y /\ (**Mu** Z (p \/ <> Z)))),

  and the operators Mu and Nu represent least and greatest fixed point operators respectively.

- The CTL module presents the structure (operators and operations) and the simplification rules.

Remarks : other modules have been neglected as for example the satisfaction module (SAT) based on a Reduced Binary Decision Diagram.[13-14]

```
(fmod MU is
sorts Variable Prop Formula .
subsort Variable < Formula .
subsort Prop < Formula .
ops True False : → Prop .
op ~_ : Formula → Formula [ prec 52 ] .                    *** Negative operator
op _/\_ : Formula Formula → Formula [ comm prec 55 ] .  *** AND operator
```

Top

```
op _\/_ : Formula Formula → Formula [ comm prec 59 ] .   *** OR    -//-
op <>_ : Formula → Formula [ prec 53 ] .
op '['_ : Formula → Formula [ prec 53 ] .
op Mu__ : Variable Formula → Formula [ prec 61 ] .
op Nu__ : Variable Formula → Formula [ prec 61 ] .
endfm)

(fmod CTL is
protecting SAT .
protecting MU .
sort CTLFormula .
subsort Prop < CTLFormula .
op !_ : CTLFormula → CTLFormula [ prec 53 ] .
ops EX_ : CTLFormula → CTLFormula [ prec 53 ] .
ops EG_ : CTLFormula → CTLFormula [ prec 63 ] .
ops E_U_ : CTLFormula CTLFormula → CTLFormula[ prec 63 ] .
ops AX_ : CTLFormula → CTLFormula [ prec 53 ] .
ops EF_ , AF_ , AG_ : CTLFormula → CTLFormula [ prec 63 ] .
ops A_U_ : CTLFormula CTLFormula → CTLFormula  [ prec 63 ] .
***   equivalence  between  CTL operators according to the theorem 3
Var P  Q  P1 P2 : Formula .
eq  AF P  =  P \/ AX AF P .
eq  EF P  =   P \/ EX EF P .
eq  AG P =   P /\ AX AG P .
eq  EG P  =   P /\ EX EG P .
eq  A(P U Q) = Q \/ (P /\ AX A(P U Q)) .
eq  E(P U Q)   = Q \/ (P /\ EX E(P U Q)) .
**** other reduction equivalence rules could be written in this part of the program, for example those
presented in the Table 2. ****
endfm)
```

8. Conclusion

Software and Hardware become more and more difficult to prove their correctness without errors due to the size (variables and constraints) of the corresponding mathematical models. To handle the complexity of the design, the necessity of looking for new techniques and algorithms become imperative.

We have shown that testing Boolean formulae, specifying the dynamic structure of a critical system is considered as an efficient Model Checking algorithm. The manipulation of such formulae becomes more easy using a rewriting-based system framework like Maude. The efficiency of Maude language is performed because of the deduction system of inference rules which can simulates both sequential and parallel distributed systems. Such rules facilitate the normalisation (normal form) of the corresponding model to deal practically with CTL and to compare the specification and the implementation of critical systems in term of the axiomatization of the Boolean equations. As an extension of the actual work, we plan to generalize the rewriting logic directly to express CTL logic using the eval function presented in the Algorithm 2, instead of fixpoints-search model checking algorithms. This future perspective opens interesting implementation problems due to the efficiency of the theoretical point of view aspect of rewriting logic and the efficient Maude engine which is planned to execute 15 millions instruction per second.

9. References

[1] M. Kaufmann, J. S. Moore, ACL2 an Industrial Strenght of Nqthm, In the Conference of Computer Assurance (COMPASS-96), pages 23-34, IEEE, computer Society Press, 1996.

[2] K. L. McMillan. A Methodology for Hardware Verification using Compositional Model-Checking. Sci- of Comp-Prog-, 37(1-3):279-309, May 2000.

[3] M. Clavel, F. J. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, J. F. Quesada, Maude Specification and Programming in Rewriting Logic, Mar. 1999.
Maude System documentation http://maude.csl.sri.com/papers.

[4] G. W., Brams, Reseaux de Petri : Theorie et Pratique, Edition Masson, 1983.

[5] J. Y. Girard, Linear Logic, Theoretical Computer Science, 50: 1-102, 1987.

[6] P. Boravansky, C. Kirchner, H. Kirchner, P. –R Moreau, C. Ringeissen, An Overview of Elan, In C. Kirchner and H. Kirchner, editors, Proc. Second International Workshop on Rewriting Logic and Applications, Electronic Notes in Theoretical Computer Science, Pont-à Mousson (France), September, 1998. Elsevier.

[7] R. Bons, R. Lee, M. Wagenaar and C. D. Wrigley, Modelling Inter-organizational Trade Procedures using Documentary Petri Nets, HICCS 95, 1995.

[8] J. Meseguer and J.A. Goguen, "Order-Sorted Algebra I: Equational Deduction for Multiple Inheritence, Overloading, Exceptions and Partial Operation", Technical Report, SRI-CSL-89-10, July, 1989.

[9] E.M Clarke, E.A.,Emerson, and A.P, Sistla, Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications, ACM Transactions on Programming Languages, 8 (2), 1986, pages 244–263.

[10] Z. Manna, A. Pnueli, Verification of Concurrent Programs: the Temporal Framework in the Correctness Problem in Computer Science, Ed. R S. Boyer and J. S Moore, International Lecture Series in Comp. Sci. Academic Press, NewYork, 1981.

[11] J.P Queille, J. Sifakis, Fairness and Related Properties in Transition Systems, Research Report 292, IMAG, Grenoble, 1982.

[12] E.M Clarke, E. A. Emerson, Design and Synthesize of Synchronisized Squeletons using Branching Time Temporal Logic. LNCS 131, 52-71, 1981.

[13] R. E. Bryant, Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams, Carnegie Mellon, Technical Report CMU-CS-92-160, 1992.

[14] T. A. Chen , B. Yang, R. E. Bryant, Breadth-First with Depth-First BDD Construction: A hybrid Approach. School of Computer Science, Carnegie Mellon University. CMU-98-120.

[15] M.L Rebaiaia, M. Benmohamed, J. Jaam, A. Hasnah, Verification-Based Algorithms for Integer Programming and Genetic Algorithms, In the International Journal of Applied Science and Computation, Vol.10, No.3, 2003.

Top