Generalized Quantitative Temporal Reasoning: An Automata Theoretic-Approach *

E. Allen Emerson and Richard J. Trefler

Computer Sciences Department and Computer Engineering Research Center
University of Texas, Austin TX, 78712, USA

Abstract. This paper proposes an expressive extension to Propositional Linear Temporal Logic dealing with real time correctness properties and gives an automata-theoretic model checking algorithm for the extension. The algorithm has been implemented and applied to examples.

1 Introduction

In a landmark paper, [Pn77], Pnueli identified a very general and important class of computing systems now called 'reactive systems' (cf. [HP85] [Pn86]). Characterized by their ongoing behavior, reactive systems and their sub-components interact with an environment over which they have little control. Such systems, e.g. operating systems, tend to be quite complex and they have necessitated the development of powerful tools for their verification. In [Pn77] it was argued that temporal logic is a highly appropriate formalism for specifying and verifying the ongoing operation of reactive systems.

Propositional Linear Time Logic (PLTL) [Pn77] allows the simple expression of many important system properties at a qualitative level. Using operators such as 'G' and 'F' meaning, respectively, 'always' and 'sometime' PLTL can express the requirement that 'every *request* from a client should be eventually met with a *response* from the server' as $\mathsf{G}(request \Rightarrow \mathsf{F} response)$.

Recently, however, it has been recognized that in many applications the specification of correct operation requires quantitative as well as qualitative properties. Real time systems, those systems whose correct operation includes time-critical specifications, require such quantitative analysis. One can introduce quantitative operators such as '$\mathsf{F}^{\preceq 5}$' which, informally, means 'sometime before more than five time units have elapsed'. With the resulting formalism we can express properties such as 'every *request* from a client should be met with a *response* from the server within five time units' as $\mathsf{G}(request \Rightarrow \mathsf{F}^{\preceq 5} response)$.

In this paper we present a simple but general framework for handling an enriched class of quantitative problems. Our formalism, RTPLTL+ (Real Time PLTL+), is an extension of PLTL that employs natural notations from formal language and automata theory. In particular we have identified an expressive

* This work was supported in part by NSF grant CCR9415496 and SRC contract 95-DP-388.

yet tractable fragment of regular expressions enhanced with 'and', 'negation', and 'exponentiation' operators. Testing emptiness of arbitrary extended regular expressions is non-elementary, however, the fragment used here in conjunction with PLTL can be tested for emptiness in time exponential in the size of the regular expression. An example of the types of specifications we are interested in, a constraint on the set of computations of a system, is exhibited below.

The term $(\overline{\mathit{request} + \mathit{response}}^{*}\mathit{request})$ is a requirement on strings of system actions specifying strings which contain *request* as the last element of the string and no occurrences of either *request* or *response* anywhere else in the string. $(\overline{\mathit{request} + \mathit{response}}^{*}\mathit{request})^3$ specifies three consecutive occurrences of strings satisfying $(\overline{\mathit{request} + \mathit{response}}^{*}\mathit{request})$, i.e. *request* occurs three times and *response* has not occurred. *true* specifies any computation; therefore the subformula $(\overline{\mathit{request} + \mathit{response}}^{*}\mathit{request})^3\mathit{true}$ is satisfied by any computation with a prefix satisfying $(\overline{\mathit{request} + \mathit{response}}^{*}\mathit{request})^3$. Similarly, $(\overline{\mathit{response}}^{*}\mathit{response}) \cap (\overline{\mathit{request}}^{*}\mathit{request})^{\preceq 3}$ specifies that exactly one *response* has occurred while fewer than four *request*s have occurred. These fragments are used to express the following specification. If the server ever receives three successive *request*s from a client, and the server has issued no *response* since receiving the first *request*, then the server will issue a *response* before receiving a fourth *request*. This is expressed as $\mathsf{G}((\overline{\mathit{request} + \mathit{response}}^{*}\mathit{request})^3\mathit{true} \Rightarrow ((\overline{\mathit{response}}^{*}\mathit{response}) \cap (\overline{\mathit{request}}^{*}\mathit{request})^{\preceq 3})\mathit{true})$.

Verifying that a reactive system obeys a specification, written as a formula in one of the formalisms mentioned above, can be accomplished with a technique known as model checking [CE81] (cf. [QS82]). Model checkers answer the question 'given a specific reactive system M and a formula ϕ, do all computations of M satisfy the formula ϕ?' We present an automata-theoretic model checking algorithm that allows us to model check formulae of RTPLTL+ over general representations of reactive systems. The algorithm has been implemented on top of the SMV [Mc92] model checking system.

Section 2, below, discusses syntax and semantics. Model checking is described and analyzed in Section 3. Section 4 contains some examples and discusses the implementation of the model checking algorithm. Finally, section 5 contains a summary.

2 Preliminaries

2.1 Syntax

The full paper [ET96] presents a unified syntax for CTL, PLTL, CTL* and certain quantitative extensions, viz., RTCTL, RTPLTL, RTCTL+, RTPLTL+ and RTCTL*+. Here, however, we will focus on PLTL and its extension RTPLTL+.

We use the symbol AP to denote the finite set of underlying atomic proposition symbols. ACT denotes the finite set of atomic action symbols. Elements of AP will be represented by P, Q, *etc.*, elements of ACT by B, C, D, *etc.*, and $\mathcal{N}$ will represent the set of non-negative integers.

The set of regular expressions over ACT are constructed by the following rules.

E1 For each $B \in ACT$, B is a regular expression.
E2 λ is a regular expression.
E3 For r, r' regular expressions, (rr'), $(r \wedge r')$; and $(\overline{r})$ are regular expressions.
E4 For r a regular expression, $i \in \mathcal{N}$, $(r^*), (r^i), (r^{\leq i})$, and $(r^{\geq i})$ are regular expressions.

Path formulae are formed according to the rules:

P1. Each atomic proposition P is a formula.
P2. If ϕ and ψ are path formulae then so are $\neg\phi$ and $\phi \wedge \psi$.
P3a. If ϕ and ψ are path formulae then so are $\mathsf{X}\phi$ and $(\phi \mathsf{U} \psi)$.
P3b. If ϕ and ψ are path formulae and r is a regular expression then $(\phi \mathsf{U}^r \psi)$ is a path formula.

In the sequel we will sometimes drop parentheses from formulae and expressions when the parsing seems clear.

PLTL is the set of formulas formed by rules P1, P2, and P3a while Regular PLTL (RPLTL) extends PLTL with rule P3b.

RTPLTL+ is a subset of RPLTL that restricts the type of regular expressions allowed in rule P3b. Supposing ACT $= \{B_1, \dots, B_n\}$ then we will sometimes use ACT to denote the regular expression $\overline{(\overline{B_1} \wedge \dots \wedge \overline{B_n})}$.

Let $n \in \mathcal{N}$ and $B \in$ ACT a term is one of '$= nB$', '$\preceq nB$', or '$\succeq nB$' which are shorthands for $((\overline{B})^* B)^n$, $((\overline{B})^* B)^{\leq n} (\overline{B})^*$, and $((\overline{B})^* B)^n (\text{ACT})^*$ respectively. A *ce* expression is any boolean combination of terms.

Let $m, n, b \in \mathcal{N}$, $i \in [1:n]$, $B_i, C \in$ ACT and $\gamma_i \subseteq$ ACT such that $B_i \in \gamma_i$. If $\gamma_i = \{B_i, D_1, \dots, D_m\}$ then $\overline{\gamma_i}$ is a shorthand for $\overline{(B_i + D_1 + \dots + D_m)}$, which, to avoid the proliferation of parentheses, may be written as $\overline{B_i + D_1 + \dots + D_m}$. Regular formulae are formed by the four rules below.

R1a. $(\overline{\gamma_1}^* B_1 \dots \overline{\gamma_n}^* B_n)$ is a regular formula.
R1b. $(\overline{\gamma_1}^* B_1 \dots \overline{\gamma_n}^* B_n)^{\succeq b}$, a shorthand for $(\overline{\gamma_1}^* B_1 \dots \overline{\gamma_n}^* B_n)^b (\text{ACT})^*$, is a regular formula.
R1c. $(\overline{\gamma_1}^* B_1 \dots \overline{\gamma_n}^* B_n)^{\preceq b}$, a shorthand for $\overline{(\overline{\gamma_1}^* B_1 \dots \overline{\gamma_n}^* B_n)^{b+1} (\text{ACT})^*}$, is a regular formula and is .
R2. If ρ_1 and ρ_2 are regular formulae then so are $(\rho_1 \rho_2)$ and $(\rho_1 \cap \rho_2)$ which are shorthands for $(\rho_1 \wedge \overline{\rho \text{ACT}^*})\rho_2$ and $(\rho_1 \wedge \rho_2)$ respectively.

RTPLTL+ is the subset of RPLTL such that for any sub-formula $\phi \mathsf{U}^r \psi$ either r is a *ce* expression or $\phi = \neg(P \wedge \neg P)$ and $r = \rho \wedge \overline{(\rho \text{ACT}^*)}$ for some regular formula ρ. When dealing with regular expressions which contain a ρ formula we typically write $(\neg(P \wedge \neg P)) \mathsf{U}^{\rho \wedge \overline{(\rho \text{ACT}^*)}} \psi$ as $(\rho)\psi$.

Derived operators, similar to PLTL, $\mathsf{F}, \mathsf{G}^{ce}$ and $(\backslash(\rho))\phi = \neg(\rho(\neg\phi))$ are also allowed.

We also use the following shorthand notation. Given formulae of the form $((\rho_1 \rho_2) \dots \rho_n)$, if the ρ_i are all identical then we will write $(\rho_1)^n$ as a shorthand for $((\rho_1 \rho_2) \dots \rho_n)$.

2.2 Semantics

Before defining the semantics of the formulae, some intuition regarding regular formulae may be in order. Formulae of the type $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)$ have a straightforward meaning. These formulae express restrictions on the order of the atomic actions of computations (paths through a structure); furthermore, the meaning of the formulae is equivalent to the meaning of their identical regular expressions. $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^b$ is a shorthand for b copies of $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)$ and formulae of this type are also equivalent to their identical regular expressions. However, formulae of the form $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\preceq b}$ do not have a meaning equal to their identical regular expressions. $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\preceq b}$ expresses the requirement that there are no more than b occurrences of the sequence $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)$, it does not require that there exists a $b' \in [0 : b]$ such that $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{b'}$ be satisfied. In particular $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\preceq 0}$ is true of a sequence so long as the sequence does not satisfy $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)$. While the empty string satisfies these requirements it is not the only string that does so.

Temporal logics, such as PLTL, are usually interpreted over the computations or paths in (Kripke) structures, cf.[Ar94]. A Kripke structure is a triple which consists of a set of states S, a transition relation on the state set R, and a labeling function L. L labels the states and/or transition relation arcs with, respectively, the atomic propositions true at a state and the atomic actions associated with transitions.

Unlike RTCTL, defined in [EMSS90], RTPLTL+ does not implicitly associate a 'clock event' with each transition. Here we can denote clock events by a distinguished action C and stipulate that the clock ticks infinitely often. In fact RTPLTL+ allows the use of multiple independent clocks.

Let $M = (S, R, L)$ be a structure such that S is a finite set of states. $R \subseteq S \times (\text{ ACT } \times S)$ is a total transition relation and $L : S \cup R \rightarrow 2^{\text{ AP }} \cup \text{ACT}$ such that for all $s \in S, L(s) \in 2^{\text{ AP }}$ and for all $s, s' \in S$, and $\sigma \in$ ACT such that $(s, \sigma, s') \in R$, $\text{L}(s, \sigma, s') = \sigma$.

Let x be a 'full path' in M, then x is of the form $x_0\sigma_0 x_1 \sigma_1 \ldots$ where for $i \geq 0$, $x_i \in S$, $\sigma_i \in$ ACT and $(x_i, \sigma_i, x_{i+1}) \in R$. x_i, σ_i denote, respectively, the ith state and the ith action of a path while x^i denotes the full path $x_i \sigma_i x_{i+1} \sigma_{i+1} \ldots$, and $x|\text{ACT}$ denotes the projection of x onto ACT.

Given a full path x in M we denote that x satisfies or models path formula ϕ by $M, x \models \phi$. Similarly x does not satisfy ϕ is denoted by $M, x \not\models \phi$. When M is understood we will sometimes drop it from the $\models$ notation.

$\models$ is defined for RPLTL formulae by the following rules.

Let $\sigma \in \text{ACT}^*$ then the meaning of regular expression r is defined as follows.

ES1 $\sigma \in B$, for $B \in ACT$ iff $\sigma = B$.
ES2 $\sigma \in \lambda$ iff σ is the empty string.
ES3 If $r = (r_1 r_2)$ then $\sigma \in r$ iff $\sigma = \sigma 1 \sigma 2$ such that $\sigma_1 \in r_1$ and $\sigma_2 \in r_2$. If $r = (r_1 \wedge r_2)$ then $\sigma \in r$ iff $\sigma \in r_1$ and $\sigma \in r_2$. If $r = \overline{r_1}$ then $\sigma \in r$ iff $\sigma \notin r_1$.
ES4 If $r = (r_1)^0$ then $\sigma \in r$ iff $\sigma \in \lambda$. $r = (r_1)^i$, for $0 < i$, then $\sigma \in r$ iff $\sigma = \sigma_1 \sigma_2$ and $\sigma_1 \in r$ and $\sigma_2 \in (r_1)^{i-1}$. If $r = (r_1)^{\leq i}$ then $\sigma \in r$ iff there

exists $j \leq i$ such that $\sigma \in (r_1)^j$. If $r = (r_1)^{\geq i}$ then $\sigma \in r$ iff there exists $j \geq i$ such that $\sigma \in (r_1)^j$. If $r = (r_1)^*$ then $\sigma \in r$ iff there exists a $j \in \mathcal{N}$ such that $\sigma \in (r_1)^j$.

Let $x = x_0\sigma_0 \ldots$ be a full path in M, ϕ, ψ, ψ' are path formulae and r is a regular expression then

PS1. $\phi = P$ for some $P \in \mathrm{AP}$: $M, x \models \phi$ iff $P \in L(x_0)$.
PS2. $\phi = \neg\psi$: $M, x \models \phi$ iff $M, x \not\models \psi$. $\phi = \psi \wedge \psi'$: $M, x \models \phi$ iff $M, x \models \psi$ and $M, x \models \psi'$.
PS3a. $\phi = \mathsf{X}\psi$: $M, x \models \phi$ iff $M, x^1 \models \psi$. $\phi = \psi \mathsf{U} \psi'$: $M, x \models \phi$ iff there exists $i \in \mathcal{N}$ such that $M, x^i \models \psi'$ and for all $j \in [0 : i-1]$, $M, x^j \models \psi$.
PS3b. $\phi = \psi \mathsf{U}^r \psi'$: $M, x \models \phi$ iff there exist $i \in \mathcal{N}$ such that $\sigma_0 ... \sigma_{i-1} \in r$, $M, x^i \models \psi'$ and for all $j \in [0 : i-1]$, $M, x^j \models \psi$.

We denote the length of an RTPLTL+ formula ϕ by $|\phi|$ and the magnitude of the formula by $||\phi||$. $|\phi|$ corresponds to the number propositions and operators. When ϕ is an atomic proposition it has magnitude 0. $||\neg\phi|| = ||\phi||$, and when ϕ is a positive boolean combination of ϕ' and ϕ'' then and $||\phi|| = ||\phi'|| + ||\phi''||$. Formulae of the form $\mathsf{X}\phi$ have magnitude $||\phi||$; formulae of the form $\phi \mathsf{U} \psi$ have magnitude $||\phi|| + ||\psi||$. ce terms kB, $\preceq kB$ and $\succeq kB$ all have magnitude k. $||\neg ce|| = 1 + ||ce||$ and $||ce \wedge ce'|| = ||ce|| \cdot ||ce'||$. Then $||\phi \mathsf{U}^{ce} \psi|| = ||\phi|| + ||\psi|| + ||ce||$. Regular formulae of type R1a have (respectively R1b, R1c) have magnitude $n(\max(|\gamma_i|))$, where $|\gamma_i|$ is equal to the number of elements in the set γ_i, ($bn(\max(|\gamma_i|))$, $bn(\max(|\gamma_i|))$). Formulae of the type $(\rho_1 \rho_2)$ and $(\rho_1 \cap \rho_2)$ have magnitude $||\rho_1|| + ||\rho_2||$ and $||\rho_1|| \cdot ||\rho_2||$, respectively. Finally, $||\rho\phi|| = ||\rho|| + ||\phi||$.

A formula is in positive normal form, PNF, when only propositional constants are negated. Using the appropriate short forms, given above, and DeMorgan rules any RTPLTL+ formula ϕ can be transformed into an equivalent formula ϕ' which is in PNF, in time linear in the length of of ϕ.

3 Model Checking RTPLTL+

Given structure $M = (S, R, L)$, as defined above, and a formula ϕ of RTPLTL+ we define a model checking procedure which determines whether there is a path x in M such that $M, x \models \phi$. This is the dual of the question posed in the introduction but can be shown to be equivalent via the following observation. The computations of M satisfy specification ϕ iff there is no computation x of M such that $M, x \models \neg\phi$

We extend a standard automata theoretic technique to decide this problem [VW86]. The technique consists of creating an automaton, $\mathcal{A}_{\neg\phi}$, on infinite strings, cf.[Bu62] and [NP85], which accepts only those strings which satisfy the formula $\neg\phi$. Combine the structure M with $\mathcal{A}_{\neg\phi}$ to form the product automaton $M \times \mathcal{A}_{\neg\phi}$. $M \times \mathcal{A}_{\neg\phi}$ is an automaton, on infinite strings, whose language is empty if and only if M is a model of ϕ.

Before considering the automaton for arbitrary RTPLTL+ formula ϕ we first define automata which recognize infinite strings that satisfy formulae of the form $\rho true$ and automata which recognize finite strings that satisfy counting expressions.

Suppose $\psi = \rho true$ such that $\rho = ((\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n) \cap (\overline{C}^* C)^{\preceq b})$, and for all $i \in [1 : n]$, $B_i \neq C$. $\mathcal{A}_\rho = (\mathrm{ACT}, \mathcal{Q}, \delta, q_{(0,0)}, F)$ is a Büchi automaton where $\mathcal{Q} = \{q_{(0,0)}, \ldots, q_{(0,b+1)}, \ldots, q_{(n,0)}, \ldots, q_{(n,b)}\}$, $F = \{q_{(n,0)}, \ldots, q_{(n,b)}\}$, and δ : $\mathcal{Q} \times \mathrm{ACT} \to \mathcal{Q}$ is a deterministic transition relation defined by the transition diagram in figure 1 . Note that in the figure Σ stands for ACT, $\Sigma 1 = (\Sigma \setminus \gamma_1) \setminus \{C\}$, $\Sigma 2 = (\Sigma \setminus \gamma_2) \setminus \{C\}$, *etc*, and $\gamma_i' = \gamma_i \setminus \{B_i, C\}$. In the sequel we shall sometimes refer to states q_{f} and $q_{\mathrm{f'}}$, the states so marked in the diagram.

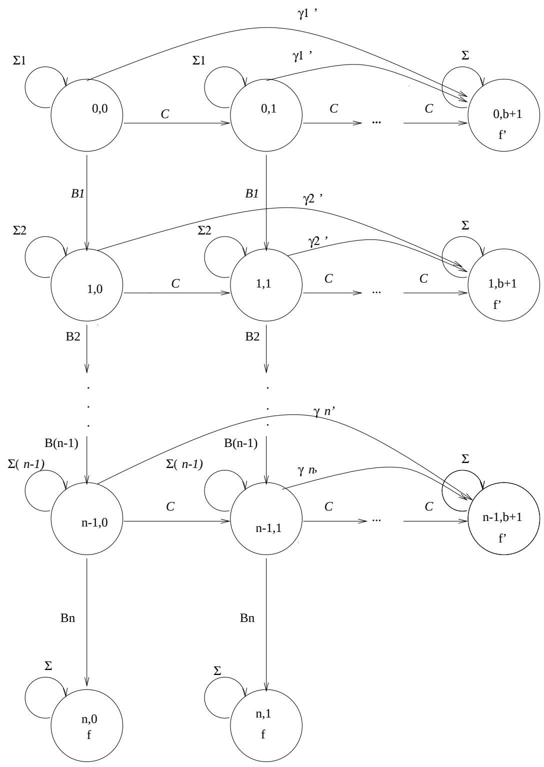

Fig. 1. automaton for $((\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n) \cap (\overline{C}^* C)^{\preceq b})true$

As constructed $\mathcal{A}_\rho$ accepts ω-strings over the alphabet ACT that conform to ρ, *i.e.* the strings contain B_1, B_2 to B_n in order before the appearance of more more than b C's and no action in γ_1 occurs before B_1, no action in γ_2 occurs between the first occurrence of B_1 and the next occurrence of B_2, *etc.*

We can in an algorithmic manner construct automata like the above for all the ρ expressions in the language; the details are straightforward and have been left out due to space restrictions.

We will sometimes refer to formulae such as ψ (respectively ψ'), formulae with unnegated (negated) regular components as their primary connective, as positive (negative) formulae. By extension we refer to $\mathcal{A}_\rho$ ($\mathcal{A}_{\overline{\rho}}$) as positive (negative) automata.

Claim 1 *Let x be a full path in arbitrary M and ψ and ψ' formulas as above then $x \models \psi$ iff $(x|ACT) \in \mathcal{L}(\mathcal{A}_\rho)$ and $x \models \psi'$ iff $(x|ACT) \in \mathcal{L}(\mathcal{A}_{\overline{\rho}})$.*

Let ce be a counting expression, then there exists a deterministic finite automaton $\mathcal{A}_{ce} = (\mathrm{ACT}, \mathcal{Q}, \delta, q_0, F)$ such that for all $\sigma \in \mathrm{ACT}^*$, $\sigma \in \mathcal{L}(\mathcal{A}_{ce})$ iff $\sigma \models ce$. Constructed recursively from the structure of ce according to the rules for creating product and complementary finite automata, the basic idea is to keep track of the number of occurrences of the actions specified in the counting expressions.

Claim 2 *Given a counting expression ce, deterministic automaton $\mathcal{A}_{ce}$ can be constructed in time linear in $\|ce\|$ such that $\mathcal{L}(\mathcal{A}_{ce}) = \{\sigma \in ACT^* \mid \sigma \models ce\}$ and $|\mathcal{A}_{ce}|$ is linear in $\|ce\|$.*

Let ϕ be a formula of RTPLTL+ in PNF. For each regular sub-formula ρ ($\overline{\rho}$) and counting expression ce there is a corresponding automaton $\mathcal{A}_\rho$ ($\mathcal{A}_{\overline{\rho}}$) or $\mathcal{A}_{ce}$. Number these automata $1...a$. Then for $j \in [1:a]$, $\mathcal{A}_j = (\mathrm{ACT}, \mathcal{Q}_j, \delta_j, q_0^j, F_j)$ and we refer to the i-th state of the j-th automata by q_i^j.

Theorem 3. *Given a formula ϕ of RTPLTL+ there is a Büchi automaton $\mathcal{A}_\phi$ such that for any structure $M = (S, R, L)$ and full path x of M, $M, x \models \phi$ iff $x \in L(\mathcal{A}_\phi)$.*

Proof: We proceed as follows. Using a modified version of the tableaux construction for PLTL, a tableaux T is constructed from the formula ϕ. T encodes models of ϕ and we can use the structure of T to form the automaton $\mathcal{A}_\phi$.

Before describing the tableaux construction we give a categorization of RTPLTL+ formulae as elementary or non-elementary formulae. Non-elementary formulae are then separated into Alpha-formulae and Beta-formulae. Intuitively, an Alpha-formulae ϕ with constituents ψ, ψ' is true iff ψ and ψ' are true while a Beta-formula ϕ with constituents ψ and ψ' is true iff one or both of the constituents is true. Note that in the following we will abuse notation and consider individual states of the automata $\mathcal{A}_j$ as formulae.

Propositions and formulae of the form $\mathsf{X}\phi$ are elementary. The following lists characterize Alpha- and Beta-formulae and give their constituent formulae. Alpha-formulae : $\phi \wedge \phi'$ with constituents ϕ and ϕ'; q_{f}^j, where $\mathcal{A}_j$ is the automaton associated with $\rho\phi$ or $\overline{\rho}\phi$ with constituent ϕ; $\rho\phi$, where $\mathcal{A}_j$ is the automaton associated with $\rho\phi$ with constituent q_0^j; $\overline{\rho}\phi$, where $\mathcal{A}_j$ is the automaton associated with $\overline{\rho}\phi$ with constituent q_0^j; $\phi \mathsf{U}^{ce} \phi'$, where $\mathcal{A}_j$ is the automaton for ce with constituent $\phi\mathsf{U}^{q^j_0}\phi'$; $\phi\mathsf{U}^{q^j_i}\phi'$, where $q^j_i \notin F_j$ with constituents ϕ and $\mathsf{X}(\phi\mathsf{U}^{q^j_{h0}}\phi') \vee \ldots \vee \mathsf{X}(\phi\mathsf{U}^{q^j_{hn}}\phi')$ where $q^j_{h0}, \ldots, q^j_{hn}$ are the successor states of q^j_i; $\phi\mathsf{V}^{ce}\phi'$, where $\mathcal{A}_j$ is the automaton for ce : with constituent $\phi\mathsf{V}^{q^j_0}\phi'$. Beta-formulae : $\phi \vee \phi'$ with constituents ϕ and ϕ'; $\phi\mathsf{U}\phi'$ with constituents ϕ' and $\phi \wedge \mathsf{X}(\phi\mathsf{U}\phi')$; $\phi\mathsf{V}\phi'$ with constituents $\phi \wedge \phi'$ and $\phi \wedge \mathsf{X}(\phi\mathsf{V}\phi')$; $\phi\mathsf{U}^{q^j_i}\phi'$, where $q^j_i \in F_j$ with constituents ϕ' and $\phi \wedge (\mathsf{X}(\phi\mathsf{U}^{q^j_{h0}}\phi') \vee \ldots \vee \mathsf{X}(\phi\mathsf{U}^{q^j_{hn}}\phi'))$ where $q^j_{h0}, \ldots, q^j_{hn}$ are the successor states of q^j_i; $\phi\mathsf{V}^{q^j_i}\phi'$, where $q^j_i \notin F_j$ with constituents ϕ' and $\mathsf{X}(\phi\mathsf{V}^{q^j_{h0}}\phi') \vee \ldots \vee \mathsf{X}(\phi\mathsf{V}^{q^j_{hn}}\phi')$ where $q^j_{h0}, \ldots, q^j_{hn}$ are the successor states of q^j_i; $\phi\mathsf{V}^{q^j_i}\phi'$, where $q^j_i \in F_j$ with constituents $\phi \wedge \phi'$ and $\phi \wedge (\mathsf{X}(\phi\mathsf{V}^{q^j_{h0}}\phi') \vee \ldots \vee \mathsf{X}(\phi\mathsf{V}^{q^j_{hn}}\phi'))$ where $q^j_{h0}, \ldots, q^j_{hn}$ are the successor states of q^j_i; q^j_i, where q^j_i is not labeled with f with constituents $\mathsf{X}(q^j_{h0})$, ... , $\mathsf{X}(q^j_{hn})$ where $q^j_{h0}, \ldots, q^j_{hn}$ are the successor states of q^j_i.

The tableaux for a formula ϕ is created by 'growing' a finite graph whose nodes represent sets of sub-formulae of ϕ which are satisfied along computations satisfying ϕ. Nodes are labeled by 'downwardly' consistent sets of sub-formulae, *i.e.* if a node is labeled by an Alpha formula ψ then it is also labeled by both of ψ's constituents. If ψ is a Beta formula then the node must be labeled with at least one of ψ's constituents. Nodes with no next-time formulae have a single successor which is labeled by the empty set. Otherwise, node V's successors consist of the entire set of nodes which are first labeled with ψ iff $\mathsf{X}\psi$ is in the label of V, and then are made downwardly consistent. Arcs from a node, V to its successor(s), U are labeled by actions $B \in$ ACT according to the following rules : for all $q^j_i \in V$, $q^j_i \notin F_j$ there is a $q^j_{i'} \in U$ such that $\delta_j(q^j_i, B) = q^j_{i'}$; for all $q^j_{i'} \in U$ either $i' = 0$ and $\mathcal{A}_j$ is the automaton for ρ ($\overline{\rho}$) and $\rho\phi \in U$ ($\overline{\rho}\phi \in U$), or there is an $q^j_i \in V$ and $\delta_j(q^j_i, B) = q^j_{i'}$; for all $\phi\mathsf{U}^{q^j_i}\psi \in V$ either $q^j_i \in F_j$ and $\psi \in V$, or there is an $\phi\mathsf{U}^{q^j_{i'}}\psi \in U$ and $\delta_j(q^j_i, B) = q^j_{i'}$; for all $\phi\mathsf{U}^{q^j_{i'}}\psi \in U$ either $i' = 0$ and $\phi\mathsf{U}^{ce}\psi \in U$ and $\mathcal{A}_j$ is the automaton for ce, or there is a $\phi\mathsf{U}^{q^j_i}\psi \in V$ and $\delta_j(q^j_i, B) = q^j_{i'}$; for all $\phi\mathsf{V}^{q^j_i}\psi \in V$ then $\psi \in V$ and $\phi \in V$, or $\psi \in V$ and $q^j_i \notin F_j$, or $q^j_i \notin F_j$ and there is a $\phi\mathsf{V}^{q^j_{i'}}\psi \in U$ such that $\delta_j(q^j_i, B) = q^j_{i'}$, or $\phi \in V$ and there is a $\phi\mathsf{V}^{q^j_{i'}}\psi \in U$ such that $\delta_j(q^j_i, B) = q^j_{i'}$; for all $\phi\mathsf{V}^{q^j_{i'}}\psi \in U$ either $i' = 0$ and $\phi\mathsf{V}^{ce}\psi \in U$ and $\mathcal{A}_j$ is the automaton for ce, or there is a $\phi\mathsf{V}^{q^j_i}\psi \in V$ and $\delta_j(q^j_i, B) = q^j_{i'}$. When no such B exists we label the arc with the empty set. When V contains no automata related formula then the arc is left unlabeled, meaning that any $B \in$ ACT can cause that transition.

We identify similarly labeled nodes by one representative with multiple incoming and outgoing arcs. By requiring the uniqueness of node labels, it is guaranteed that the graph is finite, and of size no more than double exponential in the length of formula ϕ.

Once the graph has been completed it is pruned by removing any inconsistent nodes. Any remaining eventualities are then numbered and a Büchi acceptance condition is then used to ensure that no eventuality is pending forever.

Given a non-empty tableaux T for formula ϕ we construct a Büchi automaton

$\mathcal{A}_\phi$ whose language contains all stings in $(2^{\mathrm{AP}} \times \mathrm{ACT})^\omega$ satisfying ϕ and does not contain any string that does not satisfy ϕ.

$\mathcal{A}_\phi = (\Sigma, \mathcal{T}, \delta, \mathcal{T}_0, F)$ where $\Sigma = 2^{\mathrm{AP}} \times \mathrm{ACT}$, $\mathcal{T} = (\mathrm{AND} \times \{0, \ldots, l\}) \cup \mathbf{sink}$, where AND is the set of nodes of T, and $\mathcal{T}_0 = \{\langle t, 0\rangle | \phi \in t\}$. $\delta : \Sigma \times \mathcal{T} \to 2^{\mathcal{T}}$ such that $\langle t', k'\rangle \in \delta(\langle t, k\rangle, \langle s, \sigma\rangle)$ iff for all $P \in t, P \in L(s)$, for all $\neg P \in t, P \notin L(s)$, t' is a child of t in T, σ is an element of the subset of ACT which labels the arc from t to t', and if eventuality k is pending in t then $k = k'$ otherwise $k' = (k+1) \bmod (l+1)$. $\mathbf{sink} \in \delta(\langle t, k\rangle, \langle s, \sigma\rangle)$ iff t contains no next time formulae and for all $P \in t, P \in L(s)$ and for all $\neg P \in t, P \notin L(s)$. $\mathbf{sink} \in \delta(\mathbf{sink}, \langle s, \sigma\rangle)$ for all $\langle s, \sigma\rangle \in \Sigma$. Finally, $F = \{\mathbf{sink}\} \cup \{\langle t, k\rangle | k = 0\}$.

The theorem follows from the construction of the automaton and the definition of the satisfaction relation for RTPLTL+ formulae. □

Theorem 4. *$\mathcal{L}(M \times \mathcal{A}_\phi) \neq \emptyset$ iff there is a full path x in M such that $M, x \models \phi$.*

Proof: The proof is immediate from the previous theorem. □

Theorem 3 gives a model checking procedure that runs in time linear in the size of the structure M and polynomial in the size of the tableaux for formula ϕ. T, the tableaux for ϕ, is at most of size exponential in the $|\phi| + ||\phi||$ since each node has a unique label.

Theorem 5. *Given a formula ϕ of RTPLTL+ and structure $M = (S, R, L)$, let size $= |\phi| + ||\phi||$, then the model checking problem 'do the computations of M satisfy ϕ' is decidable in time $\mathcal{O}(|M| \times \mathbf{EXP}(size))$.*

Proof: Theorem 3 gives a method for creating the Büchi automaton $\mathcal{A}$ for the RTPLTL+ formula $\neg(\Phi \Rightarrow \phi)$ which accepts only those computations that satisfy Φ and do not satisfy ϕ. From the construction in the theorem $\mathcal{A}$ is of size exponential in the length and magnitude of the formula $\neg(\Phi \Rightarrow \phi)$.

Form the product automaton $M \times \mathcal{A}$, and test this automaton for emptiness. Testing Büchi automaton $\mathcal{A}'$ for emptiness is in $\mathcal{O}(|\mathcal{A}'|)$. Hence we can test whether $\mathcal{L}(M \times \mathcal{A}) = \emptyset$ in time linear in the size of $M \times \mathcal{A}$. $\mathcal{L}(M \times \mathcal{A}) = \emptyset$ iff for all computations x of M, $M, x \not\models \neg(\Phi \Rightarrow \phi)$ iff for all computations x of M, $M, x \models (\Phi \Rightarrow \phi)$ iff M is a fair model of ϕ. □

The structure M is typically of immense size while the specification formula is usually small. Since the model checking algorithm is of linear complexity in the structure size, the potentially exponential blowup in the formula size should be tolerable, cf. [LP85]. The complexity is further ameliorated by the use of symbolic model checking techniques in the implementation of the algorithm.

4 Examples

We list a few example specifications which exhibit a pattern typical of real time systems requirements. The requirements are of the general form 'G(antecedent $\Rightarrow$ consequent)' where the antecedent specifies the occurrence of some time bounded condition and the consequent specifies a time bounded extension to the antecedent. In the sequel C represents one time unit.

Example 1. If B occurs exactly two times within five time units, then immediately following the second occurrence of B, D occurs within three time units. $\mathsf{G}(\mathsf{F}^{2B \wedge \preceq 5C} true \Rightarrow \mathsf{F}^{2B \wedge \preceq 5C} \mathsf{F}^{D \wedge \preceq 3C} true)$.

Example 2. If B occurs, then immediately following B, D should occur at least five times within eighteen time units and there should be at least three time units between any two of the five consecutive occurrences of D. $\mathsf{G}((\overline{B}^* B) true \Rightarrow (((\overline{B}^* B)((\overline{D}^* D)(\overline{D + C}^* C)^3)^4 (\overline{D}^* D)) \cap (\overline{C}^* C)^{\preceq 18}) true)$.

Example 3. If the actions B, D, E, F occur, exactly once each and in order, within ten time units, i.e. F occurs before eleven time units have elapsed since the occurrence of B, then G occurs within nine time units of F. Let $\Delta = \overline{B + D + E + F}^*$. $\mathsf{G}(((\Delta\ B\ \Delta\ D\ \Delta\ E\ \Delta\ F) \cap (\overline{C}^* C)^{\preceq 10}) true \Rightarrow ((\Delta\ B\ \Delta\ D\ \Delta\ E\ \Delta\ F) \cap (\overline{C}^* C)^{\preceq 10}) \mathsf{F}^{G \wedge \preceq 9C} true)$.

We have implemented an RTPLTL+ model checking algorithm on top of SMV model checking environment. Model checking RTPLTL+ is accomplished by converting formulae of the logic into their automata and then translating the automata into SMV modules.

We have employed our model checker in solving the Generalized Railroad Crossing problem [HJ93]. The problem is to build a controller which will sense the approach of a train to the railroad crossing and lower a gate across the road preventing road traffic from crossing the tracks.

Correct behavior of the controller can be expressed by two specifications: first, a safety property which guarantees that the gate is down whenever a train is crossing the road ; and second, a liveness property which ensures that if no train is in or approaching the crossing then the gate will be in the upright position. The safety property can be expressed as $G(incrossing \Rightarrow safe)$. $\mathsf{G}(\mathsf{G}^{\preceq 5 clock}(\neg train) \Rightarrow \mathsf{F}^{\preceq 5 clock}((up \mathsf{U} train) \vee \mathsf{G}(up \wedge \neg train)))$ expresses the liveness property.

Using our RTPLTL+ model checking system we were able to verify or find errors in various implementations of the railroad crossing system. For example, if not enough lead time is given to the gate it may not be able to close before a train enters the crossing. The tests conducted were done on an IBM RS6000. Translating the specifications into SMV modules took under a minute. Testing the combined specification and railroad system modules for emptiness also took less than a minute.

5 Summary

We have presented and implemented a general and natural framework for reasoning about quantitative temporal properties. Our models of systems can encode the computations of asynchronous systems using the abstraction of an interleaving syntax. Our logics allow one to reason about properties expressible in PLTL and we have added the ability to discuss regular sequences over paths at a very reasonable cost. Combining the logics with the models allows for the consideration of quantitative properties of independent events. In particular, the RTPLTL+ formula $\mathsf{GF}^{b_1 C_1 \wedge \preceq b_2 C_2} true$ expresses a restriction on the divergence of independent clocks C_1 and C_2. While the syntax for regular formulas is different from, and does not encompass all regular expressions, our techniques are general enough to handle any deterministic finite state machine in place of regular formulae. Model checking RTPLTL+ preserves the utility of PLTL model checking procedures in that the algorithm is linear in the size of the structure.

There has been a great deal of related work in the field and we only mention the work that most closely bears on our own. Alur and Henzinger have written an excellent survey [AH92] which covers many theoretical and practical considerations involved in designing a real time logic.

[AH89][AH94] defines the logic TPTL (Timed Propositional Temporal Logic), which is a real time extension to PLTL. However, unlike TPTL, RTPLTL+ is not restricted to models involving a single time sequence.

Presburger arithmetic is an expressive language for writing quantitative specifications in but has a costly decision procedure. Combining CTL or PLTL with Presburger arithmetic allows the specification of non-regular properties [BE95a] [BE95b], i.e. properties which are not definable as ω-regular sets.

Extended Temporal Logic (ETL) [Wo83] is an extension of PLTL that allows each right linear grammar to define a temporal operator.

Acknowledgments

We would like to thank Insup Lee and Hong-liang Xie for drawing our attention to example specifications similar to the ones in Section 4. We are grateful to Panagiotis Manolios and Kedar Namjoshi for their many insightful comments and questions regarding this work.

References

[AH89] Alur, R., and Henzinger, T. A. , A Really Temporal Logic. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science.* IEEE Computer Society Press, New York, pp. 164-169, 1989.

[AH92] Alur, R. and Henzinger, T. A. , Logics and Models of Real Time: A Survey. In *Real Time: Theory in Practice.* J. W. de Bakker, K. Huizing, W. -P. de Roever, and G. Rozenberg, eds. Lecture Notes in Computer Science, Vol. 600. Springer-Verlag, New York, pp. 74-106, 1982.

[AH94] Alur, R. and Henzinger, T. A. , A Really Temporal Logic. In *Journal of the Association for Computing Machinery.* Vol. 41, No. 1, January 1994, pp. 181-204, 1994.

[Ar94] Arnold, A., Finite Transition Systems : Semantics of Communicating Systems. Translated by John Plaice, Prentice Hall, 1994.

[BE95a] Bouajjani, A., Echahed, R. and Habermehl, P., Verifying Infinite State Processes with Sequential and Parallel Composition. In *ACM POPL95* pp. 95-106.

[BE95b] Bouajjani, A., Echahed, R. and Habermehl, P., On The Verification Problem of Nonregular Properties for Nonregular Processes. In *IEEE LICS95* pp. 123-133.

[Bu62] Buchi, J. R., On a Decision Method in restricted Second Order Arithmetic, Proc. 1960 Inter. Congress on Logic, Methodology, and Philosophy of Science, pp. 1-11.

[CE81] Clarke, E. M., and Emerson, E. A., Design and Verification of Synchronization Skeletons using Branching Time Temporal Logic, Logics of Programs Workshop, IBM Yorktown Heights, New York, Springer LNCS no. 131, pp. 52-71, May 1981.

[Em95] Emerson, E. A., Automated Temporal Reasoning about Reactive Systems. In *Logics for Concurrency*, Faron Moller and Graham Birtwistle, Eds., Springer Verlag, Berlin, 1996, pp. 41-101.

[EMSS90] Emerson, E. A., Mok, A. K., Sistla, A. P., and Srinivasan, J., Quantitative Temporal Reasoning. In *CAV 90: Computer-aided Verification*. E. M. Clarke and R.P. Kurshan Eds. Lecture Notes in Computer Science, Vol. 531. Springer-Verlag, New York, pp. 136-145, 1990.

[ET96] Emerson, E. A. and Trefler, Richard J., Generalized Quantitative Temporal Reasoning. Dept. of Computer Sciences, University of Texas at Austin, technical report TR-96-20, 1996.

[HJ93] Heitmeyer, C. L., Jeffords, R.D., Labaw, B.G., A Benchmark for Comparing Different Approaches for Specifying and Verifying Real-Time Systems. In *Proc., 10th Intern. Workshop on Real-Time Operating Systems and Software*, May, 1993.

[HP85] Harel, D. and Pnueli, A., On the Development of Reactive Systems. In *Logics and Models of Concurrent Systems*. K. Apt Ed. NATO Advanced Summer Institutes, Vol. F-13. Springer-Verlag, pp. 477-498, 1985.

[LP85] Litchtenstein, O., and Pnueli, A., Checking That Finite State Concurrent Programs Satisfy Their Linear Specifications, POPL85, pp. 97-107, Jan. 85.

[Mc92] McMillan, K.L., Symbolic Model Checking: An approach to the state explosion problem. Ph.D. Thesis, Department of Computer Science, Carnegie Mellon University, 1992.

[NP85] Nivat, M., and Perrin, D., Eds. Automata on Infinite Words. Springer-Verlag, Berlin, 1985.

[Pn77] Pnueli, A., The Temporal Logic of Programs, 18th annual IEEE-CS Symp. on Foundations of Computer Science, pp. 46-57, 1977.

[Pn86] Pnueli, A., Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends, in Current Trends in Concurrency: Overviews and Tutorials, ed. J. W. de Bakker, W.P. de Roever, and G. Rozenberg, Springer LNCS no. 224, 1986.

[QS82] Queille, J. P., and Sifakis, J., Specification and verification of concurrent programs in CESAR, Proc. 5th Int. Symp. Prog., Springer LNCS no. 137, pp. 195-220, 1982.

[VW86] Vardi, M., and Wolper, P. , An Automata-theoretic Approach to Automatic Program Verification, Proc. IEEE LICS, pp. 332-344, 1986.

[Wo83] Wolper, P., Temporal Logic Can Be More Expressive *Information and Control*, Vol. 56, 1983, pp. 72-99.