Generalized Quantitative Temporal Reasoning[†]

E. A. Emerson Richard J. Trefler

July 17, 1996

Abstract

Reactive systems form a large and important class computing systems. Temporal logic has proven to be a very expressive formalism for describing such systems. Model checking provides an automated proof technique for ensuring that a reactive system satisfies correctness properties specified in temporal logic. In this paper we discuss real time extensions to certain temporal logics and give model checking algorithms for these extensions.

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.

CTL (Computation Tree Logic) [Em81] [CE81], because it has an efficient model checking algorithm [CE81], is an especially useful kind of temporal logic. Formulae of CTL include the operators 'A,' 'G' and 'F' meaning, respectively, 'for all computations,' 'at all states of a computation' and 'at some point in a computation.' Using these operators CTL and its related logics can express qualitative properties of reactive systems. For example, one can express the requirement that 'every *request* from a client should be met with a *response* from the server' as $\mathsf{AG}(\mathit{request} \Rightarrow \mathsf{AF}\mathit{response})$.

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. RTCTL (Real Time CTL) [EMSS90] is an example of a logic designed to express timing considerations. RTCTL adds to the syntax of CTL, operators like '$\mathsf{F}^{\leq 5}$' which, informally, means 'at some time along a computation before more than five time units have elapsed.' With this formalism we can express properties like 'every *request* from a client should be met with a *response* from the server within five time units' as $\mathsf{AG}(\mathit{request} \Rightarrow \mathsf{AF}^{\leq 5}\mathit{response})$. The surge of interest in real time systems has led to a number of formalisms proposing to deal with the technicalities of real time; many of the these formalisms are quite complex.

In this paper we present a simple but general framework for handling an enriched class of quantitative problems. Our formalism is an extension of RTCTL that employs natural notations from formal language and automata theory. An example of the types of specifications we are interested in, a constraint on the set of computations of a system, is exhibited below.

> If the server ever receives three consecutive *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

[†]This work was supported by NSF grant CCR9415496

fourth *request*. This is expressed as $\mathsf{G}((\overline{request + response}^{*}\, request)^3 true \Rightarrow ((\overline{response}^{*}\, response) \cap (\overline{request}^{*}\, request)^{\leq 3})true)$.

$(\overline{request + response}^{*}\, request)$ is a requirement on strings of system actions that is satisfied when a string contains *request* as the last element of the string and no occurrences of either *request* or *response* anywhere else in the string. $(\overline{request + response}^{*}\, request)^3$ specifies three consecutive occurrences of strings satisfying $(\overline{request + response}^{*}\, request)$, i.e. '*request*' occurs three times and '*response*' has not occurred. *true* is satisfied by any computation therefore the sub-formula $(\overline{request + response}^{*}\, request)^3 true$ is satisfied by any computation with a prefix satisfying $(\overline{request + response}^{*}\, request)^3$. Similarly, '$((\overline{response}^{*}\, response) \cap (\overline{request}^{*}\, request)^{\leq 3})$' specifies that one '*response*' has occurred while less than four '*request*'s have occurred.

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]. 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 efficient extensions of existing model checking algorithms that allow us to model check formulae of our language over general representations of reactive systems.

Section 2, below, gives the syntax and semantics of the quantitative and qualitative languages analyzed in the remainder of the paper. Model checking for the quantitative linear time logic RTPLTL+ is described and analyzed in Section 3. Section 4 contains some interesting specifications and their translations into RTPLTL+. Section 5 is devoted to algorithms for model checking the branching real time logic RTCTL+ and an analysis of these algorithms. Example specifications written in RTCTL+ syntax are given in Section 6. Finally, section 7 is a discussion of related and future work.

2 Preliminaries

2.1 Syntax

Below we present a unified syntax for CTL, Propositional Linear Time Logic (PLTL) [Pn77], CTL* and certain quantitative extensions, viz., RTCTL, RTPLTL, RTCTL+, RTPLTL+ and RTCTL*+.

We use the symbol AP to denote the set of underlying atomic proposition symbols. ACT denotes the 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.

Semantics of these formulae will be presented in terms of a structure $M = (S, R, L)$, where S is a set of states, R is a transition relation on the set of states, and L is a function that labels states and transitions with subsets of AP and elements of ACT respectively. Formulae true or false of states in the structure are denoted 'state' formulae. Formulae true or false of the paths through M are denoted 'path' formulae. 'Regular' formulae are also modeled on paths and describe the actions which occur along the paths.

Let $k \in \mathcal{N}, B \in$ ACT then a term is of the form 'kB', '$\leq kB$' and '$\geq kB$.' A counting expression *ce* is a boolean combination of terms. As a shorthand we will write '$1B$' as B.

The state formulae are defined as follows:

S1. Each atomic proposition P is a formula.

S2. If f and g are state formulae then so are $\neg f$ and $f \wedge g$.

S3a. If ϕ is a path formula then $\mathsf{E}\phi$ is a state formula.

S3b. If f and g are state formulae then so are $\mathsf{EX}f$, $\mathsf{E}(f\mathsf{U}g)$, $\mathsf{AX}f$ and $\mathsf{A}(f\mathsf{U}g)$.

S3c. If f and g are state formulae and *ce* is a counting expression then $\mathsf{E}(f\mathsf{U}^{ce}g)$ and $\mathsf{A}(f\mathsf{U}^{ce}g)$ are state formulae.

S4. If f is a state formula and ρ is a regular formula then $\mathsf{E}\rho f$ and $\mathsf{A}\rho f$ are state formulae.

Path formulae are formed according to the rules:

P1. Each state formula is a path 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 ce is a counting expression then $(\phi \mathsf{U}^{ce} \psi)$ is a path formula.

P4. If ϕ is a path formula and ρ is a regular formula then $\rho\phi$ is a path formula.

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

R1a. $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)$ is a regular formula.

R1b. $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\geq b}$ is a regular formula.

R1c. $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\leq b}$ is a regular formula.

R2. If ρ_1 and ρ_2 are regular formulae then so are $(\rho_1 \rho_2)$ and $(\rho_1 \cap \rho_2)$.

CTL is the language restricted to rules **S1, S2** and **S3b**. PLTL is formed by the rules **S1, P1, P2,** and **P3a**. CTL* is the set of state formulae formed by **S1, S2, S3a, P1, P2** and **P3a**. RTCTL can be formed by adding the rule **S3c** to the rules of CTL and restricting the allowable counting expressions to ones of the form $\leq kC$, $k \geq 1$ (C represents the 'time' unit which is implicit in the RTCTL formulae). RTPLTL adds rule **P3b** to the rules for PLTL. RTPLTL+ adds rule **P4** to RTPLTL and RTCTL adds rule **S4** to RTCTL without any restrictions on counting expressions.

Derived operators are also allowed and we give a listing of them below.

- $f \vee g \;=\; \neg(\neg f \wedge \neg g)$.
- $true \;=\; P \vee \neg P$.
- $false \;=\; \neg true$.
- $f \Rightarrow g \;=\; \neg f \vee g$.
- $\mathsf{A}\psi \;=\; \neg\mathsf{E}\neg\psi$.
- $\phi \mathsf{V} \psi \;=\; \neg(\neg\psi \mathsf{U} \neg\phi)$.
- $\phi \mathsf{V}^{ce} \psi \;=\; \neg(\neg\psi \mathsf{U}^{ce} \neg\phi)$.
- $\mathsf{F}\psi \;=\; true \mathsf{U} \psi$.
- $\mathsf{G}\psi \;=\; \neg\mathsf{F}\neg\psi$.
- $\mathsf{F}^{ce}\phi \;=\; true \mathsf{U}^{ce} \phi$.
- $\mathsf{G}^{ce}\phi \;=\; \neg\mathsf{F}^{ce}\neg\phi$.
- $\overline{\rho}\phi \;=\; \neg\rho(\neg\phi)$.
- $(\backslash(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n) \cap (\overline{C}^* C)^{\leq b})\phi \;=\; \neg((\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n) \cap (\overline{C}^* C)^{\leq b})\neg\phi$

We also use the following shorthand notations. Given $\rho = (\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)$ and $n \geq 1$, then if for all $k, k \in [1:n]$, $\overline{\gamma_1} = \overline{\gamma_k}$ and $B_1 = B_k$ then $(\overline{\gamma_1}^* B_1)^n$ is a shorthand for ρ. Also, given formulae of the form $((\rho_1 \rho_2) \ldots \rho_n)$, if the ρ_i are all identical then we will write $(\rho_1)^n$ as a shorthand for $((\rho_1 \rho_2) \ldots \rho_n)$.

2.2 Intuition

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 $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\leq b}$ do not have a meaning equal to their identical regular expressions. $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\leq b}$ 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 $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\leq 0}$ is true of 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. Similarly $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\geq b}$ requires of a string only that there is a prefix of the string which satisfies $(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^b$.

2.3 Semantics

Temporal logics, such as CTL, are usually interpreted with respect to Kripke structures. 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. In particular, the family of logics discussed here are interpreted over finitely branching structures with finite state sets.

RTCTL implicitly labels each transition with one or more 'clock' actions or time units; the algorithms given in [EMSS90] are restricted to the case when only a single clock action labels each transition but the generalizations are straight forward. RTCTL+ makes no such assumptions. A clock action is merely one of a set of possible actions which may effect state transitions. We require only that all valid paths through the structure be 'clock fair,' i.e. all infinite paths must have infinitely many clock ticks.

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 $L : S \cup R \to 2^{\text{ AP}} \cup \text{ACT}$ 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 $(s, \sigma, s') \in R$, $\mathrm{L}(s, \sigma, s') = \sigma$.

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 $x_i\sigma_i x_{i+1}\sigma_{i+1} \ldots$. $x|\text{ACT}$ denotes the projection of x onto ACT.

Appendix A contains some basic notations and facts about finite and infinite strings and string automata. We note that a structure M can be viewed as a Büchi automaton accepting exactly the strings in $(2^{\text{AP}} \times \text{ACT})^\omega$ which are computations in M.

Given a state s (respectively path x) in M we denote that s (respectively x) satisfies or models state formula f (path formula ϕ) by $M, s \models f$ ($M, x \models \phi$). Similarly s (x) does not satisfy f (ϕ) is denoted by $M, s \not\models f$ ($M, x \not\models \phi$). Extending these notions to counting expression and regular formulae we write, for $\sigma \in \text{ACT}^*$, $\sigma \models ce$ and $\sigma \models \rho$ to denote that a the sequence of actions, σ, satisfies the counting expression ce or, respectively, the regular formula ρ. When M is understood we will sometimes drop it from the $\models$ notation.

The semantics of formulae formed by the syntactic rules defines the satisfaction relation, and is given below.

Given state $s \in S$ and state formulae f, g, g', path formulae ϕ, ψ and regular formula ρ

SS1. $f = P$ for some $P \in \text{AP} : M, s \models f$ iff $f \in L(s)$.

SS2. $f = \neg g : M, s \models f$ iff $M, s \not\models g$. $f = g \wedge g' : M, s \models f$ iff $M, s \models g$ and $M, s \models g'$.

SS3a. $f = \mathsf{E}\phi : M, s \models f$ iff there exists a full path x in M such that $x_0 = s$ and $M, x \models \phi$.

SS3b. $f = \mathsf{EX}g : M, s \models f$ iff there exists full path x in M such that $x_0 = s$ and $M, x^1 \models g$. $f = \mathsf{E}(g\mathsf{U}g') : M, s \models f$ iff there exists full path x in M such that $x_0 = s$ and $M, x \models g\mathsf{U}g'$.

SS3c. $f = \mathsf{E}(g\mathsf{U}^{ce}g') : M, s \models f$ iff there exists full path x in M such that $x_0 = s$ and $M, x \models g\mathsf{U}^{ce}g'$.

SS4. $f = \mathsf{E}\rho g : M, s \models f$ iff there exist a path x such that $x_0 = s$ and $M, x \models \rho g$.

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

PS1. ϕ is a state formula : $M, x \models \phi$ iff $M, x_0 \models \phi$.

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. $\phi = \psi\mathsf{U}^{ce}\psi' : M, x \models \phi$ iff there exist $i \in \mathcal{N}$ such that $\sigma_0 \ldots \sigma_{i-1} \models ce$, $M, x^i \models \psi'$ and for all $j \in [0 : i-1], M, x^j \models \psi$.

PS4. $\phi = \rho\psi : M, x \models \phi$ iff there exists $i \in \mathcal{N}$ such that (i is the least element of $\mathcal{N}$ such that $\sigma = \sigma_0\sigma_1 \ldots \sigma_{i-1}$ and $\sigma \models \rho$) and $M, x^i \models \psi$.

Let $\sigma \in$ ACT*, such that $|\sigma| = m$ and ρ be a regular formula then

RS1a. $\rho = (\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)$: $\sigma \models \rho$ iff there exists $j_1, \ldots, j_n \in [0 : m-1]$ such that for all $i \in [1 : n-1] j_i < j_{i+1}$, $j_n = m-1$, $\sigma_{j_1} = B_1$ and for all $k \in [0 : j_1 - 1] \sigma_k \notin \gamma_1$ and for all $i \in [2 : n], \sigma_{j_i} = B_i$ and for all $k \in [j_{i-1} + 1 : j_i - 1], \sigma_k \notin \gamma_i$.

RS1b. $\rho = (\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\geq b}$: $\sigma \models \rho$ iff $b = 0$ or there exists $i \in [0 : m-1]$ such that $\sigma_0 \ldots \sigma_j \models (\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^b$.

RS1c. $\rho = (\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\leq b}$: $\sigma \models \rho$ iff $\sigma \not\models (\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\geq b+1}$.

RS2. $\rho = (\rho_1\rho_2) : \sigma \models \rho$ iff there is an i, the least element of $[0 : m-1]$ such that $\sigma_0 \ldots \sigma_{i-1} \models \rho_1$, and $\sigma_i \ldots \sigma_{m-1} \models \rho_2$. $\rho = (\rho_1 \cap \rho_2)$: $\sigma \models \rho$ iff $\sigma \models \rho_1$ and $\sigma \models \rho_2$.

Let $\sigma \in$ ACT*, such that $|\sigma| = m$, and ce be a counting expression then

CES1. $ce = kB$: $\sigma \models ce$ iff $k = 0$ and there exists no $j \in [0 : m-1]$ such that $\sigma_j = B$ or $k \neq 0$ and there exists unique $j_1, \ldots, j_k \in [0 : m-1]$ such that $j_k = m-1$, and for all $i \in [1 : k], \sigma_{j_i} = B$ and it is not the case that there exists unique $j_1, \ldots, j_{k+1} \in [0 : m-1]$ such that for all $i \in [1 : k+1], \sigma_{j_i} = B$.

CES2. $ce = \leq kB : \sigma \models ce$ iff it is not the case that there exists unique $j_1, \ldots, j_{k+1} \in [0 : m-1]$ such that for all $i \in [1 : k+1], \sigma_{j_i} = B$.

CES3. If $ce = \geq kB$: $\sigma \models ce$ iff there exists unique $j_1, \ldots, j_k \in [0 : m-1]$ such that for all $i \in [1 : k], \sigma_{j_i} = B$.

CES4. $ce = ce_1 \wedge ce_2$: $\sigma \models ce$ iff $\sigma \models ce_1$ and $\sigma \models ce_2$. $ce = \neg ce_1 : \sigma \models ce$ iff $\sigma \not\models ce_1$.

2.4 Time

RTCTL+ and RTPLTL+ are logics for reasoning about quantitative system properties, including time, therefore it is necessary to make time explicit in our model. Time will be represented by a monotonically increasing sequence of integers. $\tau : (S \times \text{ACT})^\omega \to \mathcal{N}^\omega$ maps the computations of M into 'time sequences.'

Given a full path $x = x_0\sigma_0 x_1\sigma_1 \ldots \in (S \times \mathrm{ACT})^\omega$, and supposing that the clock action is represented by C , then $\tau(x)_0 = 0$ and

$$\tau(x)_{i+1} = \begin{cases} \tau(x)_i + 1 & \text{if } C = \sigma_i \\ \tau(x)_i & \text{otherwise} \end{cases}$$

.

This definition leaves open the possibility that the time sequence is bounded above for any particular computation. We wish to avoid this and require that time increase infinitely often. Let $\Phi = \overset{\infty}{\mathsf{F}}\,(\overline{C}^*C)\,true$, where $\overset{\infty}{\mathsf{F}}$ is a shorthand for GF. Φ is a type of fairness [Fr86] constraint that guarantees that the clock ticks infinitely often. $M, x \models \Phi$ implies the elements of $\tau(x)$ are not bounded above by any integer.

Logics which are at least as expressive as PLTL and RTPLTL can express Φ directly and it is a straightforward matter to incorporate these constraints into the model checking environment. Structure M satisfies RTPLTL formula ϕ under the fairness constraint Φ iff for all full paths x of M, $M, x \models \Phi \Rightarrow \phi$. $\Phi \Rightarrow \phi$ is a formula of RTPLTL+ and hence given a model checking procedure for arbitrary RTPLTL+ formulae we can model check such formulae and ensure correct timing behavior in the model.

Fairness constraints in branching time logics are implemented by evaluating all path quantifiers under the constraint Φ. $M, s \models \mathsf{A}\phi$ under Φ iff for all paths x, such that $x_0 = s$ and $M, x \models \Phi$, $M, x \models \phi$. Similarly, $M, s \models \mathsf{E}\phi$ under Φ iff there is a path x, such that $x_0 = s$, $M, x \models \Phi$ and $M, x \models \phi$. Using CTL* syntax we can write the equivalent formulae $\mathsf{A}(\Phi \Rightarrow \phi)$ and $\mathsf{E}(\Phi \wedge \phi)$ respectively. CTL, however, can not express these constraints. Fairness is incorporated directly into the model checking algorithms for the various modalities. The details of this procedure are given in [EL87] and we will assume them in the algorithms for model checking RTCTL+.

Extending these definitions to multiple clocks is accomplished in a straightforward manner. When the model includes n independent clocks then $\tau : (S \times \mathrm{ACT})^\omega \to (\mathcal{N}^n)^\omega$ and Φ is expanded to include conjuncts for each clock C_i. Notice that there is nothing special about the clock action C. C may represent any action, likewise fairness may be relativized to any subset of system actions not just the clocks.

2.5 Related Concepts

In the sequel we will make use of various manipulations and categorizations of formulae. We mention them here and then will feel free to make use of them without further explanation.

We denote the length of a formula ϕ by $|\phi|$. When ϕ is an atomic proposition then $|\phi| = 1$. Length is defined for boolean combinations of formulae by $|\neg\phi| = 1 + |\phi|$ and when ϕ is a positive boolean combination of ϕ' and ϕ'' then $|\phi| = |\phi'| + |\phi''| + 1$. $|\mathsf{E}\phi| = 1 + |\phi|$ and $|\mathsf{A}\phi| = 1 + |\phi|$. Formulae of the form $\mathsf{X}\phi$ have length equal to $1 + |\phi|$ and formulae of the form $\phi\mathsf{U}\psi$ have length equal to $|\phi| + |\psi| + 1$. The length of a ce formula is defined as follows, terms kB, $\leq kB$ and $\geq kB$ all have length $\log k$. $|\neg ce| = 1 + |ce|$ and $|ce \wedge ce'| = |ce| + |ce'| + 1$. Then $|\phi\mathsf{U}^{ce}\psi| = |\phi| + |\psi| + |ce|$. Regular formulae of type **R1a** have (respectively **R1b, R1c**) have length $n(\max(|\gamma_i|))$, where $|\gamma_i|$ is equal to the number of elements in the set γ_i, $(\log(b)n(\max(|\gamma_i|))$, $\log(b)n(\max(|\gamma_i|)))$. Formulae of the type $(\rho_1\rho_2)$ and $(\rho_1 \cap \rho_2)$ have length $|\rho_1| + |\rho_2| + 1$. Finally, formulae of the form $|\rho\phi| = |\rho| + |\phi| + 1$.

The positive normal form, PNF, of a formula ϕ is a formula ϕ' where negations have been 'pushed' in as far as possible. We use the appropriate short forms, given above, and De Morgan rules to accomplish this. We give the rules for creating the PNF of a formula below and note that it is straightforward to show that for full paths x and states s, $M, x \models \phi$ iff $M, x \models \mathrm{PNF}(\phi)$ and $M, s \models f$ iff $M, s \models \mathrm{PNF}(f)$ for all ϕ and f.

- $\mathrm{PNF}(P) = P$.
- $\mathrm{PNF}(\neg P) = \neg P$.
- $\mathrm{PNF}(f \wedge f') = \mathrm{PNF}(f) \wedge \mathrm{PNF}(f')$.
- $\mathrm{PNF}(\neg(f \wedge f')) = \mathrm{PNF}(\neg f) \vee \mathrm{PNF}(\neg f')$.

- $\mathrm{PNF}(f \vee f') = \mathrm{PNF}(f) \vee \mathrm{PNF}(f')$.
- $\mathrm{PNF}(\neg(f \vee f')) = \mathrm{PNF}(\neg f) \wedge \mathrm{PNF}(\neg f')$.
- $\mathrm{PNF}(\rho\phi) = \rho(\mathrm{PNF}(\phi))$.
- $\mathrm{PNF}(\neg\rho\phi) = \overline{\rho}(\mathrm{PNF}(\neg\phi))$.
- $\mathrm{PNF}(\overline{\rho}\phi) = \overline{\rho}(\mathrm{PNF}(\phi))$.
- $\mathrm{PNF}(\neg\overline{\rho}\phi) = \rho(\mathrm{PNF}(\neg\phi))$.
- $\mathrm{PNF}(\mathsf{X}\phi) = \mathsf{X}(\mathrm{PNF}(\phi))$.
- $\mathrm{PNF}(\neg\mathsf{X}\phi) = \mathsf{X}(\mathrm{PNF}(\neg\phi))$.
- $\mathrm{PNF}(\mathsf{E}\phi) = \mathsf{E}(\mathrm{PNF}(\phi))$.
- $\mathrm{PNF}(\neg\mathsf{E}\phi) = \mathsf{A}(\mathrm{PNF}(\neg\phi))$.
- $\mathrm{PNF}(\mathsf{A}\phi) = \mathsf{A}(\mathrm{PNF}(\phi))$.
- $\mathrm{PNF}(\neg\mathsf{A}\phi) = \mathsf{E}(\mathrm{PNF}(\neg\phi))$.
- $\mathrm{PNF}(\phi\mathsf{U}\phi') = (\mathrm{PNF}(\phi))\mathsf{U}(\mathrm{PNF}(\phi'))$.
- $\mathrm{PNF}(\neg(\phi\mathsf{U}\phi')) = (\mathrm{PNF}(\neg\phi'))\mathsf{V}(\mathrm{PNF}(\neg\phi))$.
- $\mathrm{PNF}(\phi\mathsf{V}\phi') = (\mathrm{PNF}(\phi))\mathsf{V}(\mathrm{PNF}(\phi'))$.
- $\mathrm{PNF}(\neg(\phi\mathsf{V}\phi')) = (\mathrm{PNF}(\neg\phi'))\mathsf{U}(\mathrm{PNF}(\neg\phi))$.
- $\mathrm{PNF}(\phi\mathsf{U}^{ce}\phi') = (\mathrm{PNF}(\phi))\mathsf{U}^{ce}(\mathrm{PNF}(\phi'))$.
- $\mathrm{PNF}(\neg(\phi\mathsf{U}^{ce}\phi')) = (\mathrm{PNF}(\neg\phi'))\mathsf{V}^{ce}(\mathrm{PNF}(\neg\phi))$.
- $\mathrm{PNF}(\phi\mathsf{V}^{ce}\phi') = (\mathrm{PNF}(\phi))\mathsf{V}^{ce}(\mathrm{PNF}(\phi'))$.
- $\mathrm{PNF}(\neg(\phi\mathsf{V}^{ce}\phi')) = (\mathrm{PNF}(\neg\phi'))\mathsf{U}^{ce}(\mathrm{PNF}(\neg\phi))$.

Formulae of temporal logic express conditions on the behavior of their models. Conditions which determine both the present state of the model and its future behavior. Generally, these conditions can be separated into assertions about the present state and assertions about the next state. $\phi\mathsf{U}\phi'$ is true of computations when ϕ' is true at some future time, t, and ϕ is true at all times until t. Stated in terms of present and next time, $\phi\mathsf{U}\phi'$ is true when when either ϕ' is true now or ϕ is true now and $\phi\mathsf{U}\phi'$ is true next time. $\phi\mathsf{U}\phi' = \phi' \vee (\phi \wedge \mathsf{X}(\phi\mathsf{U}\phi'))$, this validity is immediate from the definitions. Model checking procedures use this relationship to attack the problem of satisfaction, since the satisfaction relationship can be very intuitively expressed in terms of now and next time.

However, a problem arises. $\phi\mathsf{U}\phi'$ is a guarantee that ϕ' will eventually be satisfied. $\phi' \vee (\phi \wedge \mathsf{X}(\phi\mathsf{U}\phi'))$ masks that eventuality and seems only to require that the present state satisfy ϕ and $\mathsf{X}(\phi\mathsf{U}\phi')$ a condition on the next state which can again be expanded into a condition on that state and its successor. Repeating this argument *ad infinitum* would seem to indicate that ϕ' need never be true, a contradiction. Furthermore, $\phi\mathsf{U}\phi'$ requires only that ϕ' eventually become true and does not give, *a priori*, a time when it must become true.

A good deal of the complexity of model checking stems from the need to check explicitly for the 'eventual' requirements of temporal formulae. Formulae which express eventual requirements are denoted as 'eventualities.' Eventualities are formulae of the form $\phi\mathsf{U}\phi'$, $\mathsf{E}(f\mathsf{U}g)$, $\mathsf{A}(f\mathsf{U}g)$, $\phi\mathsf{U}^{ce}\phi'$, $\mathsf{E}(f\mathsf{U}^{ce}g)$, $\mathsf{A}(f\mathsf{U}^{ce}g)$, $\rho\phi$, $\mathsf{E}\rho f$ and $\mathsf{A}\rho f$. Notice that $\overline{\rho}\phi$, $\phi\mathsf{V}\phi'$ and their branching time counterparts are not eventualities.

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. The technique consists of creating an automaton, $\mathcal{A}_{\neg\phi}$, on infinite strings 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 $\psi = \rho true$ where $\rho = ((\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n) \cap (\overline{C}^* C)^{\leq b})$. $\mathcal{A}_\rho = (\mathrm{ACT}, \mathcal{Q}, \delta, q_{(0,0)}, F)$ is a Büchi automaton where $\mathcal{Q} = \{q_{(0,0)}, q_{(0,1)} \cdots, q_{(0,b+1)}, q_{(1,0)}, \cdots, q_{(n-1,b+1)}, q_{(n,0)}, \cdots, q_{(n,b)}\}$, $F = \{q_{(n,0)}, \ldots, q_{(n,b)}\}$ and $\delta : \mathcal{Q} \times \mathrm{ACT} \to \mathcal{Q}$ is a deterministic transition relation defined in the transition diagram below. 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.

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*.

Suppose $\psi' = (\setminus(\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n) \cap (\overline{C}^* C)^{\leq b}) false$. $\mathcal{A}_{\overline{\rho}} = (\mathrm{ACT}, \mathcal{Q}, \delta, q_{(0,0)}, F')$ where the acceptance condition $F' = \mathcal{Q} \setminus F$.

Generally, when $\rho = (\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)$ then $\mathcal{A}_\rho = (\mathrm{ACT}, \mathcal{Q}, \delta, q_0, F)$ where $\mathcal{Q} = \{q_0, \ldots, q_{n+1}\}$, $F = \{q_n\}$, and δ is defined as follows. For $i \in [0 : n-1]$, $\delta(q_i, B_{i+1}) = q_{i+1}$, if $B \notin \gamma_i$ then $\delta(q_i, B) = q_i$, and if $B \in \gamma_i$ and $B \neq B_i$ then $\delta(q_i, B) = q_{n+1}$. When $\rho = (\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\geq b}$ and $b \geq 1$ the automaton is similar to the first case except that it is built as if $\rho = \overbrace{\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n \ldots \overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n}^{b}$, $\delta(q_{bn}, B) = q_{bn}$ for all B, and $F = \{q_{bn}\}$. When $b = 0$ then $F = \mathcal{Q} = \{q_0\}$ and $\delta(q_0, B) = q_0$ for all B. When $\rho = (\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\leq b}$ build $\mathcal{A}$ as if the expression were $\rho = (\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n)^{\geq b+1}$ except choose the set of final states as $F = \mathcal{Q} \setminus \{q_{(b+1)n}\}$.

Automata for concatenated regular formulae are similar to the above. If $\mathcal{A}_{\rho_1}$ and $\mathcal{A}_{\rho_2}$ are the automata for $\rho_1 true$ and $\rho_2 true$ respectively, then by replacing the final states of $\mathcal{A}_{\rho_1}$ with the initial state of $\mathcal{A}_{\rho_2}$ we get an automaton for $(\rho_1 \rho_2) true$. Automata for for regular formulae of the form $\rho_1 \cap \rho_2$ are created by forming the product automaton from the automata for ρ_1 and ρ_2.

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 $(x|ACT) \in \mathcal{L}(\mathcal{A}_{\overline{\rho}})$.*

Proof: Let $\sigma = (x|\mathrm{ACT})$. $M, x \models \rho true$ iff there exists $i \in \mathcal{N}$ such that ($\sigma_0 \ldots \sigma_{i-1} \models \rho$ and i is the least such element of $\mathcal{N}$) and $M, x^i \models true$. If x is a computation of M, $M, x^j \models true$ for all $j \in \mathcal{N}$. So $M, x \models \rho true$ iff there exists an i such that $\sigma_0 \ldots \sigma_{i-1} \models \rho$. Suppose there exists such an i, extending δ to a function on strings in the usual way, $\delta(q_{0,0}, \sigma_0 \ldots \sigma_{i-1}) \in q_{\mathrm{f}}$. Once $\mathcal{A}_\rho$ is in a state labeled with f it remains in that state forever. Therefore $\sigma \in \mathcal{L}(\mathcal{A}_\rho)$. Suppose there does not exist an i such that $\sigma_0 \ldots \sigma_{i-1} \models \rho$. By the construction of $\mathcal{A}_\rho$, $\delta(q_{0,0}, \sigma_0 \ldots \sigma_{i-1}) \notin q_{\mathrm{f}}$ for any $i \in \mathcal{N}$ and hence $\sigma \notin \mathcal{L}(\mathcal{A}_\rho)$.

Analysis for the negative case is similar.

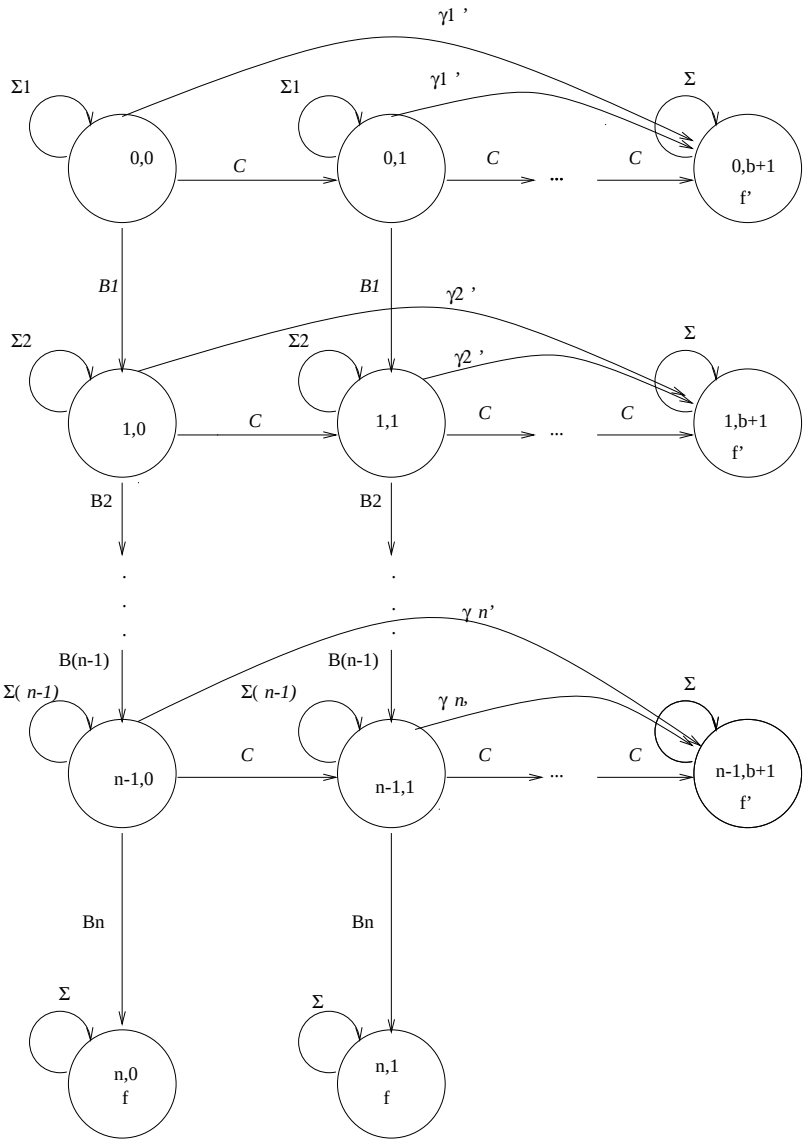

Figure 1: automaton for $((\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n) \cap (\overline{C}^* C)^{\leq b})\, true$

□

Let ce be a counting expression, then there exists a deterministic finite automaton $\mathcal{A}_{ce} = (\text{ACT}, \mathcal{Q}, \delta, q_0, F)$ such that for all $\sigma \in \text{ACT}^*$, $\sigma \in \mathcal{L}(\mathcal{A}_{ce})$ iff $\sigma \models ce$. We construct the automaton inductively from the structure of ce. If $ce = 0B$ then $\mathcal{Q} = \{q_0, q_1\}$, $F = \{q_0\}$ and $\delta(q_0, B) = q_1$, $\delta(q_0, B') = q_0$ for all $B' \neq B$, and $\delta(q_1, B') = q_1$ for all B'. If $ce = kB$ then $\mathcal{Q} = \{q_0, \ldots, q_{k+1}\}$, $F = \{q_k\}$ and δ is defined according to the following rules. For all $i \in [0 : k-1]$, $\delta(q_i, B) = q_{i+1}$ and $\delta(q_i, B') = q_i$ for all $B' \neq B$. $\delta(q_k, B') = q_{k+1}$ and $\delta(q_{k+1}, B') = q_{k+1}$ for all B'. When $ce = \leq kB$ then the automaton is same as the previous one except that $\delta(q_k, B') = q_k$ for all $B' \neq B$ and $F = \{q_0, \ldots, q_k\}$. When $ce = \geq kB$ then the automaton is the same as the one in the case of $ce = kB$ except q_{k+1} is removed from the state set and $\delta(q_k, B') = q_k$ for all B'. When $ce = ce_1 \wedge ce_2$ then $\mathcal{A}_{ce} = \mathcal{A}_{ce_1} \times \mathcal{A}_{ce_2}$. When $ce = \neg ce_1$ then $\mathcal{A}_{ce} = \mathcal{A}_{ce_1}$ except that $F = \mathcal{Q} \setminus F_1$. The following claim follows from the above construction.

Claim 2 *Given a counting expression ce, deterministic automaton $\mathcal{A}_{ce}$ can be constructed in time exponential in $|ce|$ such that $\mathcal{L}(\mathcal{A}_{ce}) = \{\sigma \in ACT^* | \sigma \models ce\}$ and $|\mathcal{A}_{ce}|$ is exponential 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 $1 \ldots a$. Then for $j \in [1 : a]$, $\mathcal{A}_j = (\text{ACT}, \mathcal{Q}_j, \delta_j, q_0^j, F_j)$ 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 $\mathcal{A}_\phi$.

Tableaux are similar to automata with no acceptance conditions. They encode models of a formula by aggregating conditions which the model must satisfy now and conditions which must be satisfied in the next state. This local focus ensures that all paths in the tableaux are safe, i.e. they are propositionally consistent and they do not violate any of their next state requirements, but does not guarantee the satisfaction of eventualities. Therefore a tableaux may encode behavior which is not a model of ϕ. Satisfaction of eventualities, such as $\psi \mathsf{U} \psi'$, may be postponed indefinitely. This postponing behavior is required of the tableaux because there is no *a priori* bound on the point where a computation x finally satisfies ψ', x is only required to satisfy ψ' eventually. Paths in the tableaux that never satisfy ψ' or any other eventuality must be excluded. We accomplish this by using Büchi conditions on the states of the automata which say that if an eventuality is promised infinitely often then it is fulfilled infinitely often.

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 tables characterize the 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 $\phi \mathsf{U}^{q_0^j} \phi'$.
- $\phi \mathsf{U}^{q_i^j} \phi'$, where $q_i^j \notin F_j$: with constituents ϕ and $\mathsf{X}(\phi \mathsf{U}^{q_{h0}^j} \phi') \vee \ldots \vee \mathsf{X}(\phi \mathsf{U}^{q_{hn}^j} \phi')$ where $q_{h0}^j \ldots q_{hn}^j$ are the successor states of q_i^j.
- $\phi \mathsf{V}^{ce} \phi'$, where $\mathcal{A}_j$ is the automaton for ce : with constituent $\phi \mathsf{V}^{q_0^j} \phi'$.

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 $\phi \wedge \mathsf{X}(\phi \mathsf{V} \phi')$.
- $\phi \mathsf{U}^{q_i^j} \phi'$, where $q_i^j \in F_j$: with constituents ϕ' and $\phi \wedge (\mathsf{X}(\phi \mathsf{U}^{q_{h0}^j} \phi') \vee \ldots \vee \mathsf{X}(\phi \mathsf{U}^{q_{hn}^j} \phi'))$ where $q_{h0}^j \ldots q_{hn}^j$ are the successor states of q_i^j.
- $\phi \mathsf{V}^{q_i^j} \phi'$, where $q_i^j \notin F_j$: with constituents ϕ' and $\mathsf{X}(\phi \mathsf{V}^{q_{h0}^j} \phi') \vee \ldots \vee \mathsf{X}(\phi \mathsf{V}^{q_{hn}^j} \phi')$ where $q_{h0}^j \ldots q_{hn}^j$ are the successor states of q_i^j.
- $\phi \mathsf{V}^{q_i^j} \phi'$, where $q_i^j \in F_j$: with constituents $\phi \wedge \phi'$ and $\phi \wedge (\mathsf{X}(\phi \mathsf{V}^{q_{h0}^j} \phi') \vee \ldots \vee \mathsf{X}(\phi \mathsf{V}^{q_{hn}^j} \phi'))$ where $q_{h0}^j \ldots q_{hn}^j$ are the successor states of q_i^j.
- q_i^j, where q_i^j is not labeled with f : with constituents $\mathsf{X}(q_{h0}^j)$, ... , $\mathsf{X}(q_{hn}^j)$ where $q_{h0}^j, \ldots, q_{hn}^j$ are the successor states of q_i^j.

A tableaux is a bipartite graph, where AND-nodes and OR-nodes make up the two sets of nodes. Each node is identified, or labeled, by a set of formulae all of which are true of the node. Creation of the tableaux for ϕ proceeds by labeling the initial OR-node with the formula ϕ. Procedure **OR_Node**, given in figure 2, describes the generation of the set $\mathcal{V}$ of AND-nodes which are children of the OR-Node U. AND-node successors are then generated from the labeling of the AND-node. OR-nodes are so designated because their labels are satisfied when any one of their children are satisfied, AND-nodes are satisfied only when all their children are satisfied.

We say that $M, x \models \mathcal{S}$ for any set of formulae $\mathcal{S}$ iff $M, x \models \psi$ for all $\psi \in \mathcal{S}$.

Suppose V is an AND-node. That is V is a successor of an OR-node and the label of V a set returned in the result of procedure OR-node. V contains a set of propositions, negated propositions and states of the $\mathcal{A}_j$'s. These elements specify the 'state' that V designates. Furthermore, for any full path x of M, $M, x \models U$ if $M, x \models V$. Notice that we have included the conditions that $M, x \models q_i^j$, $M, x \models \phi \mathsf{U}^{q_i^j} \phi'$ and $M, x \models \phi \mathsf{V}^{q_i^j} \phi'$. This lifting of the satisfaction relation comes directly from the structure of the automaton $\mathcal{A}_\rho$ $(\mathcal{A}_{\overline{\rho}}, \mathcal{A}_{ce})$. Referring to the formula and automaton in figure 1, $M, x \models q_{(0,0)}$ iff $M, x \models ((\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n) \cap (\overline{C}^* C)^{\leq b}) true$, $M, x \models q_{(0,1)}$ iff $M, x \models ((\overline{\gamma_1}^* B_1 \ldots \overline{\gamma_n}^* B_n) \cap (\overline{C}^* C)^{\leq b-1}) true$, $M, x \models q_{(1,0)}$ iff $M, x \models ((\overline{\gamma_2}^* B_2 \ldots \overline{\gamma_n}^* B_n) \cap (\overline{C}^* C)^{\leq b}) true$, *etc.* When we transform the tableaux into an automaton which accepts models of ϕ, V specifies acceptance conditions on the models when the automaton is in state V. Furthermore, the next-time formulae of V , formulae whose first element is the X operator, specify the next state relation. The successor of V is the OR-node U whose label contains the formula ψ iff $\mathsf{X}\psi$ is in the label of V. An arc from AND-node V to OR-node U is labeled by the set of $B \in \mathrm{ACT}$ such that the following conditions hold.

- For all $q_i^j \in V$, $q_i^j \notin F_j$ there is a $q_{i'}^j \in U$ such that $\delta_j(q_i^j, B) = q_{i'}^j$.
- For all $q_{i'}^j \in U$ 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 $q_i^j \in V$ and $\delta_j(q_i^j, B) = q_{i'}^j$.

Procedure OR_Node $(U, \mathcal{V})$
/* OR_Node takes as input a set U of formulae and a set $\mathcal{V}$
/* of sets of formulae. If $\mathcal{V}$ is empty then OR_Node will return,
/* in $\mathcal{V}$ the set of AND_node labels that are successors of U.
begin
Repeat Until $U = \emptyset$
Remove ϕ from U;
If $\mathcal{V} = \emptyset$ then $\mathcal{V} := \{\{\phi\}\}$
Else for all $V \in \mathcal{V}, V := V \cup \{\phi\}$;
Case ϕ of the form
$\psi \wedge \psi' : \mathcal{V} := \text{OR\_Node}(\{\psi, \psi'\}, \mathcal{V})$
$\psi \vee \psi' : \mathcal{V} := \text{OR\_Node}(\{\psi\}, \mathcal{V}) \cup \text{OR\_Node}(\{\psi'\}, \mathcal{V})$
$\psi \mathsf{U} \psi' : \mathcal{V} := \text{OR\_Node}(\{\psi \wedge \mathsf{X}(\psi \mathsf{U} \psi')\}, \mathcal{V}) \cup \text{OR\_Node}(\{\psi'\}, \mathcal{V})$
$\psi \mathsf{V} \psi' : \mathcal{V} := \text{OR\_Node}(\{\psi \wedge \mathsf{X}(\psi \mathsf{V} \psi')\}, \mathcal{V}) \cup \text{OR\_Node}(\{\psi \wedge \psi'\}, \mathcal{V})$
$\psi \mathsf{U}^{ce} \psi' : \mathcal{V} := \text{OR\_Node}(\{\psi \mathsf{U}^{q_0^j} \psi'\}, \mathcal{V})$
$\psi \mathsf{U}^{q_i^j} \psi' : \mathcal{V} := \text{OR\_Node}(\{\psi, \mathsf{X}(\psi \mathsf{U}^{q_{h0}^j} \psi')\}, \mathcal{V}) \cup \ldots \cup \text{OR\_Node}(\{\psi, \mathsf{X}(\psi \mathsf{U}^{q_{hn}^j} \psi')\}, \mathcal{V})$
/* if $q_i^j \notin F_j$
$\psi \mathsf{U}^{q_i^j} \psi' : \mathcal{V} := \text{OR\_Node}(\{\psi'\}, \mathcal{V}) \cup$
$\text{OR\_Node}(\{\psi, \mathsf{X}(\psi \mathsf{U}^{q_{h0}^j} \psi')\}, \mathcal{V}) \cup \ldots \cup \text{OR\_Node}(\{\psi, \mathsf{X}(\psi \mathsf{U}^{q_{hn}^j} \psi')\}, \mathcal{V})$
/* if $q_i^j \in F_j$
$\psi \mathsf{V}^{ce} \psi' : \mathcal{V} := \text{OR\_Node}(\{\psi \mathsf{V}^{q_0^j} \psi'\}, \mathcal{V})$
$\psi \mathsf{V}^{q_i^j} \psi' : \mathcal{V} := \text{OR\_Node}(\{\mathsf{X}(\psi \mathsf{V}^{q_{h0}^j} \psi')\}, \mathcal{V}) \cup \ldots \cup$
$\text{OR\_Node}(\{\mathsf{X}(\psi \mathsf{V}^{q_{hn}^j} \psi')\}, \mathcal{V})$
/* if $q_i^j \notin F_j$
$\psi \mathsf{V}^{q_i^j} \psi' : \mathcal{V} := \text{OR\_Node}(\{\psi', \psi\}, \mathcal{V}) \cup \text{OR\_Node}(\{\psi, \mathsf{X}(\psi \mathsf{V}^{q_{h0}^j} \psi')\}, \mathcal{V}) \cup \ldots \cup$
$\text{OR\_Node}(\{\psi, \mathsf{X}(\psi \mathsf{V}^{q_{hn}^j} \psi')\}, \mathcal{V})$
/* if $q_i^j \in F_j$
$\rho\psi : \mathcal{V} := \text{OR\_Node}(\{q_{(0,0)}^j\}, \mathcal{V})$
$\overline{\rho}\psi : \mathcal{V} := \text{OR\_Node}(\{q_{(0,0)}^j\}, \mathcal{V})$
$q_i^j : \mathcal{V} := \text{OR\_Node}(\{\mathsf{X} q_{h0}^j\}, \mathcal{V}) \cup \ldots \cup \text{OR\_Node}(\{\mathsf{X} q_{h0}^j\}, \mathcal{V})$
/* if if $q_i^j \in F_j$
$q_f^j : \mathcal{V} := \text{OR\_Node}(\{\psi\}, \mathcal{V})$;
/* Where $\mathcal{A}_j$ is the automaton for $\rho\psi$ or $\overline{\rho}\psi$.
endCase
endRepeat
Return($\mathcal{V}$);
end

Figure 2: Procedure OR_Node

- For all $\phi \mathsf{U}^{q_i^j} \psi \in V$ either $q_i^j \in F_j$ and $\psi \in V$, or there is an $\phi \mathsf{U}^{q_{i'}^j} \psi \in U$ and $\delta_j(q_i^j, B) = q_{i'}^j$.
- For all $\phi \mathsf{U}^{q_{i'}^j} \psi \in U$ either $i' = 0$ and $\phi \mathsf{U}^{ce} \psi \in U$ and $\mathcal{A}_j$ is the automaton for ce, or there is a $\phi \mathsf{U}^{q_i^j} \psi \in V$ and $\delta_j(q_i^j, B) = q_{i'}^j$.
- For all $\phi \mathsf{V}^{q_i^j} \psi \in V$ then $\psi \in V$ and $\phi \in V$, or $\psi \in V$ and $q_i^j \notin F_j$, or $q_i^j \notin F_j$ and there is a $\phi \mathsf{V}^{q_{i'}^j} \psi \in U$ such that $\delta_j(q_i^j, B) = q_{i'}^j$, or $\phi \in V$ and there is a $\phi \mathsf{V}^{q_{i'}^j} \psi \in U$ such that $\delta_j(q_i^j, B) = q_{i'}^j$.
- For all $\phi \mathsf{V}^{q_{i'}^j} \psi \in U$ either $i' = 0$ and $\phi \mathsf{V}^{ce} \psi \in U$ and $\mathcal{A}_j$ is the automaton for ce, or there is a $\phi \mathsf{V}^{q_i^j} \psi \in V$ and $\delta_j(q_i^j, B) = q_{i'}^j$.

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 $B \in \mathrm{ACT}$ can cause that transition.

By requiring the uniqueness of node labels, we guarantee that the graph is finite, and of size no more than exponential in the length of formula ϕ. We identify similarly labeled OR-nodes by one representative with multiple incoming and outgoing arcs, likewise for AND-nodes.

Once the graph has been completed repeat the following pruning procedure until the graph has stabilized. The procedure ensures that consistency conditions, such as no label requiring both P and $\neg P$, are met.

Repeat the following until the graph stabilizes.

- Remove any AND_node whose successor arc is labeled by the empty set.
- Remove any node which contains ψ and $\neg\psi$, for any ψ.
- Remove any AND_node whose successor has been removed.
- Remove any OR_node all of whose original successors have been removed.
- Delete any AND_node which has an eventuality which is not fulfill-able in the graph. (See below for a more detailed explanation).

Eventualities are formulae of the form $\psi \mathsf{U} \psi'$, $\psi \mathsf{U}^{ce} \psi'$, $\psi \mathsf{U}^{q_i^j} \psi'$, $\rho\psi'$ and q_i^j from positive $\mathcal{A}_j$ representing formulae of the form $\rho\psi'$. They assert that sometime in the future ψ' will become true and the tableaux must be able to fulfill this promise. Whether the tableaux can fulfill this promise is checkable in time linear in the size of the tableaux.

Suppose AND-node V is labeled with eventuality ϕ and $\phi = \psi \mathsf{U} \psi'$. ψ is fulfill-able at V iff there is a path in T from V to AND-node V' which is labeled with ψ'. Further, all AND-nodes on the path, except V' must be labeled with ϕ. Checking for the existence of such a path is tantamount to checking for the existence of a directed acyclic graph (DAG) rooted at V which contains only one successor for each OR-node and whose leaves are labeled with ψ'. Using standard graph theoretic techniques we can check this in linear time [Em95].

Finally we note that even though the eventualities of the AND_Nodes can be fulfilled there is as yet no guarantee that any particular path through the tableaux will fulfill them. Therefore, we view the tableaux as a Büchi automata whose acceptance condition will guarantee that any eventuality that is encountered infinitely often will be satisfied infinitely often.

Number the eventualities in the tableaux 1 through l. Augment the states of the tableaux with a counter from 0 through l. The states of the automaton $\mathcal{A}_\phi$ will have two components, one which respects the states of the tableaux and one which represents the current eventuality of interest. Given a particular eventuality, a state either expresses that the eventuality is pending (not yet satisfied) or is not pending. An eventuality is not pending if it is satisfied in that state or was not pending in any predecessors and is not pending now. Therefore, given a run of $\mathcal{A}_\phi$ in state (t, i), where i represents the ith eventuality, if i is pending in t then

the next state of the run must be some (t', i). If eventuality i is not pending in t then the next state will be $(t', (i+1) mod (l+1))$. F_ϕ the acceptance set of $\mathcal{A}_\phi$, is the set of states where the second component is 0.

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 AND-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 $\neg P \in t, P \notin L(s)$, if U is the child of t in T then t' is a child of U in T, σ is an element of the subset of ACT which labels the arc from t to U 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, $F = \{\mathbf{sink}\} \cup \{\langle t, k\rangle | k = 0\}$.

We can view arbitrary structure M as a Büchi automaton accepting exactly the strings in $(2^{\mathrm{AP}} \times \mathrm{ACT})^\omega$ which are computations in M. In the sequel we shall, given a string or computation $x \in (2^{\mathrm{AP}} \times \mathrm{ACT})^\omega$, write that $x \models \phi$ without specific reference to a structure when there is no particular structure to reference.

Claim 4 *Let $x \in \Sigma^\omega$, $x = \langle s_0, \sigma_0\rangle\langle s_1, \sigma_1\rangle \ldots$, there is an accepting run, $r = \langle t_0, 0\rangle\langle t_1, 1\rangle \ldots$, of $\mathcal{A}_\phi$ on x implies for all $j \in \mathcal{N}$ if $\psi \in t_j$ then $x^j \models \psi$.*

Proof: Suppose for some $j, r_j = \mathbf{sink}$. Then for all $j' \geq j, r_{j'} = \mathbf{sink}$ and therefore for all $j', x^{j'} \models \psi$, for all $\psi \in t_{j'}$.

We can now restrict our attention to the $r_j \neq \mathbf{sink}$.

Suppose $P \in t_j$. r_j has a successor on input x_j iff $P \in s_j$ iff $x^j \models P$. Similarly, if $\neg P \in t_j$, r_j has a successor on input x_j iff $P \notin s_j$ iff $x^j \models \neg P$.

Suppose $\psi \wedge \psi' \in t_j$. By the construction of T, $\psi \wedge \psi' \in t_j$ implies $\psi \in t_j$ and $\psi' \in t_j$. If $x^j \models \psi$ and $x^j \models \psi'$ then $x^j \models \psi \wedge \psi'$. Similarly, if $\psi \vee \psi' \in t_j$ then $\psi \in t_j$ or $\psi' \in t_j$. In the former case, $x^j \models psi$ and in the latter case $x^j \models \psi'$. Therefore $x^j \models \psi \vee \psi'$.

Suppose $\mathsf{X}\psi \in t_j$. By the construction of T, $\psi \in t_{j+1}$. $x^{j+1} \models \psi$ implies $x^j \models \mathsf{X}\psi$.

Suppose $\psi \mathsf{U} \psi' \in t_j$. By the construction of T either $\psi' \in t_j$ or, $\mathsf{X}(\psi \mathsf{U} \psi') \in t_j$ and $\psi \in t_j$. $\psi' \in t_j$ and $x^j \models \psi'$ implies $x^j \models \psi \mathsf{U} \psi'$. Otherwise $\psi, \mathsf{X}(\psi \mathsf{U} \psi') \in t_j$ and because r is an accepting run there is a $j' > j$ such that $\psi \mathsf{U} \psi'$ is not pending at $t_{j'}$, which implies that $\psi' \in t_{j'}$ and therefore that $x^j \models \psi'$. Furthermore, by the construction of $\mathcal{A}_\phi$, $\psi \in t_k$ for all $k \in [j : j' - 1]$ and therefore $x^k \models \psi$ which implies that $x^j \models \psi \mathsf{U} \psi'$.

Suppose $\psi \mathsf{V} \psi' \in t_j$. Either $\psi, \psi' \in t_j$ or $\psi, \mathsf{X}(\psi \mathsf{V} \psi') \in t_j$. If $\psi, \psi' \in t_j$ then $x^j \models \psi$ and $x^j \models \phi$ therefore $x^j \models \psi \mathsf{V} \psi'$. Otherwise, by the construction of $\mathcal{A}_\phi$, either there exists $j' > j$ such that $\psi, \psi' \in t_{j'}$ and for all $k \in [j : j'] \psi \in t_k$, or for all $k \geq j, \psi \in t_k$. In either case $\psi \in t_k$ implies that $x^k \models \psi$. In the latter case this implies that $x^j \models \psi \mathsf{V} \psi'$ and in the former case we have $x^{j'} \models \psi \wedge \psi'$ and therefore $x^j \models \psi \mathsf{V} \psi'$.

Suppose $\rho\psi \in t_j$ and that $\mathcal{A}_\rho$ is the automaton for ρ. We will refer to the states of $\mathcal{A}_\rho$ by q_0, q_1, *etc.* By the construction of $\mathcal{A}_\phi$, $q_0, \mathsf{X} q_h \in t_j$. Furthermore, $q_h \in t_{j+1}$ and $\delta_\rho(q_0, \sigma_j) = q_h$. Since r is an accepting run there is a $j' \geq j$ such that $q_\mathrm{f} \in t_{j'}$ and by the construction of $\mathcal{A}_\phi$, for all $k \in [j : j']$ there is a $q_k \in t_k$ such that if $k < j'$ then $\delta_\rho(q_k, \sigma_k) = q_{k+1}$. By the definition of $\mathcal{A}_\rho$ this implies that $\delta_\rho(q_0, \sigma_j \ldots \sigma_{j'-1}) = q_\mathrm{f}$ and therefore that $x^j|\ \mathrm{ACT}\ \in \mathcal{L}(\mathcal{A}_\rho)$. $q_\mathrm{f} \in t_j$ implies $\psi \in t_j$ and therefore that $x^{j'} \models \psi$. Therefore, we have that $x^j \models \rho\psi$.

Suppose $\overline{\rho}\psi \in t_j$ and that $\mathcal{A}_{\overline{\rho}}$ is the automaton for $\overline{\rho}$. By the construction of $\mathcal{A}_\phi$ either there exist a $j' \geq j$ such that $q_\mathrm{f} \in t_{j'}$ and for all $k \in [j : j']$ there is a $q_k \in t_k$ such that for all $k < j'$ $\delta_{\overline{\rho}}(q_k, \sigma_k) = q_{k+1}$, or for all $k \geq j$ there is a $q_k \in t_k$, $q_k \in F_{\overline{\rho}}$, and $\delta_{\overline{\rho}}(q_k, \sigma_k) = q_{k+1}$. In the former case $x|\ \mathrm{ACT}\ \notin \mathcal{L}(\mathcal{A}_{\overline{\rho}})$ but since $q_\mathrm{f} \in t_{j'}$ then, by the construction of T, $\psi \in t_{j'}$ and $x^{j'} \models \psi$ implies that $x^j \models \overline{\rho}\psi$. In the latter case, since the $q_k \in F_{\overline{\rho}}$, $x|\ \mathrm{ACT}\ \in \mathcal{L}(\mathcal{A}_{\overline{\rho}})$ and therefore $x^j \models \overline{\rho}\psi$.

Suppose $\psi \mathsf{U}^{ce} \psi' \in t_j$ and that $\mathcal{A}_{ce}$ is the automaton for ce. $\psi \mathsf{U}^{ce} \psi' \in t_j$ implies that $\psi \mathsf{U}^{q_0} \psi', \psi \in t_j$. $\psi \in t_j$ implies that $x^j \models \psi$. r is accepting implies that there is a $j' \geq j$ such that $\psi \mathsf{U}^{q_{j'}} \psi' \in t_{j'}$, $\psi' \in t_j$ and

$q_{j'} \in F_{ce}$. Furthermore, by the construction of $\mathcal{A}_\phi$, for all $k \in [j : j']$ there is a $\psi \mathsf{U}^{q_k} \psi' \in t_k$ such that $\delta_{ce}(q_k, \sigma_k) = q_{k+1}$ for $k < j'$ and $q_j = q_0$. This implies that $\sigma_j \ldots \sigma_{j'-1} \in \mathcal{L}(\mathcal{A}_{ce})$. Since $\psi \mathsf{U}^{q_k} \psi' \in t_k$, $k < j'$ then $\psi \in t_k$, by the construction of $\mathcal{A}_\phi$. Therefore $x^k \models \psi$ for $k < j'$ and $x^{j'} \models \psi'$ which implies that $x^j \models \psi \mathsf{U}^{ce} \psi'$.

Suppose $\psi \mathsf{V}^{ce} \psi' \in t_j$ and that $\mathcal{A}_{ce}$ is the automaton for ce. Then $\psi \mathsf{V}^{q_0} \psi' \in t_j$ and by the construction of $\mathcal{A}_\phi$ there is a $j' \geq j$ such that for all $k \in [j : j']$ there is a $\psi \mathsf{V}^{q_k} \psi' \in t_k$ such that for $k < j$ $\delta_{ce}(q_k, \sigma_k) = q_{k+1}$, $q_j = q_0$ and either $\psi, \psi' \in q_{j'}$, or $\psi \in q_{j'}$ and $q_{j'} \notin F_{ce}$. Otherwise, for all $k \geq j$ there is a $\psi \mathsf{V}^{q_k} \psi' \in t_k$ such that $\delta_{ce}(q_k, \sigma_k) = q_{k+1}$ and $q_j = q_0$. In either case if $\psi \mathsf{V}^{q_k} \psi' \in t_k$ then either $\psi \in t_k$ or $q_k \notin F_{ce}$. If $\psi \in t_k$ then $x^k \models \psi$ and if $\psi' \in t_k$ then $x^k \models \psi'$, furthermore, if $q_k \notin F_{ce}$ then $\sigma_j \ldots \sigma_{k-1} \notin \mathcal{L}(\mathcal{A}_{ce})$ and so we have that in either case $x^j \models \psi \mathsf{V}^{ce} \psi'$.

□.

Claim 5 *Let $x \in \Sigma^\omega$, $x = \langle s_0, \sigma_0 \rangle \ldots$, then $x \models \phi$ implies $x \in \mathcal{L}(\mathcal{A}_\phi)$.*

Proof: Suppose $x \models \phi$ then we can construct a satisfying run of $\mathcal{A}_\phi$ on x.

By the construction of the AND-OR graph there is a path, through that graph, $U_0 V_0 U_1 V_1 \ldots$ such that if $P \in V_i$ then $P \in s_i$, if $\neg P \in V_i$ then $P \notin s_i$, and furthermore, there is an arc labeled by σ_i from V_i to U_{i+1}. This path defines a path in $\mathcal{A}_\phi$ $\langle V_0, 0 \rangle \ldots \langle V, 0 \rangle \ldots$ which is accepting. □

□(Theorem 3)

Theorem 6 *$\mathcal{L}(M \times \mathcal{A}_\phi) \neq \emptyset$ 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 linear in the size of the tableaux for formula ϕ. T, the tableaux for ϕ, is at most of size exponential in the length of ϕ since each node has a unique label and there are at most $2^{|\phi|}$ such labels.

Theorem 7 *Given a formula ϕ of RTPLTL+, fairness constraint Φ and structure $M = (S, R, L)$, the model checking problem 'do the fair computations of M satisfy ϕ' is decidable in time $\mathcal{O}(|M| \times 2^{2^{|\Phi|+|\phi|}})$.*

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 of the formula $\neg(\Phi \Rightarrow \phi)$ which is exponential in binary encoding of the numeric constants of Φ and ϕ.

Form the product automaton $M \times \mathcal{A}$, and test this automaton for emptiness. Testing Büchi automaton $\mathcal{A}'$ for emptiness is in $\mathcal{O}(|\mathcal{A}'|)$ (see the appendix for details). 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.

□

4 RTPLTL+ Example Specifications

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 ⇒ consequent)' where the antecedent specifies the occurrence of some time bounded condition and the consequent specifies a time bounded extension to the antecedent.

Example 1. If B occurs exactly four times within ten time units, then immediately following the fourth occurrence of B, D occurs within three time units.

$\mathsf{G}(\mathsf{F}^{4B \wedge \leq 10C} \mathit{true} \Rightarrow \mathsf{F}^{4B \wedge \leq 10C} \mathsf{F}^{D \wedge \leq 3C} \mathit{true})$

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.

$\mathsf{G}((\overline{B}^{\,*}B)\,true \Rightarrow (((\overline{B}^{\,*}B)((\overline{D}^{\,*}D)(\overline{D+C}^{\,*}C)^3)^4(\overline{D}^{\,*}D)) \cap (\overline{C}^{\,*}C)^{\leq 18})\,true)$

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 $\Delta = \overline{B+D+E+F}^{\,*}$ then

$$\begin{array}{ll}\mathsf{G}(& ((\Delta\ B\ \Delta\ D\ \Delta\ E\ \Delta\ F) \cap (\overline{C}^{\,*}C)^{\leq 10})\,true \\ & \Rightarrow \\ & ((\Delta\ B\ \Delta\ D\ \Delta\ E\ \Delta\ F) \cap (\overline{C}^{\,*}C)^{\leq 10}) \\ & \mathsf{F}^{G\wedge \leq 9C}\,true \\ )\end{array}$$

Example 4. If B occurs, then D should occur before F has occurred three times.

$\mathsf{G}((\overline{B}^{\,*}B)\,true \Rightarrow (\overline{B}^{\,*}B)((\overline{D}^{\,*}D) \cap (\overline{F}^{\,*}F)^{\leq 3})\,true)$

Example 5. If B occurs, D, E, F and G should occur, exactly once each and in order, within ten time units and D, E and F should have occurred within five time units. Let $\Delta = \overline{D+E+F+G}^{\,*}$ and $\Delta' = \overline{D+E+F}^{\,*}$, then we have

$$\begin{array}{ll}\mathsf{G}(& \mathsf{F}^B\,true \\ & \Rightarrow \\ & \mathsf{F}^B((\Delta\ D\ \Delta\ E\ \Delta\ F\ \Delta\ G) \cap (\overline{C}^{\,*}C)^{\leq 10})\,true \\ & \wedge \\ & \mathsf{F}^B((\Delta'\ D\ \Delta'\ E\ \Delta'\ F) \cap (\overline{C}^{\,*}C)^{\leq 5})\,true \\ )\end{array}$$

Example 6. If B occurs followed by D and there are no more than two occurrences of E between B and D, then F happens five time units after D.

$$\mathsf{G}(\mathsf{F}^B(\mathsf{F}^{D\wedge \leq 2E}\,true) \Rightarrow (\mathsf{F}^B(\mathsf{F}^{D\wedge \leq 2E}(\mathsf{F}^{F\wedge \leq 5C}\,true))))$$

5 Model Checking RTCTL+

CTL and RTCTL, branching time temporal logics, are of special consideration because their model checking procedures have only linear time complexity. While RTCTL+ model checking is not linear, the exponential cost of model checking is due to the increased expressiveness obtained from the addition of regular formulae and counter expressions. Regular formulae are expensive because the size of the associated automaton is multiplicative in the number of '$\cap$' operators and exponential in the binary encoding of the numeric constants. Similarly, counter expressions are model checked with automata exponential in the binary encoding of constants and multiplicative in the number of '$\wedge$' symbols in the counter expressions. This blow up seems unavoidable and in a manner similar to the Lichtenstein Pnueli [LP85] thesis we argue that the constants and regular expressions will usually be of a manageable size in comparison with the size of the model.

This section covers the development of a model checking algorithm similar to those presented in [EMSS90] and [EL87]. The algorithm runs in time linear in the structure size, linear in the number of temporal connectives of a formula, and exponential in the binary encoding of the constants of a formula and the number of conjunctive arguments to regular formulae and counter expressions.

5.1 Model Checking

Model checking RTCTL+ formulae is very similar to model checking RTCTL formulae with fairness constraints. We give the most important algorithms from [EMSS90] modified to ensure fairness and explain how to use the concepts from the linear time section to handle formulae with regular sub-formulae and counting expressions.

Given RTCTL+ formula f, we denote by $\mathrm{SUB}(f)$ the set of sub-formulae of f; we import the concept of positive and negative formulae from the previous sections.

$\mathrm{SUB}(f)$ is defined recursively as follows:

- $f = P$: for $P \in \mathrm{AP}$ then $\mathrm{SUB}(f) = \{f\}$.
- $f = \neg g$: $\mathrm{SUB}(f) = \{f\} \cup \mathrm{SUB}(g)$.
- $f = f_1 \wedge f_2$: $\mathrm{SUB}(f) = \{f\} \cup \mathrm{SUB}(f_1) \cup \mathrm{SUB}(f_2)$.
- $f = \mathsf{EX} f_1$: $\mathrm{SUB}(f) = \{f\} \cup \mathrm{SUB}(f_1)$.
- $f = \mathsf{E}(f_1 \mathsf{U} f_2)$: $\mathrm{SUB}(f) = \{f\} \cup \mathrm{SUB}(f_1) \cup \mathrm{SUB}(f_2)$.
- $f = \mathsf{E}(f_1 \mathsf{U}^{ce} f_2)$: $\mathrm{SUB}(f) = \{f\} \cup \mathrm{SUB}(f_1) \cup \mathrm{SUB}(f_2)$.
- $f = \mathsf{AX} f_1$: $\mathrm{SUB}(f) = \{f\} \cup \mathrm{SUB}(f_1)$.
- $f = \mathsf{A}(f_1 \mathsf{U} f_2)$: $\mathrm{SUB}(f) = \{f\} \cup \mathrm{SUB}(f_1) \cup \mathrm{SUB}(f_2)$.
- $f = \mathsf{A}(f_1 \mathsf{U}^{ce} f_2)$: $\mathrm{SUB}(f) = \{f\} \cup \mathrm{SUB}(f_1) \cup \mathrm{SUB}(f_2)$.
- $f = \mathsf{E}\rho g$: $\mathrm{SUB}(f) = \{f\} \cup \mathrm{SUB}(g)$.
- $f = \mathsf{E}\overline{\rho} g$: $\mathrm{SUB}(f) = \{f\} \cup \mathrm{SUB}(g)$.
- $f = \mathsf{A}\rho g$: $\mathrm{SUB}(f) = \{f\} \cup \mathrm{SUB}(g)$.
- $f = \mathsf{A}\overline{\rho} g$: $\mathrm{SUB}(f) = \{f\} \cup \mathrm{SUB}(g)$.

Inducting on the number of connectives, $\mathsf{E}, \mathsf{A}, \mathsf{X}, \mathsf{U}, \mathsf{U}^{ce}, \wedge, \neg$, in a formula f, it can be shown that $|\mathrm{SUB}(f)|$ is linear in the number of connectives.

The top level procedure Model_Check, shown in figure 3, takes as input an RTCTL+ formula f, a structure M and fairness constraint Φ and returns structure M' labeled with the sub-formulae of f that respect the structure of M. I.e. if $M = (S, R, L)$ then $M' = (S, R, L')$ and for all $s \in S$, $g \in L'(s)$ iff $M, s \models g$ under Φ, where g or its negation is an element of $\mathrm{SUB}(f)$.

As defined in [EL87] the Fair State Problem is 'given $M = (S, R, L)$ and fairness constraint Φ, determine the states $s \in S$ such that there exists a full path $x = x_0 \sigma_0 \ldots$ in M, $x_0 = s$, and $M, x \models \Phi$.' [EL87] gives an algorithm, which we refer to as FSP, for this problem running in time linear in M and quadratic in the size of Φ. However, when $\Phi = \overset{\infty}{\mathsf{F}} (\overline{C}^* C) true$ a state s is a fair state iff there is a strongly connected component of M, reachable from s, which contains states $s_1, s2$ and $(s_1, C, s_2) \in R$. Which requirement can be checked in time linear in the size of M.

Model_Check uses FSP to label the fair states in S and then proceeds in a bottom up fashion. M is already labeled with propositions and by extension their negations. Once M' has been labeled with all sub-formulae of length $\leq n$, it is a simple matter to extend the labeling to sub-formulae of length $n + 1$. Model_Check handles the obvious cases; a state is labeled by $\neg f$ precisely when it is not labeled by f. Add $f \wedge f'$ to the label of s iff s is already labeled with f and f'. $\mathsf{EX} g$ is added to $L(s)$ just when s has a successor t such that t satisfies g and Φ.

Procedure $\mathsf{Model\_Check}(f, M, \Phi)$
/* Input: structure $M = (S, R, L)$, RTCTL+ formula f
/* and fairness constraint Φ
/* Output: $M' = (S, R, L')$ where $f \in L'(s)$ iff $M, s \models f$.
begin
$S' := \mathsf{FSP}(M, \Phi)$;
for each $s \in S', L(s) := L(s) \cup \{\Phi\}$;
$\mathsf{Sub} := \mathrm{SUB}(f)$
for $i := 1$ to length(f) do
  for each f', in Sub, of length i do
    case structure of f' is of the form
      P : skip; /* M is already labeled with atomic propositions
      $\neg g$: for each $s \in S$, if $g \notin L(s)$ then $L(s) := L(s) \cup \{f'\}$;
      $g_1 \wedge g_2$: for each $s \in S$, if $g_1 \in L(s)$ and $g_2 \in L(s)$ then $L(s) := L(s) \cup \{f'\}$;
      $\mathsf{EX}g$: for each $s \in S$, if there exists $t \in S$ and $B \in ACT$ such that $(s, B, t) \in R$, $g \in L(t)$ and $\Phi \in L(t)$
                then $L(s) := L(s) \cup \{f'\}$;
      $\mathsf{AX}g$: for each $s \in S$, if for all $t \in S$ and $B \in$ ACT such that $(s, B, t) \in R$, $\Phi \in L(t)$ implies $g \in L(t)$,
                then $L(s) := L(s) \cup \{f'\}$;
      $\mathsf{E}(g_1 \mathsf{U} g_2)$: $\mathsf{EU\_Check}(g_1, g_2, f', \Phi)$;
      $\mathsf{A}(g_1 \mathsf{U} g_2)$: for each $s \in S$
                if $g_1 \notin L(s)$ then $L(s) := L(s) \cup \{\neg g_1\}$;
                if $g_2 \notin L(s)$ then $L(s) := L(s) \cup \{\neg g_2\}$;
                if $\neg g_1, \neg g_2 \in L(s)$ then $L(s) := L(s) \cup \{\neg g_1 \wedge \neg g_2\}$;
              end for
              $\mathsf{EU\_Check}(\neg g_2, \neg g_1 \wedge \neg g_2, \mathsf{E}(\neg g_2 \mathsf{U} \neg g_1 \wedge \neg g_2), \Phi)$;
              $\mathsf{EG\_Check}(\neg g_2, \mathsf{EG}(\neg g_2), \Phi)$;
              for all $s \in S$ if neither $\mathsf{E}(\neg g_2 \mathsf{U} \neg g_1 \wedge \neg g_2) \in L(s)$
              nor $\mathsf{EG}(\neg g_2) \in L(s)$
              then $L(s) := L(s) \cup \{f'\}$;
      $\mathsf{E}(g_1 \mathsf{U}^{ce} g_2)$: $\mathsf{Ece\_Check}(M \times \mathcal{A}_{ce}, g_1, g_2, ce, f', \Phi)$;
      $\mathsf{A}(g_1 \mathsf{U}^{ce} g_2)$: $\mathsf{Ace\_Check}(M \times \mathcal{A}_{ce}, g_1, g_2, ce, f', \Phi)$;
      $\mathsf{E}\rho g$: $\mathsf{posE\_Check}(M \times \mathcal{A}_{\rho}, g, f', \Phi)$;
      $\mathsf{A}\rho g$: for all $s \in S$ such that $g \notin L(s)$ then $L(s) := L(s) \cup \{\neg g\}$;
            $\mathsf{negE\_Check}(M \times \mathcal{A}_{\overline{\rho}}, \neg g, \mathsf{E}\overline{\rho}\neg g, \Phi)$;
            for $s \in S$ such that $\mathsf{E}\overline{\rho}\neg g \notin s$
              $L(s) := L(s) \cup \{f'\}$;
      $\mathsf{E}\overline{\rho} g$: $\mathsf{negE\_Check}(M \times \mathcal{A}_{\overline{\rho}}, g, f', \Phi)$;
      $\mathsf{A}\overline{\rho} g$: for all $s \in S$ such that $g \notin L(s)$ $L(s) := L(s) \cup \{\neg g\}$;
            $\mathsf{E\_Check}(M \times \mathcal{A}_{\rho}, \neg g, \mathsf{E}\rho\neg g, \Phi)$;
            for $s \in S$ such that $\mathsf{E}\rho\neg g \notin s$
              $L(s) := L(s) \cup \{f'\}$;
    endcase
  endfor
endfor
end

Figure 3: Procedure $\mathsf{Model\_Check}$

```
Procedure EU_Check(g_1, g_2, f, Φ)
begin
      EU := ∅;
      for each s ∈ S do
          if g_2 ∈ L(s) and Φ ∈ L(s) then
            EU := EU ∪ {s};
            L(s) := L(s) ∪ {f};
          endif
      endfor
      while EU ≠ ∅
            remove s from EU;
            PRE := {s'|(s', B, s) ∈ R, g_1 ∈ L(s') and f ∉ L(s')};
            for each s' ∈ PRE do L(s') := L(s') ∪ {f};
            EU := EU ∪ PRE;
      endwhile
end
```

Figure 4: Procedure EU_Check

Figure 4 contains the procedure EU_Check given in [EL87]. EU_Check labels each state s in M with $\mathsf{E}(g_1 \mathsf{U} g_2)$ exactly when M contains a fair path $x = x_0\sigma_0 \ldots$, $x_0 = s$ and $M, x \models g \mathsf{U} g'$. Each state s which satisfies Φ and g' satisfies $\mathsf{E}(g \mathsf{U} g')$. EU_Check finds all such states and 'works backwards' finding all states which satisfy g and are connected to states already known to satisfy $\mathsf{E}(g \mathsf{U} g')$ under Φ. This set of predecessor states also satisfies $\mathsf{E}(g \mathsf{U} g')$ under Φ. Eventually all states reachable, in reverse, from the initial set of satisfying states are examined and the algorithm traverses each edge in the graph at most once. Therefore EU_Check runs in time linear in the size of the structure.

Claim 8 *Given $M = (S, R, L)$, $f = \mathsf{E}(g_1 \mathsf{U} g_2)$, and Φ, such that for all $s \in S$, $M, s \models g_1$ (respectively g_2, Φ) iff $g_1(g_2, \Phi) \in L(s)$ then EU_Check adds f to $L(s)$ exactly when $M, s \models f$.*

Figure 5 contains procedure EG_Check which labels states in M with formulae of the form $\mathsf{EG}g$. EG_Check reduces M to a sub-graph which contains only those states which satisfy g and then uses the FSP to find the states with a fair path which satisfies f. Any state which satisfies $\mathsf{EG}g$ under Φ must have have a computation, starting in that state, which satisfies $(\mathsf{G}g) \wedge \Phi$. Just those states in the sub-graph with infinite computations satisfy $\mathsf{EG}g$ under Φ.

Claim 9 *Given $M = (S, R, L), f = \mathsf{G}g$ and Φ, such that for all $s \in S$, $M, s \models g$ (respectively Φ) iff $g(\Phi) \in L(s)$ then EG_Check adds f to the label of state s precisely when $M, s \models f$.*

Since $\mathsf{A}(f_1 \mathsf{U} f_2) = \neg(\mathsf{E}(\neg f_2 \mathsf{U} \neg f_1 \wedge \neg f_2) \vee \mathsf{EG}(\neg f_2))$ we have as a corollary that Model_Check correctly labels states that satisfy $\mathsf{A}(f_1 \mathsf{U} f_2)$.

Procedure pos_Echeck, in figure 6 below, gives an algorithm for determining the states of M that model $\mathsf{E}\rho g$ under fairness constraint Φ , where it is has previously been determined which states model g and which states model Φ. Note that the correctness of the algorithm is based on the fact that M can be viewed as a Büchi automaton accepting only those strings which are computations of M.

Suppose f is of the form $\mathsf{E}\rho g$. Then for all $s \in S$, $M, s \models f$ under fairness constraint Φ iff there exists a full path $x = x_0\sigma_0 \ldots$ in M, such that $x_0 = s$ and $M, x \models (\rho g) \wedge \Phi$. When determining whether a path satisfies a fairness constraint like $\overset{\infty}{\mathsf{F}}\, \phi$ one can ignore arbitrarily long finite prefixes of the path. Therefore it suffices

Procedure EG_Check(g, f, Φ)
begin
    $S' := \{s | g \in L(s)\}$;
    $R' := \{(s, B, s') | (s, B, s') \in R \text{ and } s, s' \in S\}$;
    $M' := (S', R', L)$;
    $S_1 := \mathsf{FSP}(M, \Phi)$
    for all $s \in S_1$
        $L(s) := L(s) \cup f$;
    endfor
end

Figure 5: Procedure EG_Check

Procedure pos_Echeck$(M \times \mathcal{A}_\rho, g, \Phi, f)$
/* $M \times \mathcal{A}_\rho$ is the product automaton */
/* described in the text */
begin
    SCC := the set of strongly connected components of $Q_{M \times \mathcal{A}_\rho}$;
    GSC := the elements of SCC whose intersection with $F_{M \times \mathcal{A}_\rho}$ is non-empty;
    GS := $\{s \in S|$ there is a path from $\langle q_{0,0}, s\rangle$to a state in an element of GSC $\}$;
    for all $s \in GS$, $L(s) := L(s) \cup \{f\}$;
end

Figure 6: Procedure pos_Echeck

to check $M, x \models \rho(g \wedge \Phi)$. x may not be a fair path, however, $M, x \models \rho(g \wedge \Phi)$ guarantees a state x_i such that $M, x_i \models g \wedge \Phi$ which implies that there is a fair full path $y = y_0\sigma_0' \ldots$, such that $y_0 = x_i$ and the path $x' = x_0\sigma_0 \ldots x_{i-1}\sigma_{i-1}y_0\sigma_0' \ldots$ is a fair path which satisfies $(\rho g) \wedge \Phi$.

Given regular formula ρ the automaton $\mathcal{A}_\rho = (\text{ACT}, \mathcal{Q}_\rho, \delta_\rho, q_0, F_\rho)$ accepts strings $\sigma \in \text{ACT}^\omega$ which satisfy ρ. Define the product automaton $M \times \mathcal{A}_\rho = (\Sigma, \mathcal{Q}, \delta, \langle s, q_0\rangle, F)$ as follows. $\Sigma = S \times$ ACT, $\mathcal{Q} = S \times (\mathcal{Q}_\rho \cup \{q_T, q_F\})$ and $F = \{\langle s, q_T\rangle | s \in S\}$. $\delta : \mathcal{Q} \times \Sigma \to 2^{\mathcal{Q}}$ defined by the following rules. For $q \in \mathcal{Q}_\rho, q \notin F_\rho$, $\langle q', s'\rangle \in \delta(\langle q, s\rangle, \langle t, B\rangle)$ iff $t = s$, $(s, B, s') \in R$, and $\delta_\rho(q, B) = q'$. For $q \in \mathcal{Q}_\rho, q \in F_\rho$ $\langle q_T, s'\rangle \in \delta(\langle q, s\rangle, \langle t, B\rangle)$ iff $t = s$, $(s, B, s') \in R$, $\Phi \in L(s)$, and $g \in L(s)$; if $\Phi \notin L(s)$ or $g \notin L(s)$ then $\langle q_F, s'\rangle \in \delta(\langle q, s\rangle, \langle t, B\rangle)$. If $q \in \{q_T, q_F\}$ then $\langle q, s'\rangle \in \delta(\langle q, s\rangle, \langle t, B\rangle)$ iff $s = t$ and $(s, B, s') \in R$.

$L(M \times \mathcal{A}_\rho) = \emptyset$ iff there are no computations, x of M such that $M, x \models (\rho g) \wedge \Phi$. pos_ECheck uses this fact to find all the states $s \in S$ which satisfy $\mathsf{E}(\rho g) \wedge \Phi$ by a standard Büchi automata emptiness algorithm.

The product construction given in pos_ECheck is linear in the size of $|\mathcal{A}_\rho|$ and $|M|$. Clearly the final product automaton is still Büchi and so it can be tested for emptiness in time linear in its size.

Claim 10 *Given input formula* $\mathsf{E}\rho g$*, fairness constraint* Φ*, and product automaton* $M \times \mathcal{A}_\rho$*, procedure* pos_ECheck *correctly labels the states* s *of* M *with* $\mathsf{E}\rho g$ *such that* $M, s \models \mathsf{E}\rho g$ *under fairness constraint* Φ*, in time linear in the size of* $\mathsf{E}\rho g$ *and* M*.*

Formulae of the form $\mathsf{E}\overline{\rho} g$ are handled in a similar fashion, see figure 7, by changing the acceptance set of the automaton $M \times \mathcal{A}_{\overline{\rho}}$ to include all states of $\mathcal{Q}_{\overline{\rho}}$ except q_{F} and checking that the fairness condition holds in the strongly connected components of the automaton structure.

Procedure neg_Echeck$(M \times \mathcal{A}_{overline\rho}, g, \Phi, f)$
/* $M \times \mathcal{A}_\rho$ is the product automaton */
/* described in the text */
begin
SCC := the set of strongly connected components of $Q_{M\times\mathcal{A}_{\overline{\rho}}}$;
GSC := the elements of SCC whose intersection with $F_{M\times\mathcal{A}_\rho}$ is non-empty
and which contain a transition of the form $(\langle s,q\rangle, C, \langle t,q\rangle)$
/* i.e. the SCC which are also fair according to $\overset{\infty}{\mathsf{F}}\ C$. */
GS := $\{s \in S|$ there is a path from $\langle q_{0,0}, s\rangle$to a state in an element of GSC $\}$;
for all $s \in GS$, $L(s) := L(s) \cup \{f\}$;
end

Figure 7: Procedure neg_Echeck

Claim 11 *Given input formula* $\mathsf{E}\overline{\rho}g$*, fairness constraint* Φ*, and product automaton* $M \times \mathcal{A}_{\overline{\rho}}$ *procedure* neg_ECheck *correctly labels the states* s *of* M *with* $\mathsf{E}\overline{\rho}g$ *such that* $M, s \models \mathsf{E}\overline{\rho}g$ *under fairness constraint* Φ*, in time linear in the size of* $\mathsf{E}\overline{\rho}g$ *and* M*.*

Note that $\mathsf{A}\rho g = \neg\mathsf{E}\overline{\rho}(\neg g)$ and $\mathsf{A}\overline{\rho}g = \neg\mathsf{E}\rho(\neg g)$ we can label the states of the figure which satisfy these types of formulae.

Ece_Check takes as input a formula f' of the form $\mathsf{E}(g\mathsf{U}^{ce}g')$ and labels with f' exactly those states of S which satisfy f' under the constraint Φ. Assuming that M has already been labeled with g, g' and Φ the procedure works in the following manner. Let $\mathcal{A}_{ce} = (\mathrm{ACT}, \mathcal{Q}, \delta, q_0, F)$ be the automaton for the counting expression ce then create $M \times \mathcal{A}_{ce} = (\mathrm{ACT}, \mathcal{Q}', \delta', \mathcal{Q}'_0, F')$. $\mathcal{Q}' = (S \times \mathcal{Q}') \cup \{sink\}$, $\mathcal{Q}'_0$, the set of $(q_0, s) \in \mathcal{Q}'$, is the set of start states and $F' = \{sink\}$. δ' is defined by $(s', q') \in \delta'(s, q, B)$ iff $(s, B, s') \in R$, $\delta(q, B) = q'$, $g \in L(s)$ and one of $q \notin F$, $g' \notin L(s)$, or $\Phi \notin L(s)$. $\delta'(s, q, B) = sink$ if $q \in F$, $g' \in L(s)$ and $\Phi \in L(s)$; and $\delta'(sink, B) = sink$.

Clearly $\mathcal{L}(M \times \mathcal{A}_{ce}) = \emptyset$ iff there exists no $s \in S$ such that $M, x \models f'$. Using techniques outlined in the procedure pos_Echeck we can find all the states which satisfy f' and label them appropriately.

Claim 12 *Given RTCTL+ formula* $\mathsf{E}(f\mathsf{U}^{ce}g)$*, fairness constraint* Φ *and the automaton for the counting expression ce,* $\mathcal{A}$*, and assuming that* M *has been labeled with* Φ, f *and* g*, then Procedure* Ece_Check *labels the states,* $s \in S$ *with* $\mathsf{E}(f\mathsf{U}^{ce}g)$ *iff* $M, s \models \mathsf{E}(f\mathsf{U}^{ce}g)$ *under* Φ*. Furthermore, the procedure runs in time linear in the size of* M *and linear in the size of* $\mathcal{A}$*.*

Ace_Check works in a similar way. Given the input formula $\mathsf{A}(f\mathsf{U}^{ce}g)$, fairness constraint Φ and the automaton for ce $\mathcal{A}' = (\mathrm{ACT}, \mathcal{Q}', \delta', q_0, F')$ then the first step is to create a product automaton $\mathcal{A}$ that accepts computations of M which satisfy the property $\neg(f\mathsf{U}^{ce}g) \wedge \Phi$. $\mathcal{A} = (S \times \mathrm{ACT}, \mathcal{Q}, \delta, \mathcal{Q}_0, F)$ where $\mathcal{Q} = (S \times \mathcal{Q}') \cup \{sink\}$, $\mathcal{Q}_0 = \{(s, q) | q = q_0\}$ and $F = \{sink\} \cup \{(s, q) | q \notin F'$ or $g \notin L(s)\}$. δ is defined as follows, $\delta(sink, \langle s, B\rangle) = sink$, $(s', q') \in \delta(\langle s, q\rangle, \langle t, B\rangle)$ iff $s = t, (s, B, s') \in R, (q, B, q') \in \delta', f \in L(s)$, and $(q \notin F'$ or $g \notin L(s))$, and $\delta(\langle s, q\rangle, \langle t, B\rangle) = sink$ iff $\Phi \in L(s), f \notin L(s)$ and $(q \notin F'$ or $g \notin L(s))$. As in the procedure neg_ECheck we need only check for fair strongly connected components whose intersection with the F' is none empty and which contain an arc labeled by C.

Claim 13 *Given RTCTL+ formula* $\mathsf{A}(f\mathsf{U}^{ce}g)$*, fairness constraint* Φ *and the automaton for the counting expression ce,* $\mathcal{A}'$*, and assuming that* M *has been labeled with* Φ, f *and* g*, then Procedure* Ace_Check *labels the states,* $s \in S$ *with* $\mathsf{A}(f\mathsf{U}^{ce}g)$ *iff* $M, s \models \mathsf{A}(f\mathsf{U}^{ce}g)$ *under* Φ*. Furthermore, the procedure runs in time linear in the size of* M *and linear in the size of* $\mathcal{A}$*.*

Proof: $M, x \models \mathsf{A}(f \mathsf{U}^{ce} g)$ under Φ iff $M, x \models \neg \mathsf{E} \neg (f \mathsf{U}^{ce} g)$ under Φ. Since a state, s, satisfies formula $\neg f'$ iff it is not the case that s satisfies f' it suffices to determine which states satisfy $\mathsf{E} \neg (f \mathsf{U}^{ce} g)$ under Φ. $M, s \models \mathsf{E} \neg (f \mathsf{U}^{ce} g)$ under Φ iff there exists path $x = x_0 \sigma_0 \ldots$ such that $x_0 = s$ and $M, x \models \neg (f \mathsf{U}^{ce} g) \wedge \Phi$ iff it is not the case that there exists an $i \in \mathcal{N}$ such that $\sigma_0 \sigma_1 \ldots \sigma_{i-1} \models ce$ and $M, x^i \models g$ and for all $j < i, M, x^j \models f$ and $M, x \models \Phi$. Suppose there exists $k \in \mathcal{N}$ such that $M, x^k \models \neg f$, this implies that if i exists, as above, then $i \leq k$. I.e. given that for no $i \leq k$, $M, x^i \models g$ and for all $j < i, M, x^j \models f$ and $\sigma_0 \sigma_1 \ldots \sigma_{i-1} \models ce$ then there exists no such $i > k$.

Let x be a path in M such that $M, x \models \neg (f \mathsf{U}^{ce} g) \wedge \Phi$. Then there exists an accepting run, r, of $\mathcal{A}$ on x. Suppose there exists a k as described above. Then $r = (x_0, q_0), (x_1, q_1) \ldots (x_k, q_k) sink \ldots$ which is an accepting run. If no k exists then the run $(x_0, q_0)(x_1, q_1) \ldots$ is accepting since for each (x_i, q_i) either $g \notin L(x_i)$ or $q_i \notin F'$, and $\sigma_i = C$ infinitely often. Suppose $\mathcal{A}$ has an accepting run. By the transition relation of $\mathcal{A}$, if *sink* is not an element of the run, r, then the input generating the run is a computation of M which satisfies $\neg (f \mathsf{U}^{ce} g) \wedge \Phi$. No prefix $(x_0, \sigma_0)(x_1, \sigma_1) \ldots (x_i, \sigma_i)$ can satisfy both $\sigma_0 \ldots \sigma_{i-1} \models ce$ and $M, x_i \models g$ since that would imply that $r_i = (x_i, q)$, $q \in F'$ and $g \in L(x_i)$, and $\mathcal{A}$ has no successors from such a state. If *sink* is an element of the run then the input prefix leading up to *sink* can be used to construct a computation satisfying $\neg (f \mathsf{U}^{ce} g) \wedge \Phi$ because x_i of (x_i, q), the predecessor of the first state *sink* state, satisfies Φ and $\neg f$.

□

Therefore we have the following theorem.

Theorem 14 *Given structure $M = (S, R, L)$, fairness constraint Φ and RTCTL+ formula f, the question 'does there exist a state $s \in S$ such that $M, s \models f$ under Φ' can be answered in time $\mathcal{O}(|M| \times 2^{|f|} \times |\Phi|)$.*

6 RTCTL+ Example Specifications

In this section we reformulate the example specifications in the logic RTCTL+.

Example 1. If B occurs exactly four times within ten time units, then immediately following the fourth occurrence of B, D occurs within three time units.

$\mathsf{AG}(\neg \mathsf{E}((\overline{B}^* B)^4 \cap (\overline{C}^* C)^{\leq 10}) \neg \mathsf{A}((\overline{D}^* D) \cap (\overline{C}^* C)^{\leq 3}) true)$

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 consecutive occurrences of D.

$$\mathsf{AG}\Big(\quad \mathsf{AG}^B \; \mathsf{A}(((\overline{D}^* D (\overline{D + C}^* C)^3)^4 \overline{D}^* D) \cap (\overline{C}^* C)^{\leq 18}) true \Big)$$

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 $\Delta = \overline{B + D + E + F}^*$ then

$\mathsf{AG}(\neg \mathsf{E}((\Delta\, B\, \Delta\, D\, \Delta\, E\, \Delta\, F) \cap (\overline{C}^* C)^{\leq 10}) \mathsf{EG}^{G \wedge \leq 9 C} false)$

Example 4. If B occurs then D should occur before F has occurred three times.

$\mathsf{AG}(\neg \mathsf{E}(\overline{B}^* B) \neg \mathsf{A}((\overline{D}^* D) \cap (\overline{F}^* F)^{\leq 3}) true)$

Example 5. If B occurs, D, E, F and G should occur, exactly once each and in order, within ten time units and D, E and F should have occurred within five time units. Let $\Delta = \overline{D + E + F + G}^*$ and $\Delta' = \overline{D + E + F}^*$, then we have

$$\mathsf{AG}\Big(\quad (\mathsf{AG}^{B}\mathsf{A}((\Delta\, D \Delta\, E \Delta\, F\, \Delta\, G) \cap (\overline{C}^{*} C)^{\leq 10}) true) \wedge (\mathsf{AG}^{B}\mathsf{A}((\Delta' D\, \Delta' E\, \Delta' F) \cap (\overline{C}^{*} C)^{\leq 5}) true) \Big)$$

Example 6. If B occurs followed by D and there are no more than two occurrences of E between B and D, then F happens five time units after D.

$$\mathsf{AG}(\mathsf{AG}^{B}\mathsf{AG}^{D \wedge \leq 2 E}\mathsf{AF}^{F \wedge \leq 5 C} true)$$

7 Conclusion and Related Work

In summary, we have presented 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 CTL, RTCTL and PTL 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 $\mathsf{GF}^{b_1 C_1 \wedge \leq b_2 C_2} true$ 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 RTCTL+ and RTPLTL+ preserves the utility of CTL and PLTL model checking procedures in that the algorithms are linear in the size of the structure. RTCTL model checking techniques which are linear in the log of the size of the formula constants cannot be applied, however, the models considered here are more general and the logics more generally quantitative rather than simply timing related.

There has been a great deal of related work in the field and we only mention some of the work that most closely bears on our own. Alur and Henzinger have written an excellent survey [AH92] which covers many of the basic 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. TPTL is an 'half-order' logic in that formulae such as $\Box x.(B \Rightarrow \Diamond y.(D \wedge y \leq x + 10))$ partially quantify over 'time.' Evaluation of a formula over a sequence of states and time value pairs (the sequence of time values is monotonically non-decreasing) proceeds by 'freezing' x to the value of time at the first state of the sequence. Therefore we can loosely interpret the above formula as meaning whenever B is true at state s, bind x to the value of time associated with s, and verify that there is a future state s' at which D is true and time, represented by y has not increased more than 10 units from x. [AH90] explains that x and y must refer to the same increasing sequence of time values, otherwise the logic may become undecidable. RTPLTL+ can express some of TPTL properties, e.g. the above property is expressed in RTPLTL+ as $\mathsf{G}(\mathsf{F}^{B} true \Rightarrow \mathsf{F}^{B}\mathsf{F}^{D \wedge \leq 10 C} true)$. However, RTPLTL+ is not restricted to models involving a single time sequence. TCTL [ACD90] [Al91] is the branching time analog of TPTL with a continuous time semantics.

Lewis, in [Le90], describes an extension to CTL which incorporates time bounds on the basic modalities. $\mathsf{E}(P \mathsf{U}_{\tau} Q)$ specifies that there is a computation along which P holds until Q holds and moreover Q holds within the bounds specified by τ, a contiguous subset of a discrete domain. Formulae of the logic can be encoded in RTCTL+ with a linear cost [EMSS90].

[Le90] uses, as an example of the utility of the logic, the problem of modeling asynchronous circuits with general delay assumptions. A significant drawback to this approach arises because the timing delays in circuit transitions may cause the circuit to oscillate before stabilizing. Models which encode this oscillating behavior may encode infinite computations which oscillate between unstable states. Therefore there can exist formulae of the logic which are true (false) of the model and false (true) of the circuit. [Le90] offers

a solution which incorporates timing histories of the transitions. Using the histories a model can guarantee that any such oscillating behavior is bounded by the delays on the actual gates of the circuit, or estimations of those delays. This extra data encoded in the model can, however, cause the size of the model to increase dramatically. An alternative solution is to mark those states that are stable with a proposition and rule out, through the use of fairness restrictions, any infinite behavior which does not stabilize infinitely often.

Automatic quantitative analysis of programs or hardware can be, in general, a difficult problem. 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. However, Bouajjani et. al. [BE95a] have been able to combine CTL with Presburger arithmetic, in PCTL, and model check these formulae on some finite and infinite state processes. While PCTL is more expressive than RTCTL+ it is also more costly. Model checking PCTL is undecidable on finite state processes in general, however, for a sub-language of PCTL model checking is decidable for a restricted class of finite and infinite state processes called 'guarded PA's' [BE95a].

CLTL [BE95b] is a quantitative logic that extends PLTL with Presburger arithmetic and finite state automata. Again, model checking the full logic on finite state processes is not decidable, but certain sub-logics have decidable model checking procedures. ALTL, which adds finite automata to PLTL, has a model checking procedure for finite state structures and some infinite state structures. ALTL is not, however, an obviously quantitative logic and is similar to Extended Temporal Logic (ETL) [Wo83].

ETL is an extension of PLTL that allows each right linear grammar to define a temporal operator. [Wo83] shows that this logic is more expressive that PLTL and has a decision procedure similar to and with the same complexity as PLTL. Alur and Henzinger, in [AH90], extend TPTL to TETL and show that the more expressive logic TETL can be used at no extra cost in terms of model checking procedures.

There has also been work done on probabilistic model checking [HJ89], [Ha91], [AS95] and [BdA95]. This paradigm allows one to ask what percentage of the computations of a system satisfy certain CTL or PLTL properties.

Our approach here has been to examine structures with respect to the quantitative specifications of a logic. Alternatively, specifications maybe directly embodied in automata. [Di88a] [Di88b] is a restricted example of this method and [Bur88] is an extension. Representing specifications as automata requires the specifier to make a difficult jump from the pre-formal specification to the extensional automaton. A difficulty not encountered here although the model checking method for PLTL and its real time extensions can be seen as a compiling of language specifications into automata specifications.

Henriksen et al, in [HJ95], study specifications written in a second order monadic logic of traces. The focus is on an alternative to regular expressions and they give a method for compiling specifications into reduced finite state automata.

Quantitative model checking answers the question 'do the computations of a structure satisfy a given specification?' Where the specification encodes some quantitative requirement such as 'every *request* is *responded* to within 5 time units.' Another analytic paradigm may ask what is the longest or shortest *response* time to a *request*? Campos et. al. [CE94] have developed algorithms that analyze finite state systems and answer such questions. Their algorithms find longest (provided there are no loops in the graph of the system) and shortest computation paths to goal states where the states may be required to satisfy some boolean constraint.

Finally we look at the possibility of extending our work. Two areas of immediate importance arise. Analysis of the satisfiability problem for the logic RTCTL+ and efficient implementations of model checkers for the logics.

Acknowledgements

We would like to thank Insup Lee and Hong-liang Xie for drawing our attention to example specifications similar to the ones in Sections 4 and 6. We are grateful to Panagiotis Manolios and Kedar Namjoshi for their many insightful comments and questions regarding this work.

References

[ACD90] Alur, R., Courcoubetis, C., and Dill, D., Model Checking for Real-Time Systems. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, pp. 414-425, IEEE Computer Society Press, 1990.

[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.

[AH90] Alur, R., and Henzinger, T. A. , Real-time Logics: Complexity and Expressiveness. In *Proceedings of the 5th Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, New York, pp. 390-401, 1990.

[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.

[Al91] Alur, R., *Techniques for Automatic Verification of Real-Time Systems*. PhD thesis, Stanford University, 1991.

[AS95] Aziz, A., Singhal, V., Balarin, F., Brayton, R.K., and Sangiovanni-Vincentelli, A.L., It Usually Works: The Temporal Logic of Stochastic Systems. In *Proceedings of Computer Aided Verification 1995*, Springer Verlag, pp. 155-165, 1995.

[BdA95] Bianco, A. and de Alfaro, L., Model Checking of Probabilistic and Nondeterministic Systems. In *Foundations of Software Tech. and Theor. Comp. Sci.*, Lecture Notes in Computer Science, Springer-Verlag, 1995.

[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.$j_k = m - 1$, and

[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.

[Bur88] Burch, J. R., Modeling Timing Assumptions with Trace Theory, In *Proceeding of the IEEE International Conference on Computer Design*, 1989.

[Ch74] Choueka, Y., Theories of Automata on ω-tapes: A Simplified Approach. *Journal of Computer and System Sciences* vol. 8. pp. 117-141, 1974.

[CE94] Campos, S., Clarke, E., Marrero, W., Minea, M. and Hirashi, H., Computing Quantitative Characteristics of Finite-State Real-Time Systems. Technical Report CMU-CS-94-147, Carnegie Mellon University, School of Computer Science, 1994.

[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.

[Di88a] Dill, D., Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits, PhD Thesis, Dept. of Computer Science, Carnegie Mellon University, 1988.

[Di88b] Dill, D., Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits. In *Advanced Research in VLSI: Proceedings of the fifth MIT Conference*, 1988.

[EH86] Emerson, E. A., and Halpern, J. Y., 'Sometimes' and 'Not Never' Revisited: On Branching versus Linear Time Temporal Logic, *JACM*, vol. 33, no. 1, pp. 151-178, Jan. 86.

[Em81] Emerson, E. A., Branching Time Temporal Logics and the Design of Correct Concurrent Programs, Ph. D. Dissertation, Division of Applied Sciences, Harvard University, August 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.

[EL85] Emerson, E. A., and Lei, C. L., Temporal Model Checking Under Generalized Fairness Constraints. In *Proceedings of the 18th Hawaii International Conference on System Sciences*, Hawaii, 1985.

[EL87] Emerson, E. A., and Lei, C.-L., Modalities for Model Checking: Branching Time Strikes Back, pp. 84-96, ACM POPL85; journal version appears in Sci. Comp. Prog. vol. 8, pp 275-306, 1987.

[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.

[Fr86] Francez, N., Fairness, Springer-Verlag, New York, 1986

[HJ89] Hansson, H. and Jonsson, B., A Framework for Reasoning About Time and Reliability. In *Proceedings of the 10th Annual IEEE Real Time Systems Symposium*, Santa Monica, CA., December 5-7, pp. 102-111, 1989.

[Ha91] Hansson, H., Time and Probability in Formal Design of Distributed Systems. Ph.D. Dissertation, Uppsala University, Sweden, DoCS91/27, September 1991.

[HJ95] Henriksen, J.G., Jensen, J., Jorgensen, M., Klarlund, N., Paige, R., Rauhe, T., and Sandholm, A., MONA: Monadic Second-Order Logic in Practice, BRICS Report Series, BRICS RS-95-21, University of Aarhus, 1995.

[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.

[Le90] Lewis, H. R., A Logic of Concrete Time Intervals. *Proceedings of the 5th Annual Symposium on Logic in Computer Science* (LICS), IEEE Press, pp. 380-399, Philidelphia, 1990.

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

[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.

[St81] Streett, R., Propositional Dynamic Logic of Looping and Converse, PhD Thesis, MIT, 1981; journal version appears in Information and Control 54, 121-141, 1982.

[Th91] Thomas, W., Automata on Infinite objects. In *Handbook of Theoretical Computer Science*, vol. B, (J. van Leeuwen, ed.), Elsevier/North-Holland, 1991.

[Va95] Vardi, M., An Automata-theoretic Approach to Linear Temporal Temporal Logic. In *Logics for Concurrency* (Faron Moller and Graham Birtwistle, Eds.), Springer Verlag, Berlin, 1996, pp. 238-266.

[VW94] Vardi, M. Y. and Wolper, P., Reasoning About Infinite Computations. *Information and Computation*, vol. 115(1). November 1994, pp. 1-37.

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

A Automata

We review the definitions of automata on infinite strings that will be used in constructing the proofs later in the paper. The treatment here is minimal and we refer the reader to excellent articles on the subject by Vardi [Va95] and Thomas [Th91].

Let Σ be a finite alphabet. Σ^ω represents the set of strings over Σ that are of length ω. A string $x \in \Sigma^\omega$ has the form $x_0 x_1 \ldots$. When $i \in \mathcal{N}$ ($\mathcal{N}$ refers to the set of natural numbers), x_i refers to the ith element in the string and x^i refers to the suffix of x, $x_i x_{i+1} \ldots \in \Sigma^\omega$. A Büchi automaton [Bu62] $\mathcal{A} = (\Sigma, \mathcal{Q}, q_0, \delta, F)$ is an automaton which accepts strings over Σ of length ω. Formally,

- $\mathcal{Q}$ is a finite set of states $\{q_0, \ldots, q_{n-1}\}$.
- q_0 is the start state.
- $\delta : \mathcal{Q} \times \Sigma \rightarrow 2^{\mathcal{Q}}$ is the transition function.
- $F \subseteq \mathcal{Q}$ is a set of final states.

A run of $\mathcal{A}$ on $x \in \Sigma^\omega$ is a string $r \in \mathcal{Q}^\omega$ such that $r_0 = q_0$ and for all $i \geq 0, r_{i+1} \in \delta(r_i, x_i)$. r is an accepting run if $\inf(r) \cap F \neq \emptyset$, where $\inf(r)$ is the set of states in r that appear infinitely often. We say that $\mathcal{A}$ accepts $x \in \Sigma^\omega$ if $\mathcal{A}$ has an accepting run on x. $\mathrm{L}(\mathcal{A}) = \{\mathrm{x} \in \Sigma^\omega | \mathcal{A} \text{ accepts x}\}$ defines the language accepted by $\mathcal{A}$.

Let Q be a set then, $|Q|$ = the number of elements of Q. When Q is a graph $|Q|$ denotes the number of nodes plus the number of edges. Suppose x to be a string over some alphabet. Then $|x|$ = the length of x, where the empty string is of length 0 and if $x = ay$, a a member of the alphabet and y a string of the alphabet, then $|x| = |y| + 1$.

Theorem 15 *[EL85] [EL87] [VW94] Let $\mathcal{A} = (\Sigma, \mathcal{Q}, q_0, \delta, F)$ be a Büchi automaton. Then it is decidable whether $\mathrm{L}(\mathcal{A}) = \emptyset$ in time $\mathcal{O}(n)$. Where n represents the size of the transition graph of $\mathcal{A}$.*

Proof Sketch: Determine the set of strongly connected components (SCC's) of $\mathcal{Q}$ (this can be done in time linear in the size of $\mathcal{A}$). $\mathcal{L}(\mathcal{A}) = \emptyset$ iff there is no simple path from q_0 to an SCC which contains a state $q \in F$ (this can be determined in time linear in the size of $\mathcal{A}$).

□

Theorem 16 *[Ch74] Let $\mathcal{A}_1 = (\Sigma, \mathcal{Q}_1, q_{01}, \delta_1, F_1)$ and $\mathcal{A}_2 = (\Sigma, \mathcal{Q}_2, q_{02}, \delta_2, F_2)$ be Büchi automata. There is Büchi automaton $\mathcal{A} = (\Sigma, \mathcal{Q}, q_0, \delta, F)$ such that $\mathrm{L}(\mathcal{A}) = \mathrm{L}(\mathcal{A}_1) \cap \mathrm{L}(\mathcal{A}_2)$. $\mathcal{A}$ is known as the product automaton and is denoted $\mathcal{A}_1 \times \mathcal{A}_2$.*

Proof :

- $\mathcal{Q} = \mathcal{Q}_1 \times \mathcal{Q}_2 \times \{0, 1, 2\}$

- $q_0 = \langle q_{01}, q_{02}, 0 \rangle$
- $\delta : \mathcal{Q} \times \Sigma \to 2^{\mathcal{Q}}$, and is defined by the following where $\sigma \in \Sigma$.
  $\langle q_1', q_2', 0 \rangle \in \delta(\sigma, \langle q_1, q_2, 0 \rangle)$ if $q_1' \in \delta_1(\sigma, q_1)$ and $q_2' \in \delta_2(\sigma, q_2)$ and $q_1 \notin F_1$.
  $\langle q_1', q_2', 1 \rangle \in \delta(\sigma, \langle q_1, q_2, 0 \rangle)$ if $q_1' \in \delta_1(\sigma, q_1)$ and $q_2' \in \delta_2(\sigma, q_2)$ and $q_1 \in F_1$.
  $\langle q_1', q_2', 1 \rangle \in \delta(\sigma, \langle q_1, q_2, 1 \rangle)$ if $q_1' \in \delta_1(\sigma, q_1)$ and $q_2' \in \delta_2(\sigma, q_2)$ and $q_2 \notin F_2$.
  $\langle q_1', q_2', 2 \rangle \in \delta(\sigma, \langle q_1, q_2, 1 \rangle)$ if $q_1' \in \delta_1(\sigma, q_1)$ and $q_2' \in \delta_2(\sigma, q_2)$ and $q_2 \in F_2$.
  $\langle q_1', q_2', 0 \rangle \in \delta(\sigma, \langle q_1, q_2, 2 \rangle)$ if $q_1' \in \delta_1(\sigma, q_1)$ and $q_2' \in \delta_2(\sigma, q_2)$.
- $F = \{\langle q_1, q_2, 2 \rangle | q_1 \in \mathcal{Q}_1, q_2 \in \mathcal{Q}_2\}$

The counter cycling through 0, 1 and 2, in the above construction, ensures that any accepting run of the automaton $\mathcal{A}$ will visit accepting states of $\mathcal{A}_1$ and $\mathcal{A}_2$ infinitely often. A straightforward argument shows that $\mathcal{A}$ has an accepting run on $x \in \Sigma^\omega$ iff $\mathcal{A}_1$ and $\mathcal{A}_2$ have accepting runs on x.

□