On model-checking for μ-calculus and its fragments

E. A. Emerson* C. S. Jutla† A. P. Sistla‡

August 3, 1999

Abstract

In this paper, we consider the model-checking problem for μ-calculus and show that it is equivalent to the non-emptiness problem of a certain class of automata on infinite binary trees. We also present efficient model-checking algorithms for two rich subclasses of μ-calculus formulas and relate their expressive power to well known extensions of branching time temporal logics.

1 Introduction

In this paper we consider the problem of model checking for different fragments of propositional μ-calculus. This logic was studied by many authors [6, 13] for specifying the properties of concurrent programs. It has been shown (see [18, 16, 7]) to be as expressive of automata on infinite trees. Most of the known temporal and dynamic logics can be translated into this logic.

The model checking problem for this logic was first considered in [10]. In that paper, the authors presented an algorithm that is of complexity $O((mn)^{l+1})$ where m is the length of the formula, n is the size of the Kripke structure and l is the number of alternations of least and greatest fixed points in the given formula. Thus the complexity of the algorithm is exponential in the length of the formula. Since then there have been other algorithms [3, 5, 17] that were presented. Although some of these algorithms have lower complexity than the original algorithm, their complexity is still exponential. Algorithms of linear complexity (both in the size of the structure and the formula) were given [4] for the case when there is no alternation of least and greatest fixed points in the given formula.

[1]Department of Computer Science, University of Texas at Austin, Austin, Texas. This author's research is supported in part by the ONR grant N00014-89-J-1472 and Texas Advanced Technology Program Grant 003658-250.

[2]I.B.M. Thomas J. Watson Research Laboratories

[3]Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, Chicago, IL 60680. This author's research is supported in part by NSF grants CCR-9623229, CCR-9212183, CCR-9633536.

In this paper, we consider the model-checking problem for the full μ-calculus and show that this problem is equivalent to the non-emptiness problem of parity tree automata considered in [15, 7]. More specifically, we show that the model checking problem for μ-calculus is reducible to the non-emptiness problem for parity tree automata of size $O(mn)$ where m and n are as defined above. We also show that the non-emptiness problem of parity tree automata of size p is reducible to the model checking problem for μ-calculus in which the size of the Kripke structure is $O(p)$ and the length of the formula is $O(p)$. This shows that there is an efficient algorithm for one of them iff there is such an algorithm for the other. We also show that the model-checking problem for μ-calculus is in NP∩co-NP.

Next, we consider the model checking problem for different fragments of the μ-calculus. We first consider two fragments called L_1, L_2 where L_1 is a subset of L_2. We present model checking algorithms for these fragments which are of complexity $O(mnp)$ where m is the length of the formula and n is the size of the structure and p is the alternation depth of the formula (defined in section 2). The formulas in L_1 and L_2 allow arbitrary nesting of the least and greatest fixed points. However, they restrict how the modal operators and the boolean connectives can appear in the formula. More specifically, L_1 is the set of formulas containing least and greatest fixed points, the modal operator `<R>` in which negations only apply to atomic propositions and which satisfy the following restrictions: where ever $\wedge$ is used, at least one of the two conjuncts is a propositional formula. L_2 is the set of formulas satisfying the following restrictions: negations can be applied only to closed formulas (i.e. formulas without free variables); where ever $\wedge$ appears, at least one of the two conjuncts is a closed formula. The fragment L_2 is shown to be exactly as expressive as the branching time temporal logic ECTL* considered in [19]. ECTL* is the extended version of CTL* in which ω-regular expressions are used as path formulas. We show that L_1 is exactly as expressive as the set of formulas of in ECTL* of the form $E(W)$ where E is the existential path quantifier and W is a ω-regular expression.

Preliminary version of this paper was first presented in [9]. This is one of the earliest papers that reduced model-checking problem for μ-calculus (and other branching time logics to tree automata) to the emptiness problem of tree automata. Since then there were other works that explored relationships between model-checking for μ-calculus and the emptiness problem for automata. In particular, [14] showed the equivalence to the emptiness problem for alternating string automata. Results relating the model-checking problem to the emptiness problem of tree automata have also been reported in [2]. In that work as well as our work (cf. [9]) an essential notion is to model check a branching mu-calculus formula by taking the product of its syntax diagram with the Kripke. In [2] it was explicitly articulated that this syntax diagram defined an alternating tree automaton, a topic also discussed in [7]. Since the publication of our result [9] showing that the model-checking problem for μ-calculus is in NP∩co-NP, there have been other results on this problem; in particular, [12] establishes a slightly stronger result showing that the above problem is in UP∩co-UP.

The work described in [1] (which appeared after [9]) presents on-the-fly model-checking algorithms for the logics L_1 and L_2; these algorithms are of complexity $O(|f| \cdot \mathit{alt\_level}(f) \cdot$

$(|S| + |R|))$ where $alt\_level(f)$ is the level of alternations of μs and νs in f. The definition of $alt\_level(f)$, given in [1], is different from our definition of $alt\_depth(f)$.

Our paper is organized as follows. Section 2 contains definitions and notation. Section 3 contains the result showing the equivalence of the model checking problem for the full μ-calculus and non-emptiness problem for the parity tree automata. Section 4 presents the model checking algorithms for the logics L_1 and L_2. Section 5 presents expressiveness results for the logics L_1 and L_2.

2 Definitions and Notation

In this section we define the syntax and semantics of the different fragments of the logic μ-calculus. Let $\mathcal{P}$ and $\mathcal{X}$ be two disjoint sets of elements. The elements of $\mathcal{P}$ will be called *atomic propositions* and are usually denoted by $P_1, P_2, , ...$ The elements of $\mathcal{X}$ will be called variables and are usually denoted by $x, y, ...$ The formulae of μ-calculus are formed using the symbols from $\mathcal{P}$, $\mathcal{X}$, the propositional connectives $\neg$ and $\wedge$, the modal operator `<R>`, and the symbol μ.

The set of well-formed formulas of μ-calculus are defined inductively. The symbols true and false are well-formed formulas. Every atomic proposition and every variable are well-formed formulas. If f and g are well-formed formulas then $\neg f$, $f \wedge g$ and `<R>`f are also well-formed formulas. In addition, if f is a well-formed formula in which all the occurrences of the variable x are in the scope of an even number of negations then $\mu x(f)$ is also a well-formed formula.

We say that a variable x is a free variable in a formula f if there is an occurrence of x in f which is not in the scope of some μx. Let *free-var*(f) denote all the variables that are free in f. A variable which appears in f and which is not free, is called a bound variable. A formula without any free variables is called a *closed* formula. We define the semantics of the formulas in μ-calculus with respect to a *Kripke structure*. A Kripke structure K over the set of atomic propositions $\mathcal{P}$ is a triple (S, R, L) where S is a finite set of states, $R \subseteq S \times S$ is a total binary relation (i.e. $\forall x \exists y (x, y) \in R)$), and $L : S \to 2^{\mathcal{P}}$. With each state s, L associates a set of atomic propositions that are true in that state. We assume that all Kripke structures are defined over the set $\mathcal{P}$ of atomic propositions unless otherwise stated. Let f be a formula with free-var$(f) = \{x_1, ..., x_k\}$. An evaluation ρ for f is a mapping that associates with each variable in free-var(f) a subset of S. If free-var(f) is empty then there is a unique empty evaluation ϵ for f. For a given Kripke structure K, we define a function $\mathcal{M}_{(K,f)}$ from the set of evaluations for f to the subsets of S, by induction on the structure of f as follows.

- $\mathcal{M}_{(K,P)}(\epsilon) = \{s : P \in L(s)\}$ where P is an atomic proposition;
- $\mathcal{M}_{(K,f \wedge g)}(\rho) = \mathcal{M}_{(K,f)}(\rho') \cap \mathcal{M}_{(K,g)}(\rho'')$ where ρ' and ρ'' are restrictions of ρ to the free variables of f and g respectively;
- $\mathcal{M}_{(K,\neg f)}(\rho) = S - \mathcal{M}_{(K,f)}(\rho)$;

- $\mathcal{M}_{(K,\texttt{<R>}f)}(\rho) = \{s : \exists s' \in \mathcal{M}_{(K,f)}(\rho)$ such that $(s, s') \in R\}$;
- $\mathcal{M}_{(K,\mu x f)}(\rho) = \bigcap\{\rho'(x) : \mathcal{M}_{(K,f)}(\rho') \subseteq \rho'(x)$ where ρ' is an extension of ρ such that for all $y \in$ free-var(f) and $y \neq x$, $\rho'(y) = \rho(y)\}$.

In the above definition, it is to be noted that the value of $\mathcal{M}_{(K,\mu x f)}(\rho)$ is given as a least fixed point. For a closed formula f, we say that a state s in K satisfies f (written as $K, s \models f$) iff $s \in \mathcal{M}_{(K,f)}(\epsilon)$. We define derived connectives defined as follows: $f \vee g \equiv \neg(\neg f \wedge \neg g)$, $f \rightarrow g \equiv (\neg f \vee g)$, $\texttt{[R]}f \equiv \neg\texttt{<R>}\neg f$, $\nu y f(y) \equiv \neg \mu x(\neg f(\neg x))$. It is to be noted that while μx denotes the least fixed point νy denotes the greatest fixed point operator. A formula that has no variables and no occurrence of μ and ν will be called a constant. A constant formula that has no occurrence of the modal operators $\texttt{<R>},\texttt{[R]}$ is called a propositional formula. The following lemma gives a well known property of greatest fix points and can be proven from the basic definitions.

Lemma 2.1 *Let K be any Kripke structure, $\nu x(f)$ be any formula and ρ be any evaluation for $\nu x(f)$. Then,*
$\mathcal{M}_{(K,\nu x f)}(\rho) = \bigcup\{\rho'(x) : \mathcal{M}_{(K,f)}(\rho') \supseteq \rho'(x)$ *where ρ' is evaluation for f and is an extension of ρ such that for every $y \in$ free-var(f) such that $y \neq x$, $\rho'(y) = \rho(y)\}$.*

By using DeMorgan's laws, the identities $\neg\nu y f(y) \equiv \mu x(\neg f(\neg x))$ and $\neg\texttt{[R]}f \equiv \texttt{<R>}\neg f$, we can transform any formula into an equivalent formula in which all negations apply only to the atomic propositions. Such formulas will be called normalized formulas. In our paper we will be interested in these types of formulas. A formula of the form $\mu x f$ (resp., $\nu x f$) will be called a μ-formula (resp., ν-formula).

We assume, throughout the paper, that each variable appearing in a formula is bound at most once. This means that we can not have two sub-formulas of the form $\mu x(g)$ and $\mu x(h)$ appearing in a formula. If this property is not satisfied, then by renaming the variables we can obtain an equivalent formula that satisfies this property. For any formula f, we let $SF(f)$ denote the set of sub-formulas of f.

With a normalized formula f, we define a positive integer $alt\_depth(f)$ as follows. For this, we first define the notions (actually, binary relations on $SF(f)$) *direct active* sub-formulas and *active* sub-formulas as follows. Let f be a μ-formula or a ν-formula, i.e. $f = \delta x(f')$ where $\delta \in \{\nu, \mu\}$. We say that a sub-formula g of f is a direct active sub-formula of f if $g \neq f$ (i.e. g is a strict sub-formula) and the variable x appears in g. It is fairly straightforward to show that the binary relation "direct active sub-formula" is a partial order. The transitive closure of this partial order is the relation "active sub-formula". Formally, g is an active sub-formula of f, if there exists a sequence (or a chain) of sub-formulas $h_1, h_2, ..., h_k$ such that $h_1 = f$, $h_k = g$ and for each i $(1 \leq i < k)$ h_{i+1} is a direct sub-formula of h_i.

Example: Let f be the formula $\mu x(P_1 \vee g)$ where $g = \nu y((P_2 \wedge x) \vee h)$ and $h = \mu z(y \vee P_1)$. It is easy to see that g is a direct sub-formula of f and h is a direct sub-formula of g.

The following is an alternate way of defining active sub-formulas. Let f be any open or closed formula. Construct a graph H_f, called *syntax graph* of f, as follows. Take the parse tree for the formula f, and for each leaf node in the parse tree that is a variable x which is bound in f, add a back edge from x to the unique sub-formula in the parse tree that binds x, i.e. the unique sub-formula of the form $\delta x(g')$ where $\delta \in \{\mu, \nu\}$. The above back edges create cycles in H_f. It is fairly easy to show that g is an active sub-formula of f iff there is a path from g to f in H_f.

Now, we define $alt\_depth(f)$ as follows.

- For a μ-formula f, $alt\_depth(f) = 0$ if f has no active ν-sub-formulas in it, otherwise $alt\_depth(f) = 1 + max\{alt\_depth(g) : g \textit{is an active } \nu\textit{-sub-formula of } f\}$.

- For a ν-formula f, $alt\_depth(f) = 0$ if f has no active μ-sub-formulas in it, otherwise $alt_depth(f) = 1 + max\{$alt_depth(g): g *is an active* μ*-sub-formula of* $f\}$.

- For any formula f, define $alt_depth(f) = max\{$alt_depth(g): g *is a* μ*-sub-formula or a* ν*-sub-formula of* $f\}$.

Equivalently, we can define $alt\_depth(f)$ to be the maximum number of alternations of μs and νs in any chain of direct active sub-formulas starting with f (i.e. each sub-formula in the chain is a direct active sub-formula of the preceding one).

Example: The alternation depth of the formula $\nu x(\mu y(P_1 \vee \texttt{<R>}y) \wedge \texttt{[R]}x)$ is zero. Note that the μ-sub-formula is not an active sub-formula of the main formula. On the other hand the alternation depth of, the slightly different formula, $\nu x(\mu y((P_1 \wedge x) \vee \texttt{<R>}y) \wedge \texttt{[R]}x)$ is one.

For finite Kripke structures, the least fixed point can be computed by iteration starting with an empty set and iterating until a fixed point is reached. Similarly, the greatest fixed point can be computed by starting from the set containing all states and iterating until a fixed pont is reached. These results are due to Tarski/Knaster.

3 Relationship between Model Checking and Automata

In this section we explore the relationship between the model checking problem for μ-calculus and the emptiness problem for automata on infinite trees. More specifically, we show that the model checking problem for the complete logic μ-calculus is equivalent under linear reductions to the emptiness problem of a particular type of automata on infinite trees, called parity automata. This shows that there is an efficient model checking algorithm for μ-calculus iff there is an efficient algorithm for checking emptiness of parity automata.

A corollary of the above result is that the model checking problem for formulas of μ-calculus which are of the form $\nu y(g)$ where g is in normal form and μ is the only fixed point operator appearing in g, is equivalent to the non-emptiness problem for Buchi tree automata

First, we define parity tree automata (introduced in [7, 15]). A parity tree automaton A on infinite binary trees is a 5-tuple $(\Sigma, Q, q_0, \delta, F)$ where Σ is the input alphabet, Q is the set of automaton states, q_0 is the initial states, $\delta : (Q \times \Sigma) \to 2^{Q\times Q}$ is the next move relation and $F = (F_0, F_1, ..., F_k)$ where $F_0, F_1, ..., F_k$ is a sequence of mutually disjoint subsets of Q. We call F as the acceptance condition. Note that, for any $a \in \Sigma$ and $q \in Q$, $\delta(q, a)$ is a set of pairs of the form (q', q'') where q' and q'' are automaton states; Intuitively, if the automaton is in state q and reads input a in the current node then the state of the automaton on the left child is going to be q' and its state on the right child is going to be q''. Let $p = (p_0, ...p_i, ...)$ be an infinite sequence of states of the automaton A. We say that p satisfies the acceptance condition of A if the following condition is satisfied: there exists an even number l, $0 \leq l \leq k$, such that some state in F_l appears infinitely often in p and each of the states in the set $(\bigcup_{l<j\leq k} F_j)$ only appears finitely often in p. A path p in the automaton A is a finite or infinite sequence $p_0, ..., p_i, ...$ of states of the automaton such that the following condition is satisfied: for each $i \geq 0$, $p_i \in Q$ and for some $a \in \Sigma$ and some $q' \in Q$, either $(p_{i+1}, q') \in \delta(p_i, a)$ or $(q', p_{i+1}) \in \delta(p_i, a)$.

We denote the nodes of the infinite binary tree by the set $\{0, 1\}^*$. We let Γ denote this infinite binary tree. For any $x, y \in \{0, 1\}^*$, we let xy denote the concatenation of the strings x and y. The root node of Γ is the empty string; for any node x, $x0$ and $x1$, respectively, denote the left and right child of x. An infinite path σ in the tree is an infinite sequence of nodes starting with root node and such that each succeeding node is a child of the preceding node. A *labeled tree* r is a function with domain Γ. For a labeled tree r, the label of any node $x \in \Gamma$ is $r(x)$.

An input τ to the automaton is a labeled tree with range Σ, i.e. the label of each node is from Σ. A run of r of A starting from state q on input τ is itself a labeled tree with range Q such that the root node is labeled with q and the labeling of all other nodes is consistent with the transitions of the automaton; formally, $r : \{0, 1\}^* \to Q$ associates a state of the automaton with each node of the tree, such that $r(\epsilon) = q$, and for any $x \in \{0, 1\}^*$ $(r(x0), r(x1)) \in \delta(r(x), \tau(x))$. We simply state that r is a run of A on input τ if it is a run of A starting from state q_0 (i.e. the initial state of A) on input τ. Let r be a run starting from some state q on input τ. For any infinite sequence $\sigma = \sigma_0, \sigma_1, ..., \sigma_i, ...$, where each σ_i is a node in Γ, we let $r(\sigma)$ denote the infinite sequence of automaton states $r(\sigma_0), ..., r(\sigma_i), ...$ We say that $r(\sigma)$ satisfies the acceptance condition of A if the following condition is satisfied: there exists an even number l, $0 \leq l \leq k$, such that some state in F_l appears infinitely often in $r(\sigma)$ and each of the states in the set $(\bigcup_{l<j\leq k} F_j)$ only appears finitely often in $r(\sigma)$. The run r is an accepting run of A if for every infinite path σ, $r(\sigma)$ satisfies the acceptance condition of A. We say that the automaton A accepts an input τ iff there exists an accepting run r of the automaton on the input τ. We define the size of an automaton $A = (\Sigma, Q, q_0, \delta, F)$ to be the sum of the cardinality of Q, the total number of transitions (i.e. total number of triples of the form (q, a, q') such that $q \in Q$, $a \in \Sigma$ and $q' \in \delta(q, a)$) and the sum of the cardinalities of all sets in F.

A Buchi automaton is a parity tree automaton in which the accepting condition F is of length one, i.e. it has only one set. Note that this definition is equivalent to the standard definition of Buchi tree automaton.

Theorems 3.1 and 3.2 show the equivalence of the model-checking problem for μ-calculus and the emptiness problem parity tree automata. We need the following definitions in the proofs of these theorems.

Let f be a μ-calculus formula, possibly having some free variables. Recall that $SF(f)$ denotes the set of sub-formulas of f. Let $K = (S, R, L)$ be a given Kripke structure. We define a directed graph $G_{K,f} = (V, E)$, where V is the set of vertices and E is the set of edges, defined as follows. The node set $V = \{(s, g) : s \in S, g \in SF(f)\}$. Essentially, there is one node in V corresponding to each state in S and each sub-formula of f. The set of edges leaving the node (s, g) are, defined according to the outermost connective of the sub-formula g, as follows.

- If $g = P_1$ or $g = \neg P_1$ where P_1 is an atomic proposition or $g = x$ and x is a free variable in f then there is exactly one edge from (s, g) to itself.
- If $g = x$ and x is a bound variable in the formula f and g' is the largest sub-formula of f such that $g' = \mu x(g'')$ or $g' = \nu x(g'')$, then there is exactly one edge leaving (s, g) and this edge is to (s, g').
- If $g = \mu x(g')$ or $g = \nu x(g')$, then there is an edge from (s, g) to (s, g') and this is the only edge from (s, g).
- If $g = g' \wedge g''$ or $g = g' \vee g''$, then there are two edges from (s, g), to the nodes (s, g') and (s, g'').
- If $g = \texttt{<R>}g'$ or $g = \texttt{[R]}g'$, then for each state s' such that $(s, s') \in R$, there is an edge from (s, g) to (s', g').

Roughly speaking, $G_{K,f}$ is a product graph of the K and the syntax graph of f, i.e. the graph H_f. A path in $G_{K,f}$ is a finite sequence of nodes such that there is an edge in E from each node in the path to the succeeding node. A path starting and ending with the same node is a cycle. A *strongly connected subgraph* of $G_{K,f}$ is a set of nodes C such that there is a path between every pair of nodes in C passing only through the nodes of C. A strongly connected component (scc) is a maximal strongly connected subgraph.

We say that a cycle C in $G_{K,f}$, is a ν-cycle (respectively, μ-cycle) if the longest sub-formula appearing in a node on C is ν-sub-formula (respectively, μ-sub-formula). We call a node (s, h) in V to be an $\wedge$-node if h is of the form $h_1 \wedge h_2$ or is of the form $\texttt{[R]}h_1$ and (s, h) has at least two successors (i.e. edges to two distinct nodes). All other nodes in V are called $\vee$-nodes. We call (s, h) to be an atomic node if h is P or $\neg P$ for some atomic proposition P. Note that if (s, h) is a $\vee$-node then either it is an atomic node, or h is a variable, or a μ-sub-formula, or a ν-sub-formula, or a sub-formula of the form $\texttt{<R>}h'$, or a sub-formula of the form $\texttt{[R]}h'$ and (s, h) has only one successor.

Lemma 3.1 *The graph $G_{K,f}$ satisfies the following properties.*

- *Assume that there is an edge from (s, g) to (s', g') in $G_{K,f}$.*
  - *If $g = \texttt{<R>}g'$ or $g = \texttt{[R]}g'$ then $(s, s') \in R$; otherwise, $s' = s$.*
  - *If g is not a variable then g' is a sub-formula of g. If g is a variable then g is a sub-formula of g'.*
- *For any node (s, g) in $G_{K,f}$, there is a path from (s, g) to a node on a cycle iff g has at least one variable in it (i.e. g is not a constant).*
- *Let C be strongly connected subgraph of $G_{K,f}$ of cardinality greater than one, and let g be the longest formula appearing in all the nodes on C. Then, g is a μ-sub-formula or a ν-sub-formula. In addition, all other sub-formulas appearing in some node of C themselves are active sub-formulas of g.*

Theorem 3.1 *Given a Kripke structure $K = (S, R, L)$ and any μ-calculus formula f and a state $s_0 \in S$, we can obtain a parity tree automaton A of size $O((|S| + |R|)|f|)$ in time $O((|S| + |R|)|f|)$ such that*

- *the number of sets in the acceptance condition of A is $alt\_depth(f) + 2$,*
- *A accepts at least one input iff $K, s_0 \models f$.*

Proof: We prove the theorem for the more general case where f may be an open formula, i.e. a formula having free variables. Corresponding to the formula f, the Kripke structure K, an evaluation ρ for f and a state s of K, we construct an automaton $A_{K,f,\rho,s}$ such that $A_{K,f,\rho,s}$ accepts at least one input iff $s \in \mathcal{M}_{K,f}(\rho)$.

The proof of the theorem is structured as follows. First, we give the construction of the automaton $A_{K,f,\rho,s}$. After this, we present lemmas 3.2 and 3.3. These lemmas state certain properties of the automaton $A_{K,f,\rho,s}$ that are subsequently used. After this we state and prove lemma 3.4. This lemma shows that $s \in \mathcal{M}_{K,f}(\rho)$ iff $A_{K,f,\rho,s}$ accepts at least one input.

Now we give the construction of $A_{K,f,\rho,s}$. Corresponding to K and f, first we construct the directed graph $G_{K,f} = (V, E)$ as defined earlier. We transform $G_{K,f}$ in to another graph $G'_{K,f} = (V', E')$ so that every node in it has at most two successors, i.e. two edges leaving it. The node set $V' = V \cup V''$ where V'' is a new set of nodes. The sets V'' and E' are defined below. Consider any node $u \in V$. If the number of successors of u in $G_{K,f}$ is at most 2 then all edges leaving u in $G_{K,f}$ are also present in $G'_{K,f}$, i.e. all such edges are members of E'. Now consider a node u such that it has more than two successors in $G_{K,f}$. Corresponding to each such u we have the following nodes and edges in $G'_{K,f}$. Let the number of successors of u be l where $l > 2$. Let $u_1, u_2, ..., u_l$ be the successors of u in $G_{K,f}$. We introduce $l - 2$ new nodes $(u, 1), (u, 2), ..., (u, l - 2)$. All these nodes are members of V'' (note that (u, i) is distinct from u_i or any other node in V). We have the following edges in $G'_{K,f}$. There are two edges from u— to nodes u_1 and $(u, 1)$; for each i, $1 \leq i < l - 2$, there are two edges from (u, i) — to nodes u_{i+1} and $(u, i + 1)$; finally, there are two edges from $(u, l - 2)$ to nodes u_{l-1}

and u_l. It is easy to see that in $G'_{K,f}$ there is a path from u to each of the original successors passing through the intermediate vertices. The type of the new nodes is defined to be the same as that of u, i.e. each of them is defined to be a $\wedge$-node if u is a $\wedge$-node, etc. After this, each $\wedge$-node has exactly two successors, while a $\vee$-node has either one or two successors. It is not hard to see that $|V'|$ is bounded by $|V| + |E|$ and E' is bounded by $2|E|$. Thus the size of $G'_{K,f} = |V'| + |E'|$ is at most thrice the size of $G_{K,f}$.

The automaton $A_{K,f,\rho,s_0} = (\Sigma, Q, q_0, \delta, F)$ is defined as follows. The state set Q of the automaton is simply V', the initial state q_0 is (s_0, f), the input alphabet Σ has only one symbol, say symbol a. The transitions of A are defined as follows. For any node u, $\delta(u, a)$ consists of the following pairs: if u is a $\vee$-node then $\delta(u, a) = \{(v, v) : (u, v) \in E'\}$; if u is a $\wedge$-node then $\delta(u, a) = \{(v, v') : (u, v), (u, v') \in E'\}$. Note that $V' = V \cup V''$. We say that a state/node $u \in V$ is a g-state (or g-node) if $u \in V$ and $u = (s, g)$ for some $s \in S$. An atomic node is a P-node for some atomic proposition P.

Let $k = alt\_depth(f)$. Now, we define an alternating sequence $C_0, ..., C_{k+1}$ of sets of μ-sub-formulas and ν-sub-formulas as follows. All even numbered sets contain ν-sub-formulas and all odd numbered sets contain μ-sub-formulas defined as follows. For each $i = 0, ..., k+1$ C_i is defined as follows. If i is an even number then C_i is exactly the set of all ν-sub-formulas whose alternation depth is i or $i-1$. If i is an odd number then C_i is exactly the set of all μ-sub-formulas whose alternation depth is i or $i-1$. Note that C_0 contains all alternation free ν-sub-formulas. If $k+1$ is even (resp., odd) then C_{k+1} contains all ν-sub-formulas (resp., μ-sub-formulas) of alternation depth k. It is possible some of the C_is are empty sets.

Example: Let f be the formula $\nu x(g \vee P_1)$ where $g = \mu y(x \vee \texttt{<R>} h)$ and $h = \nu z(y \vee z \wedge P_2)$. It is not hard to see that $alt\_depth(f) = 2$ and $C_0 = \{h\}$; $C_1 = \{g\}$; $C_2 = \{f\}$.

Now, we define the acceptance condition $F = (F_0, F_1, ..., F_{(2k-1)})$ as follows.
$F_0 = (S \times C_0) \cup U_1 \cup U_2$ where U_1, U_2 are as defined below. U_1 is the set of all atomic nodes u such that u is of the form (s, P) and $P \in L(s)$, or is of the form $(s, \neg P)$ and $P \notin L(s)$. U_2 is the set of all nodes of the form (s, x) such that x is a free variable in f and $s \in \rho(x)$.
For $0 < i < 2k$, $F_i = S \times C_i$.

Since the input alphabet of A_{K,f,ρ,s_0} has a only one symbol a, there is only one input to the automaton which is the labeled binary tree in which all nodes are labeled with a; we denote this input as τ. In the remainder of the proof whenever we refer to a run of an automaton then the input is assumed to be τ.

Lemma 3.2 *A run r of A_{K,f,ρ,s_0} is accepting iff for every infinite path σ of Γ $r(\sigma)$ satisfies one of the following two properties (recall that $r(\sigma)$ is the sequence of automaton states along the path given by the run r).*

1. *The maximal length sub-formula g such that,for some s, (s, g) appears infinitely often in $r(\sigma)$ is a ν-formula (i.e. the maximum length sub-formula that appears infinitely often in $r(\sigma)$ is a ν-formula).*

2. *Some node of the form (s, g), satisfying the following condition, appears forever from a certain point in $r(\sigma)$:*

g is a literal (i.e. is of the form P or $\neg P$) which is satisfied in s, or g is a free variable x of f such that $s \in \rho(x)$.

Proof: First assume that r is an accepting run. Let σ be an infinite path in the binary tree, and $r(\sigma)$ denote the sequence of states that appear in the run r along the path σ. Let C be the set of automaton states that appear infinitely often in $r(\sigma)$. Since r is an accepting run, there exists an even number i such that $F_i \cap C \neq \emptyset$, and for all $j > i$ $F_j \cap C = \emptyset$ (recall that F_is are the subsets in the accepting condition of the automaton). It should be easy to see that C is a strongly connected subgraph of $G'_{K,f}$. From the way we have defined $G_{K,f}$ and $G'_{K,f}$, it follows that either C contains a single state of the form (s', P) or $(s', \neg P)$ or (s', x) where x is a free variable in f, or C does not contain any such state. In the former case the second property of the lemma holds. Now consider the later case. Let $D = \{h :$ for some s', $(s', h) \in C\}$. Let g be the maximum length sub-formula in D. From lemma 3.1, it follows that g is either a μ-sub-formula or ν-sub-formula, and for every $g' \in D$ either $g' = g$ or g' is an active sub-formula of g. If g is a μ-sub-formula then it would imply that $alt\_depth(g) > alt\_depth(g')$ for every ν-sub-formula $g' \in D$; this would imply that every node in C of the form (s', g) belongs to F_j for some $j > i$, which contradicts our earlier assumption. Hence g is a ν-formula, i.e. property 1 of the lemma is satisfied. The other direction of the lemma is proved on similar lines and is left to the reader. ■

Now, we continue the proof of the theorem. First, we need the following notation. Let r be any labeled binary tree and x be a node in Γ, i.e. $x \in \{0,1\}^*$. We define another labeled binary tree $restriction(r, x)$ as follows: for every $y \in \{0,1\}^*$, $restriction(r, x)(y) = r(xy)$. Intuitively, $restriction(r, x)$ is the restriction of r to the sub-tree rooted at x,

Let $X \subseteq \{0,1\}^*$ be a set of nodes such that no element in X is a prefix of another element in X, i.e. all the elements in X are incomparable. Also, let H be a function that associates a labeled binary tree $H(x)$ with each element $x \in X$. For each labeled binary tree r and for each X, H as defined above, we define another labeled binary tree, denoted by $modified(r, X, H)$, as follows. For every $y \in \{0,1\}^*$, the value of $modified(r, X, H)(y)$ is as given below. If there exist $x \in X$ and $z \in \{0,1\}^*$ such that $y = xz$ then $modified(r, X, H)(y) = H(x)(z)$ (note that such x and z are going to be unique since X is a set of incomparable elements); otherwise $modified(r, X, H)(y) = r(y)$. Intuitively, if y is a member of a subtree rooted at some node in $x \in X$ then the label given by $modified(r, X, H)$ is the same as that given by $H(x)$, otherwise it is same as that given by r.

Suppose f is a formula of the form $\nu x(g)$ or of the form $\mu x(g)$. Then the only difference between the graphs $G_{K,f}$ and $G_{K,g}$ is the following. The single edge from every node of the form (s, x) leads to the node (s,f) in $G_{K,f}$, while in $G_{K,g}$ this edge leads back to (s, x). As a consequence, the following lemma holds.

Lemma 3.3 *Let f be a formula of the form $\nu x(g)$ or $\mu x(g)$. Let ρ be an evaluation for f and ρ' be an evaluation for g which is an extension of ρ. Then the following properties hold.*

1. *Any finite or infinite path p of $A_{K,g,\rho',s}$ which has no x-nodes is also a path of $A_{K,f,\rho,s}$. Further more, if p is infinite then p satisfies the acceptance condition of $A_{K,g,\rho',s}$ iff it satisfies the acceptance condition of $A_{K,f,\rho,s}$.*

2. *Let r be a run of the automaton $A_{K,f,\rho,s}$ such that for all $z \in \{0,1\}^*$, $r(z)$ is not a x-state. Then $restriction(r,0) = restriction(r,1)$. Further more, $restriction(r,0)$ is a run of $A_{K,g,\rho',s}$, and it is an accepting run iff r is an accepting run.*

The proof of the theorem follows from the following lemma.

Lemma 3.4 *For a given formula f, Kripke structure $K = (S, R, L)$, an evaluation ρ for f and a state $s_0 \in S$,*
$s_0 \in \mathcal{M}_{K,f}(\rho)$ iff the automaton A_{K,f,ρ,s_0} accepts the input τ, i.e. the automaton accepts at least one input.

Proof: The lemma is proved by induction on the structure of f. The proof is trivial for the base cases when f is of the form P or $\neg P$ where P is an atomic proposition, or for the case when f is a variable. The proof of the induction step is divided in to different cases. These cases are based on the outer most connective of f. In each case, the proof is divided in to two parts. In the first part, we show that, if $s_0 \in \mathcal{M}_{K,f}(\rho)$ then the automaton A_{K,f,ρ,s_0} accepts the input τ; we call this as the left-to-right implication. In the second part, we prove the implication in the other direction, i.e., if the automaton A_{K,f,ρ,s_0} accepts the input τ then $s_0 \in \mathcal{M}_{K,f}(\rho)$; we call this as the right-to-left implication. The different cases of the induction step are given below.

$f = f_1 \wedge f_2$**:** By induction assume that the lemma holds for f_1 and f_2. To prove the left-to-right implication, assume that $s_0 \in \mathcal{M}_{K,f}(\rho)$. This implies that $s_0 \in \mathcal{M}_{K,f_1}(\rho)$ and $s_0 \in \mathcal{M}_{K,f_2}(\rho)$. By induction there exist accepting runs r_1, r_2 of the automata A_{K,f_1,ρ,s_0} and A_{K,f_2,ρ,s_0},respectively. Define a run r of A_{K,f,ρ,s_0} as follows: $r(\epsilon) = (s_0, f)$ (i.e. the root is labeled with (s_0, f)), $restriction(r, 0) = r_1$ and $restriction(r, 1) = r_2$ (i.e. the left and right sub-trees are labeled just as r_1 and r_2 respectively). From the way we defined the automata A_{K,f,ρ,s_0}, it should be easy to see that r is an accepting run. Hence τ is accepted by A_{K,f,ρ,s_0}. To prove the right-to-left implication, assume τ is accepted by A_{K,f,ρ,s_0} and let r be an accepting run of A_{K,f,ρ,s_0}. It should be easy to see that $restriction(r, 0)$ and $restriction(r, 1)$ are accepting runs of A_{K,f_1,ρ,s_0} and A_{K,f_2,ρ,s_0}, respectively, By induction, we see that $s_0 \in \mathcal{M}_{K,f_i}(\rho)$ for $i = 1, 2$. Hence $s_0 \in \mathcal{M}_{K,f}(\rho)$.

The induction steps for the cases when $f = f_1 \vee f_2$, $f = \texttt{[R]}f_1$ or $f = \texttt{<R>}f_1$ are fairly straightforward from the definition and are left to the reader.

$f = \mu x(g)$: Let ρ be any evaluation for f. Now we define an infinite sequence $\rho'_0, \rho'_1, ..., \rho'_i, ...$ of evaluations for g such that, for each $i \geq 0$, ρ'_i is an extension of ρ defined as follows: if $i = 0$ then $\rho'_i(x) = \emptyset$; otherwise, $\rho'_i(x) = \mathcal{M}_{K,g}(\rho'_{i-1})$. Since x appears positively in g, it is the case that for each $i \geq 0$, $\rho'_{i+1}(x) \supseteq \rho'_i(x)$. By Tarski/Knaster theorem, we have $\mathcal{M}_{K,f}(\rho) = \cup_{i\geq 0}\mathcal{M}_{K,g}(\rho'_i)$.

We prove the left-to-right implication by proving the following property by induction on i. For every $s \in \mathcal{M}_{K,g}(\rho'_i)$, there is an accepting run of $A_{K,f,\rho,s}$. To prove the base case (i.e. the case when $i = 0$), assume that $s \in \mathcal{M}_{K,g}(\rho'_0)$. By the induction hypothesis of the lemma, there is an accepting run r' of $A_{K,g,\rho'_0,s}$; since $\rho'_0(x) = \emptyset$, from the way we defined the acceptance condition of $A_{K,g,\rho'_0,s}$, it should be clear that none of the the nodes is labeled by a pair of the form (s', x) by the run r' (i.e. for all $y \in \{0,1\}^*$ and for all $s' \in S$, $r'(y) \neq (s', x)$). Let r be the labeled tree such that $r(\epsilon) = (s, f)$, $restriction(r, 0) = r'$ and $restriction(r, 1) = r'$ (i.e., r labels the root node with (s, f) and the restriction of r to the left and right sub-trees is same as r'). From lemma 3.3, it should be clear that r is an accepting run of the automaton $A_{K,f,\rho,s}$.

As the induction hypothesis, assume that
(A) for all $j \leq i$ and for all $s \in \mathcal{M}_{K,g}(\rho'_j)$, there is an accepting run of $A_{K,f,\rho,s}$.

Now consider any element $s \in \mathcal{M}_{K,g}(\rho'_{i+1})$ such that $s \notin \mathcal{M}_{K,g}(\rho'_j)$ for any $j \leq i$. By the induction hypothesis of the lemma, we see that there is an accepting run r' of $A_{K,g,\rho'_{i+1},s}$. Let Z be the set of all $z \in \{0,1\}^*$ such that $r'(z)$ is an x-state (i.e. is of the form (s', x)) and no proper prefix of z has this property (i.e. $r'(z)$ is labeled with a pair of the form (s', x) and no proper ancestor of z is similarly labeled). It is easy to see that all elements in Z are incomparable (i.e. none of them is a proper prefix of the other). Let $Z' = \{z0, z1 : z \in Z\}$. Now we define a function H that associates a run with each element in Z'. Let $z' \in Z'$. Then there exists $z \in Z$ such that $z' = z0$ of $z' = z1$. Clearly $r'(z) = (s', x)$ for some $s' \in S$. Since r' is an accepting run of $A_{K,g,\rho'_{i+1},s}$ and from the way we defined this automaton, it is the case that $s' \in \rho'_{i+1}(x)$. Since $\rho'_{i+1}(x) = \mathcal{M}_{K,g}(\rho'_i)$, it is the case that $s' \in \mathcal{M}_{K,g}(\rho'_i)$. From the induction hypothesis (A), we see that there exists an accepting run of $A_{K,f,\rho,s'}$. Now we define $H(z')$ to be any such accepting run of $A_{K,f,\rho,s'}$. Now we construct a run r of $A_{K,f,\rho,s}$ as follows: $r(\epsilon) = (s, f)$ and $restriction(r, 0) = restriction(r, 1) = modified(r', Z', H)$ (i.e., the root node is labeled with (s, f); restrictions of r to the left sub-tree, similarly to the right sub-tree, is the labeled tree obtained by modifying r' in the following way: for any node which is of the form $z'y$ for some $z' \in Z'$, $y \in \{0,1\}^*$, its label is $H(z')(y)$; for any other node its label is same as that given by r'). Now we show that r is an accepting run of $A_{K,f,\rho,s}$. Consider any infinite path $\sigma = \sigma_0, ..., \sigma_i, ...$ in the tree Γ starting from the root node. Now we have the following two cases: (1) $\exists i$ such that $r(\sigma_i)$ is an x-state , i.e. $r(\sigma_i) = (s', x)$ for some $s' \in S$; in this case let i be the smallest such integer; from the way we defined r, both labeled trees $restriction(r, \sigma_i 0)$ and $restriction(r, \sigma_i 1)$ are acceptance runs of $A_{K,f,\rho,s'}$; hence $r(\sigma_{i+1}), r(\sigma_{i+2}), ...$ satisfies the acceptance condition of $A_{K,f,\rho,s'}$ and hence it also satisfies the acceptance condition of $A_{K,f,\rho,s}$. (2) No such i exists; hence $r(\sigma_1), r(\sigma_2), ...$ satisfies the acceptance condition of $A_{K,g,\rho'_{i+1},s}$; from lemma 3.3, it is seen that $r(\sigma_1), ...$ also satisfies the acceptance $A_{K,f,\rho,s}$. Hence r is an accepting run of $A_{K,f,\rho,s}$.

Now, prove the right-to-left implication as follows. We show that for every state s of K, if there is an accepting run r of $A_{K,f,\rho,s}$ then $s \in \mathcal{M}_{K,f}(\rho)$; we prove the later by showing that $s \in \mathcal{M}_{K,g}(\rho'_i)$ for some $i \geq 0$.

Let r be an accepting run of $A_{K,f,\rho,s}$. Since r is an accepting run, on every infinite path p

of the binary tree the number of nodes z such that $r(z)$ is a x-state is finite; further more we assume that on every path no two nodes are labeled by r with the same pair of the form (s', x) (if this property is not satisfied we can always get another run that satisfies this property by pumping down using standard pumping lemma type argument). As a consequence, on every path, the number of nodes z such that $r(z)$ is a x-state is bounded by $|S|$. Let $m(r)$ denote the maximum number of nodes z, on any path of the binary tree, such that $r(z)$ is a x-state. By induction on the value of $m(r)$, we show that $s \in \mathcal{M}_{K,g}(\rho'_{m(r)})$; this would automatically imply that $s \in \mathcal{M}_{K,f}(\rho)$. The base case is when $m(r) = 0$. In this case, there is no node z such that $r(z)$ is a x-state. From lemma 3.3, we see that the $restriction(r, 0)$ is an accepting run of $A_{K,g,\rho'_0,s}$; from the induction hypothesis of the lemma it follows that $s \in \mathcal{M}_{K,g}(\rho'_0)$.

Now as an induction hypothesis assume (B) given below.

(B) for every state $s \in S$, if there is an accepting run r of $A_{K,f,\rho,s}$ such that $m(r) = i$ then $s \in \mathcal{M}_{K,g}(\rho'_i)$.

Now, let s be a state and r be an accepting run of $A_{K,f,\rho,s}$ such that $m(r) = i + 1$. Let Z be the set of all nodes z in the binary tree Γ such that $r(z)$ is a x-state and no proper ancestor of z satisfies this property (i.e. z is the first such state on the path from the root to z such that $r(z)$ is a x-state). Consider any $z \in Z$ and let $r(z) = (s', x)$ for some $s' \in S$. It should be easy to see that $r(z0) = r(z1) = (s', f)$. Let $r_1 = restriction(r, z0)$. It should be easy to see that r_1 is an accepting run of $A_{K,f,\rho,s'}$ and $m(r_1) \leq i$. By the induction hypothesis (B), it follows that $s' \in \mathcal{M}_{K,g}(\rho'_i)$, i.e. $s' \in \rho'_{i+1}(x)$. Now, define a run r' such that for every $z' \in \{0, 1\}^*$, $r'(z')$ is defined as follows. If z' is a descendant of some node $z \in Z$ then $r'(z') = r(z)$, otherwise $r'(z') = r(z')$. From our previous observations and using (1) of lemma 3.3, it should be easy to see that r' is an accepting run of $A_{K,g,\rho'_{i+1},s}$. From the induction hypothesis of the lemma it follows that $s \in \mathcal{M}_{K,g}(\rho'_{i+1})$.

$f = \nu x(g)$**:** To prove the left-to-right implication in the lemma, assume that $s_0 \in \mathcal{M}_{K,f}(\rho)$. Let $C = \mathcal{M}_{K,f}(\rho)$, and ρ' be an evaluation for g which is an extension of ρ such that $\rho'(x) = C$. Clearly, $\mathcal{M}_{K,g}(\rho') = C$. By the induction hypothesis of the lemma, we see that for each $s \in C$ there is an accepting run r_s of $A_{K,g,\rho',s}$. From the way we defined $A_{K,g,\rho',s}$, it should be easy to see that if the run r_s labels any node z with (s', x) then $s' \in C$ (this is because all the descendants of z are also labeled with (s', x), and every infinite path in this subtree satisfies the acceptance condition of the automaton which requires s' to be in $\rho'(x)$). Using these labeled trees, we construct a labeled infinite graph H as follows. We first put together all the labeled binary trees r_s $(s \in C)$; we distinguish the nodes in the different binary trees. Corresponding to each tree r_s, we introduce an additional node a_s and label it with (s, f); we introduce two directed edges from a_s to the root of r_s. Now consider any node z in r_s that is labeled with (s', x). Clearly, all descendants of z are also labeled with (s', x). Let z' and z'' be the two children of z. We modify the tree as follows. We discard all the nodes in the left and right subtrees of z' and z''. From each of the nodes z' and z'', we introduce two edges to the newly introduced node $a_{s'}$ (two edges are needed so that when we unwind the graph later, we get a binary tree). This modification is done for every pair of children z', z'' of nodes of the form z in each labeled tree r_s $(s \in C)$. Let H be the resulting labeled graph.

Now consider any infinite path p in H starting from a node labeled with a pair of the form (s, f) (i.e. a new node a_s introduced earlier). If p contains infinite number of nodes that are labeled with pairs of the form (s', x) then p also contains infinite number of nodes labeled with pairs of the form (s', f); since f is the longest formula appearing in the label of any node on p and it is a ν-formula, it follows that the sequence of labels of nodes on the path p satisfies the acceptance condition of the automaton $A_{K,f,\rho,s}$. If p contains only a finite number of nodes that are labeled with pairs of the form (s', x) then there exists a suffix of p that is entirely contained with in a labeled tree $r_{s''}$ for some $s'' \in C$; clearly in this case, the sequence of labels of nodes on p satisfies the acceptance condition of $A_{K,g,\rho',s''}$, and from lemma 3.3 it follows this sequence of labels on p also satisfies the acceptance condition of $A_{K,f,\rho,s}$. From this we see that, for each of the nodes a_s (i.e. each node labeled with a pair of the form (s, f)), the labeled infinite binary tree that we get when we unwind the graph H starting from a_s gives us an accepting run of $A_{K,f,\rho,s}$. Hence, $A_{K,f,\rho,s}$ has an accepting run for each $s \in C$. This holds when $s = s_0$.

To prove the right-to-left implication in the lemma, assume that there is an accepting run r of A_{K,f,ρ,s_0}. Let C be the set of all s such that some node in the tree is labeled with (s, f) (i.e. for some $u \in \{0,1\}^*$, $r(u) = (s, f)$), and let ρ' be an evaluation for g which is an extension of ρ such that $\rho'(x) \supseteq C$. Also, let z be any node in the tree which is labeled with (s, f) for some s (i.e. $s \in C$). Let z' and z'' be the two children of z; clearly, $r(z') = r(z'') = (s, g)$. Let r' be the restriction of r to the sub-tree rooted at z', i.e. $r' = \mathit{restriction}(r, z')$ (note that for every $u \in \{0,1\}^*$ $r'(u) = r(z'u)$). From r', we define another run r'' as follows. Let U be the set of all $u \in \{0,1\}^*$ such that $r'(u)$ is a x-state and no proper ancestor of u is labeled with a x-state (i.e. there is no u' which is a proper prefix of u such that $r'(u')$ is a x-state). Consider any $u \in U$ and let $r'(u) = (s', x)$ for some $s' \in S$. With u we associate a labeled tree $H(u)$ such that every node is labeled with (s', x) (i.e. $\forall y \in \{0,1\}^*$, $H(u)(y) = (s', x)$). Note that H is a function with domain U. Let $r'' = \mathit{modified}(r', U, H)$ (for any node y which is a descendant of some $u \in U$, $r''(y) = r'(u)$ and for all other y, $r''(y) = r'(y)$). Consider any infinite path p in the binary tree. If there exists any node on p whose label under r'' is an x-state then the labels of all succeeding nodes in p are also x-states and hence the sequence of labels in p satisfies the acceptance condition of $A_{K,g,\rho',s}$, On the other hand, if none of the nodes on p is labeled with a x-state by r'' then the sequence of labels on p given by r'' is same as that given by r'; this sequence of labels satisfies the acceptance condition of $A_{K,f,\rho,s}$, and from lemma 3.3, we see that this sequence of labels also satisfies the acceptance condition of $A_{K,g,\rho',s}$. Hence r'' is accepting run of $A_{K,g,\rho',s}$, By induction hypothesis of the lemma it follows that $s \in \mathcal{M}_{K,g}(\rho')$ and hence $\mathcal{M}_{K,g}(\rho') \supseteq C$, i.e. $\mathcal{M}_{K,g}(\rho') \supseteq \rho'(x)$. From the property of the maximal fix points as given by lemma 2.1, it follows that $C \subseteq \mathcal{M}_{K,f}(\rho)$. Since $s_0 \in C$, it follows that $s_0 \in \mathcal{M}_{K,f}(\rho)$. ■

Theorem 3.2 *Given a parity tree automaton $A = (\Sigma, Q, q_0, \delta, F)$, we can obtain a Kripke structure K whose size is linear in the size of A and a μ-calculus formula f which is linear* *in the length of the acceptance condition, and a state s_0 in K, such that*

- *$alt\_depth(f) = 1+$ the number of sets in F, and*
- *A accepts at least one input iff $K, s_0 \models f$.*

Proof: The proof uses similar techniques as those given in [7]. It is organized as follows. First, we give the construction of the Kripke structure K from the automaton A. Then we state and prove lemma 3.5, from which the proof of the theorem follows, as shown immediately after the statement of the lemma.

Without loss of generality, we can assume that the alphabet of A is a singleton consisting of the symbol a. Let the acceptance condition F be given by the sequence of $k-1$ sets $(F_0, F_1, ..., F_{k-2})$. Now, we define a sequence of k sets $(G_0, G_1, ..., G_{k-1})$ as follows. Let $G_0 = Q - \bigcup_{1 \le i < (k-1)} F_i$. For $1 \le i \le k-1$, let $G_i = F_{i-1}$. For each i, $0 \le i < k$, let P_i be a new atomic proposition and $\mathcal{P}' = \{P_i : 0 \le i < k\}$. We define the Kripke structure $K = (S, R, L)$ over the set of atomic propositions $\mathcal{P}'$ as follows: $S = Q \cup (Q \times Q \times Q)$; that is, the elements of S are of the form s_1 or of the form (s_1, s_2, s_3) where s_1, s_2, s_3 are the automaton states. Corresponding to every triple of automaton states $s_1, s_2, s_3 \in Q$ such that $(s_1, s_2) \in \delta(s_1, a)$, R has the following edges: an edge from the node s_1 to the node (s_1, s_2, s_3), an edge from (s_1, s_2, s_3) to s_2 and an edge from (s_1, s_2, s_3) to s_3. Formally, $R = \{(s_1, (s_1, s_2, s_3)), ((s_1, s_2, s_3), s_2), ((s_1, s_2, s_3), s_3) \colon s_1, s_2, s_3 \in Q \text{ and } (s_2, s_3) \in \delta(s_1, a)\}$;

For each $s \in S$, $L(s)$ is defined as follows: if $s \in Q$ then $L(s) = \{P_i\}$ where $i < k$ is the unique integer such that $s \in G_i$; otherwise, $L(s) = \emptyset$.

f is given by the following formula:
$\lambda_{k-1} x_{k-1} \lambda_{k-2} x_{k-2} ... \lambda_0 x_0 (\bigvee_{0 \le i < k} (P_i \wedge \texttt{<R>[R]} x_i))$
where $\lambda_{k-1} ... \lambda_0$ is an alternating sequence μs and νs ending with μ.

Now we show that $K, q_0 \models f$ iff A accepts at least one input. To show this, we prove a more general result. First, for each $l = -1, 0, ..., k-1$, we define a set of formulas Δ_l defined as follows.
Δ_{-1} is the set of variables of the form y where $y \notin \{x_0, x_1, ..., x_{k-1}\}$.
For each $l = 0, ..., k-1$, Δ_l is the set of formulas of the form
$g_l(y) = \lambda_l x_l \lambda_{l-1} x_{l-1} ... \lambda_0 x_0 (\bigvee_{0 \le i < k} (P_i \wedge \texttt{<R>[R]} x_i) \vee y)$
where $\lambda_l ... \lambda_0$ is an alternating sequence of μs and νs ending with μ and y is any variable not in $\{x_0, x_1, ..., x_{k-1}\}$.

We need the following definitions. Let $\pi = \pi_0, ..., \pi_i, ...$ be any infinite sequence of states of the of the automaton A. We define $maxindex(\pi)$ to be the maximum integer $u < k$ such that some state in G_u appears infinitely in π. We define two conditions $cond1$ and $cond2$ with parameters as follows. Let π be any infinite sequence of states as given above and l, m be integers such that $0 \le m \le l < k$. We define $cond1(\pi, l, m)$ to be the following condition: For all $i \ge 0$, $\pi_i \in (G_0 \cup G_1 \cup ... \cup G_l)$ and $maxindex(\pi)$ is an odd number and $maxindex(\pi) \le m$.
Let C be any subset of states of the automaton A. We define $cond2(\pi, l, C)$ to be the

following condition:
There exists an $i \geq 0$ such that $\pi_i \in C$ and for all j such that $0 \leq j < i$, $\pi_j \in (G_0 \cup G_1 ... \cup G_l)$.

We have the following lemma which is similar to theorem 4.1 of [7].

Lemma 3.5 *Let l be an integer such that $-1 \leq l < k$, $s \in Q$ be any state and ρ be any evaluation for g_l. Then, for every formula $g_l(y) \in \Delta_l$ (as given above), $g_l(y)$ is satisfied at the node s in K with respect to ρ (i.e. $s \in \mathcal{M}_{K,g_l}(\rho)$) iff there exists a run r of the automaton A starting from state s such that for every infinite path $\sigma = \sigma_0, \sigma_1, ..., \sigma_i, ...$ of the binary tree Γ at least one of the following conditions is satisfied.*

1. *$l \geq 0$ and $cond1(r(\sigma), l, l)$ holds, i.e. for all $i \geq 0$, $r(\sigma_i) \in (G_0 \cup G_1 \cup ... G_l)$ and $maxindex(r(\sigma))$ is an odd number.*

2. *$l \geq 0$ and $cond2(r(\sigma), l, \rho(y))$ holds, i.e. there exists an $i \geq 0$ such that $r(\sigma_i) \in \rho(y)$ (i.e. y is satisfied at node $r(\sigma_i)$ with respect to the evaluation ρ), and for all j, $0 \leq j < i$, $r(\sigma_j) \in (G_0 \cup G_1 \cup ... G_l)$.*

3. *$l = -1$ and $r(\sigma_0) \in \rho(y)$.*

Before we prove lemma 3.5, we complete the proof of theorem 3.2. First, observe that A accepts at least one input iff there exists a run r of A starting from q_0 such that, for every infinite path σ in the binary tree Γ, $maxindex(r(\sigma))$ is an odd number. Let ρ be the evaluation for the formula $g_{k-1}(y)$ such that $\rho(y) = \emptyset$. It is easy to see that , for any $s \in Q$, the formula f is satisfied at node s in K iff the formula $g_{k-1}(y)$ is satisfied at node s in K with respect to the evaluation ρ. From lemma 3.5, we see that $g_{k-1}(y)$ is satisfied at s with respect to ρ iff there exists a run r of A starting from s such that for every path σ in Γ $maxindex(r(\sigma))$ is an odd number (note that this is due to the fact that condition 2 is not satisfied as $\rho(y) = \emptyset$). Putting all the above observations together, we get the proof of theorem 3.2.

Proof of lemma 3.5: We prove the lemma by induction on l. The base case is when $l = -1$. By definition $g_{-1} = y$ for some $y \notin \{x_0, x_1, ..., x_{k-1}\}$. In this case the lemma holds trivially. Assume that the lemma holds for all values of l up to p. Now consider the case when $l = p + 1$. Now, we write g_{p+1} as $\lambda_{p+1} x_{p+1}(h)$ where
$h = \lambda_p x_p ... \lambda_0 x_0 (\bigvee_{0 \leq i \leq p} (P_i \wedge \texttt{<R>[R]} x_i) \vee (P_{p+1} \wedge \texttt{<R>[R]} x_{p+1}) \vee y)$.

Observe that h has two free variables x_{p+1}, y, while g_{p+1} has only one free variable which is y. Now we have two cases. The first case is when $p + 1$ is an odd number. In this case, $h = \nu x_{p+1}(h)$. We prove the induction step for this case as follows. Let ρ be any evaluation for g_{p+1}. Let $\rho'_0, \rho'_1, ..., \rho'_q, ...$ be evaluations for h which are extensions of ρ such that the following conditions are satisfied: $\rho'_0(x_{p+1}) = S$; for each $q \geq 0$, $\rho'_{q+1} = \mathcal{M}_{K,h}(\rho'_q)$. From Tarski-Knaster theorem, we know that $\mathcal{M}_{K,g_{p+1}}(\rho) = \bigcap_{q \geq 0} \mathcal{M}_{K,h}(\rho'_q)$. Now, we need lemma 3.6 as given below.

Lemma 3.6 *Let $\rho'_0, ..., \rho'_q, ...$ be evaluations for h as defined above. For all $q \geq 0$, for all $s' \in Q$, $s' \in \mathcal{M}_{K,h}(\rho'_q)$ iff there exists a run r' of A starting from s' such that for every path $\sigma' = \sigma'_0, ..., \sigma'_i, ...$ of the binary tree Γ at least one of the following conditions holds.*

(a) *$cond1(r'(\sigma'), p+1, p)$ holds, and further more, the number of values of j such that $r'(\sigma'_j) \in G_{p+1}$ (i.e., the cardinality of the set $\{j : r'(\sigma'_j) \in G_{p+1}\}$) is less than or equal to q.*

(b) *For some $i \geq 0$, $r'(\sigma'_i) \in \rho(y)$ and for all j such that $0 \leq j < i$, $r'(\sigma'_j) \in (G_0 \cup G_1 ... \cup G_{p+1})$ (i.e. $cond2(r'(\sigma'), p+1, \rho(y))$ holds) and further more, the number of values of j', such that $j' < i$ and $r'(\sigma'_{j'}) \in G_{p+1}$, is less than or equal to q.*

(c) *For some $i \geq 0$, $r'(\sigma'_i) \in G_{p+1}$ and for all j such that $0 \leq j < i$, $r'(\sigma'_j) \in (G_0 \cup G_1 ... \cup G_{p+1})$ (i.e. $cond2(r'(\sigma'), p+1, G_{p+1})$ holds) and further more, the number of values of j', such that $j' < i$ and $r'(\sigma'_{j'}) \in G_{p+1}$, is equal to q.*

Proof : The lemma can be proven by induction on q. In the basis step as well as the induction step of the proof, we use the inductive hypothesis of lemma 3.5. To do this we use the following approach. Let h_1 be the formula $(P_{p+1} \wedge \texttt{<R>[R]} x_{p+1}) \vee y$. Observe that h_1 is a sub-formula of h. In h we replace the sub-formula h_1 by a new variable z. Let h'' be the resulting formula. For each $q \geq 0$, we define an evaluation ρ''_q for h'' such that $\rho''_q(z) = \mathcal{M}_{K,h_1}(\rho'_q)$. To prove the basis as well as the induction step of the the lemma, we apply the inductive hypothesis of lemma 3.5 for h'' and the evaluation ρ''_q (i.e. apply lemma 3.5 by using h'' in place of g_l and evaluation ρ''_q in place of ρ). The details of the proof of lemma 3.6 are straightforward and are left to the reader. ■

Now we continue with the proof of the inductive step for the first case of lemma 3.5. To prove the inductive step in one direction, assume that r is any run of A starting from s such that for every infinite path σ of the binary tree Γ either $cond1(r(\sigma), p+1, p+1)$ or $cond2(r(\sigma), p+1, \rho(y))$ holds. Assume that $cond1(r(\sigma), p+1, p+1)$ is satisfied. Let $q \geq 0$ be any integer. Now we show that $s \in \mathcal{M}_{K,h}(\rho'_q)$. We have two sub-cases. The first sub-case is when the cardinality of the set $\{i : r(\sigma_i) \in G_{p+1}\}$ is greater than q; in this sub-case it is straightforward to see that $cond2(r(\sigma), p+1, G_{p+1})$ holds and condition (c) of lemma 3.6 is satisfied for r and σ. The second sub-case is when the cardinality of the set $\{i : r(\sigma_i) \in G_{p+1}\}$ is less than or equal to q; in this sub-case, it should be easy to see that $cond1(r(\sigma), p+1, p)$ holds and hence condition (a) of lemma 3.6 is satisfied. Thus, for every $q \geq 0$, either condition (a) or (c) of lemma 3.6 is satisfied for r and σ. Similarly, it can be shown that if $cond2(r(\sigma), p+1, \rho(y))$ holds then, for every $q \geq 0$, condition (b) or (c) of lemma 3.6 is satisfied for r and σ. From lemma 3.6 we see that, for every $q \geq 0$, $s \in \mathcal{M}_{K,h}(\rho'_q)$; hence, we see that $s \in \mathcal{M}_{K,g_{p+1}}(\rho)$.

To prove the inductive step in the other direction, assume that $s \in \mathcal{M}_{K,g_{p+1}}(\rho)$. By Tarski-Knaster theorem, we know that $s \in \mathcal{M}_{K,h}(\rho'_q)$ for every $q \geq 0$. Now, consider the case when $q = |Q|$ where $|Q|$ is the cardinality of Q. Clearly, $s \in \mathcal{M}_{K,h}(\rho'_q)$. Hence, from lemma 3.6, we see that there exists a run r' such that for every infinite path σ' of Γ one of the three conditions of lemma 3.6 is satisfied. Now, we construct a run r such that either condition 1 or condition 2 of lemma 3.5 is satisfied for every path σ of Γ.

First we define a graph H from which the run r can be constructed. Let U be the set of of all $u \in \Gamma$ satisfying the following three properties: (i) $r'(u) \in G_{p+1}$; (ii) for every ancestor u' of u , $u' \notin \rho(y)$; (iii) there exists exactly one proper ancestor $\gamma(u)$ of u such that $r'(\gamma(u)) = r'(u)$ (this means that no other proper ancestor of u has this property). For each $u \in \Gamma$, let $parent(u)$ denote the parent of u in Γ (note $parent(u)$ is the string obtained by deleting the rightmost bit in x). The graph H is obtained from the tree Γ by making the following change: for each $u \in U$, the edge from parent(u) to u is replaced by an edge from $parent(u)$ to $\gamma(u)$; we call such an edge as a back edge and all other edges are called forward edges. Formally, H is defined as follows. The nodes of the labeled graph H are elements of $\Gamma - \{u : u \text{ is a descendant of some element in } U\}$. For every node u in H, the edges from u are defined as follows: for each $b \in \{0, 1\}$, there is an edge from u to u' where u' is given below. If ub is a node in H then $u' = ub$ and in this case the edge is called a forward edge; otherwise ub has to be in U, and in this case, $u' = \gamma(ub)$ and the edge is called back edge. The node u' is called a left successor of u if $b = 0$; otherwise, it is called a right successor of u. It is not hard to see the ϵ is a node in H.

Claim: For every infinite path $\sigma = \sigma_0, \sigma_1, ..., \sigma_i, ...$ in H starting from ϵ, either $cond1(r'(\sigma), p+1, p+1)$ or $cond2(r'(\sigma), p+1, \rho(y))$ is satisfied.

Proof of the claim: We consider two cases. The first case is when σ contains a finite number of back edges. Now, we consider the first case. Assume that σ contains at least one back edge and let i be the maximum integer such that the edge σ_i to σ_{i+1} is a back edge. Clearly, for every $j \leq i$, σ_j is an ancestor of some node in U in the tree Γ, and from the definition of U it follows that $r'(\sigma_j) \notin \rho(y)$; since every infinite path σ' of Γ passing through σ_j satisfies either (a) or (b) or (c) of lemma 3.6, it follows that $r'(\sigma_j) \in (G_0 \cup G_1 \cup ... \cup G_{p+1})$. Let σ'' denote the the sequence $\sigma_{i+1}, \sigma_{i+2}, ...$, i.e. σ'' is the suffix of σ starting from σ_{i+1}. Now, it is easy to see that there exists an infinite path π in the tree Γ having σ'' as a suffix. In this path, every node u appearing before σ_{i+1} satisfies the following properties: u is an ancestor of σ_{i+1} in Γ and hence is also a node in H; since, σ_{i+1} is a parent of some node in U (because there is a back edge from σ_{i+1}), u is an ancestor of some node in U and hence $r'(u) \notin \rho(y)$. From the above observations it is easy to see that π is also a path in H. Recall that we are considering the case when $q = |Q|$. Now, we show that either $cond1(r'(\sigma''), p+1, p)$ or $cond2(r'(\sigma''), p+1, \rho(y))$ is satisfied. If $cond1(r'(\sigma''), p+1, p)$ is satisfied then it would imply that $cond1(r'(\sigma), p+1, p+1)$ is satisfied since σ'' is a suffix of σ. If $cond2(r'(\sigma''), p+1, \rho(y))$ is satisfied then it would imply that $cond2(r'(\sigma), p+1, \rho(y))$ is satisfied since for all $j \leq i$, $r'(\sigma_j) \notin \rho(y)$. Since π is a path in the tree Γ it satisfies either condition (a), (b) or (c) of lemma 3.6. If π satisfies condition (a) then it would imply that $cond1(r'(\sigma''), p+1, p+1)$ is satisfied since σ'' is a suffix of π. Now assume that π does not satisfy condition (a) of lemma 3.6. Now we show that condition (b) of lemma 3.6 is satisfied. Suppose (b) is not satisfied. This implies that π satisfies condition (c) of lemma

3.6. Hence there exists some $i' \geq 0$ satisfying the following three properties: (i)$r'(\pi_{i'}) \in G_{p+1}$ and for all $j' < i'$, $r'(\pi_{j'}) \in G_0 \cup ...G_{p+1}$; (ii) the cardinality of the set $\{j' : j' < i'$ and $r'(\pi_{j'}) \in G_{p+1}\}$ is exactly q;(iii) for all $j' \leq i'$, $r'(\pi_{j'}) \notin \rho(y)$. (property (iii) is satisfied since we assumed condition (b) of lemma 3.6 does not hold). From the above three properties and the assumption that $q = |Q|$, using the pigeon hole principle, we see that for some $j' \leq i'$, $\pi_{j'} \in U$. However, this contradicts our earlier observation that π is a path in H. Hence π satisfies condition (b) of lemma 3.6. Since, for all $j \leq i+1$, $r'(\pi_j) \notin \rho(y)$ and σ'' is a suffix of π, it follows that $cond2(r'(\sigma''), p+1, \rho(y))$ holds. Hence the claim holds for the case when the number of back edges in σ is a finite number greater than zero. If σ has no back edges then we use tha same argument as above by taking π to be σ.

Now consider the other case when σ contains an infinite number of back edges. In this case, for every $i \geq 0$, σ_i is an ancestor of some node in U in the tree Γ; hence, $r'(\sigma_i) \in (G_0 \cup G_1 \cup ...G_{p+1})$ (this can be seen using the same argument given at the beginning of last paragraph). Further more, there exists infinite number of values of i such that $r'(\sigma(i)) \in G_{p+1}$ and hence $maxindex(r'(\sigma)) = p+1$. From this, it follows that $cond1(r'(\sigma), p+1, p+1)$ holds. This completes the proof of the claim. ∎

Now we define the run r to be the run obtained by unwinding the graph H starting from the node ϵ. To define r formally, we first define a function ϕ from Γ to the nodes of H inductively as follows: $\phi(\epsilon) = \epsilon$; for every $u \in \Gamma$, $\phi(u0)$ is the left successor of $\phi(u)$ and $\phi(u1)$ is the right successor of u. Now, for every $u \in \Gamma$, we define $r(u)$ to be $r'(\phi(u))$.

Now we show that r satisfies the condition of lemma 3.5. Let $\sigma = \sigma_0, ..., \sigma_i, ...$ be any infinite path in Γ. Consider the sequence $\phi(\sigma) = \phi(\sigma_0), ..., \phi(\sigma_i), ...$ It should be easy to see that $\phi(\sigma)$ is a path in H starting from ϵ. From the previous claim we see that either $cond1(r'(\phi(\sigma)), p+1, p+1)$ (and hence $cond1(r(\sigma), p+1, p+1)$) or $cond2(r'(\phi(\sigma)), p+1, \rho(y))$ (and hence $cond2(r(\sigma), p+1, \rho(y))$) is satisfied; hence either condition 1 or condition 2 of lemma 3.5 is satisfied for r and σ with $l = p+1$. This completes the induction step of lemma 3.5 for the case when $p+1$ is odd.

In the case when $p+1$, λ_{p+1} is μ and in this proof of the induction step is simpler. In the proof, we use Tarski-Knaster theorem for least fix points. The details are fairly straightforward and are left to the reader. This completes the proof of lemm 3.5.

∎

Consider a formula of the form $\nu x f$ where f is in normalized form and has no further νs appearing in it. The alternation depth of this formula is one. Further more, all its strict sub-formulas have an alternation depth of zero. The automaton constructed by the above theorem will have three sets F_0, F_1, F_2 where F_0 corresponds to the atomic propositions, F_1 corresponds to all the μ-sub-formulas and F_2 corresponds to the main formula. In this case, it can be shown that we can discard F_1 and combine F_0 and F_2 in to a single set to get an acceptance condition with one set, which becomes a Buchi automaton.

Corollary 3.1 *The model checking problem for formulas of the form $\nu x f$ where f is in normalized form and has no further νs appearing in it is equivalent to checking non-emptiness* *of Buchi tree automata.*

Using the above corollary, it is easy to see that the model-checking problem for CTL can be reduced to the emptiness problem for Buchi automata.

Theorem 3.3 *The model checking problem for the full μ-calculus is in NP$\cap$co-NP. Formally, the set T of encodings of all triples (K, s_0, f) satisfying the following condition is in NP$\cap$co-NP: K is a Kripke structure, s_0 is a state in K and f is a μ-calculus formula such that $K, s_0 \models f$.*

Proof: From theorem 3.1, we see that the model-checking problem for μ-calculus, i.e. the set T, is polynomial time reducible to the emptiness problem of parity tree automata. The later problem has been shown to be in NP (see [8]). This implies that the model-checking problem is also in NP. To see that model-checking for μ-calculus is in co-NP, we show that the complement T' of T is polynomial time reducible to T. For this observe that $(K, s_0, f) \in T'$ iff s_0 does not satisfy f; s_0 does not satisfy f iff s_0 satisfies $\neg f$, i.e. $(K, s_0, \neg f) \in T$. ■

4 Model Checking for the restricted Logics

In this section, we present efficient procedure for model-checking for the two logics L_1 and L_2.

First, we define the two logics L_1 and L_2. L_1 is the smallest set C of formulas satisfying the following conditions.

1. $\mathcal{P} \cup \mathcal{X} \subseteq C$.

2. If $f, g \in C$ then $f \vee g$, `<R>`f, $\mu x(f)$ and $\nu x(f)$ are also in C.

3. If f is an atomic proposition, i.e. $f \in \mathcal{P}$, then $\neg f \in C$.

4. If $f, g \in C$ and at least one of them is a propositional formula then $f \wedge g \in C$.

Rule 3 states that negations can only be applied to atomic propositions. Rule 4 states that if we have a conjunction one of the two conjuncts has to be a propositional formula. Any formula in L_1 is called a L_1-formula. Note that all L_1-formulas are in normalized form. Intuitively, the above restrictions imply that a L_1-formula is almost like a linear-time formula.

Let L_2 be the smallest set C of formulas satisfying conditions 1,2,3a and 4a where 3a and 4a are as given below:

3a. If $f \in C$ and is a closed formula then $\neg f \in C$.

4a. If $f, g \in C$ and at least one them is a closed formula then $f \wedge g \in C$.

It is to be noted that the formula f in rule 3 should be an atomic proposition while in rule 3a it can be any closed L_2-formula. Similarly, in rule 4, at least one of f and g has to be an atomic proposition, while in 4a, at least one of them has to be a closed formula. As a consequence, rules 2 and 4 are special cases of rules 2a and 4a respectively. From this, it should be easy to see that L_1 is a subset of L_2. The expressive power of L_2 is characterized by theorems 5.1 and 5.2. given in the next section.

First, we consider the logic L_1 and present an efficient model-checking algorithm for this logic. This algorithm, as we show later, can be easily extended to the logic L_2. Note that in a L_1-formula all the negations apply only to the atomic propositions and hence every L_1-formula is in normalized form. Further more, the `[R]` operator does not appear in a L_1-formula.

Now, assume that we are given a L_1-formula f and a Kripke structure $K = (S, R, L)$. Now, we present an algorithm that determines all states in S that satisfy f. the algorithm first constructs the graph $G_{K,f}$ and labels its nodes as follows. The label of a node u is maintained in the variable $label(u)$. Each of these variables takes one of the three values— $\mathsf{true}, \mathsf{false}, NIL$, and is initialized to the value NIL. During the execution of the algorithm, the values of these variables will be set to true or false. When once a variable is set to one of these two values, it will never be changed. Furthermore, for any node $u = (s, g)$, at the end of the execution of the algorithm, $label(s, g) = \mathsf{true}$ iff $K, s \models g$.

At any time during the execution of the algorithm, if $label(u) = NIL$ then we say that node u is *unlabeled* at that time. We say that a path is unlabeled if all the nodes on the path are unlabeled. Let n be the length of the formula f. We execute the following algorithm on the graph $G_{K,f}$.

1. For each node $u \in V$, $label(u) \leftarrow NIL$.

2. For each $g \in SF(f)$ in increasing lengths of g, and for each $s \in S$, update $label(u)$, where $u = (s, g)$, as follows.

   - $g = P$:   *If* $P \in L(s)$ *then* $label(u) \leftarrow \mathsf{true}$ *else* $label(u) \leftarrow \mathsf{false}$.
   - $g = \neg P$:   *If* $P \notin L(s)$ *then* $label(u) \leftarrow \mathsf{true}$ *else* $label(u) \leftarrow \mathsf{false}$.
   - $g = g' \wedge g''$:
     If for all successors u' of u it is the case that $label(u') = \mathsf{true}$ *then* $label(u) \leftarrow \mathsf{true}$; *If* for some successor u' of u such that $label(u') = \mathsf{false}$ *then* $label(u) \leftarrow \mathsf{false}$. In other cases, $label(u)$ is unchanged (i.e. $= NIL$).
   - $g = g' \vee g''$ or $g =$ `<R>`g' :
     If for some successor u' of u $label(u') = \mathsf{true}$ *then* $label(u) \leftarrow \mathsf{true}$;
     If for all successors u' of u it is the case that $label(u') = \mathsf{false}$ *then* $label(u) \leftarrow \mathsf{false}$.
     In other cases, $label(u)$ is unchanged.
   - None of the above: $label(u)$ is unchanged.

3. *For* each unlabeled node $u \in V$, *if* there exists an unlabeled path from u to an unlabeled ν-cycle *then* $label(u) \leftarrow \mathsf{true}$.

4. *For* each unlabeled node u, $label(u) \leftarrow \mathsf{false}$.

Theorem 4.1 *After the execution of the above algorithm, for any node $u = (s, g)$ in $G_{K,f}$ where g is a closed sub-formula, $label(u) = \mathsf{true}$ iff $K, s \models g$.*

Proof: First, it is to be noted that after the execution of step 2 of the above algorithm, the following conditions are satisfied. For each node $u = (s, g)$ where g is a constant, $label(u) \neq NIL$. For this case, it should be easy to see that $label(s, g) = \mathsf{true}$ iff $K, s \models g$. Also, for every node $u = (s, g)$ such that $label(s, g) = NIL$, there is at least one successor node u' such that $label(u') = NIL$. In addition, if $g = g' \wedge g''$, then for one successor u', $label(u') = \mathsf{true}$ and for the other successor u'', $label(u'') = NIL$. Due to this property, each $\wedge$-node is effectively a $\vee$-node.

Now, from theorem 3.1, we see that a closed sub-formula g is satisfied in state s iff there is an accepting run of the automaton $A_{K,g,\rho,s}$ where ρ is the empty evaluation. Since each node in $G_{K,f}$ is effectively a $\vee$-node, it is not difficult to see that if the node (s, g) is still unlabeled after step 2, then s satisfies g iff there is an unlabeled path in $G_{K,f}$ that satisfies the acceptance condition of the automaton $A_{K,g,\epsilon,s}$, i.e. iff the longest sub-formula appearing infinitely often on the path is a ν-formula; this happens iff there is an unlabeled path from (s, g) to an unlabeled ν-cycle. Step 3 detects all such nodes and labels them as $true$. Step 4 labels all other nodes as $false$. ∎

Complexity

Below, we discuss the complexity of the above algorithm. First, it is to be noted that the number of vertices in $G_{K,f}$, i.e. $|V|$, is $O(|S||f|)$. The number of edges in $G_{K,f}$, i.e. $|E| = O(|R||f| + |S||f|)$. It is not difficult to see that steps 1, 2, 4 and 5 can all be implemented in time linear in $(|V| + |E|)$.

Step 3 can be implemented using an algorithm of complexity $O(alt\_depth(f)(|V| + |E|))$. This algorithm works as follows: It first identifies all nodes that lie on unlabeled ν-cycles as follows. For each $i \leq alt\_depth(f)$, let H_i denote the directed graph obtained by restricting $G_{K,f}$ to nodes of the form (s, h) where h is a sub-formula of f such that $alt\_depth(h) \leq i$, i.e. $H_i = (V_i, E_i)$ where $V_i = \{(s, h) : (s, h) \in V \text{ and } alt\_depth(h) \leq i\}$, and $E_i = \{(u, v) : (u, v) \in E \text{ and } u, v \in V_i\}$. We say that a strongly connected component (scc) C of H_i is a ν-scc if the longest sub-formula appearing in any node of C, i.e. the longest g such that (s, g) appears in C for some s, is a ν-sub-formula. The following lemma gives us a condition for identifying all nodes that lie on ν-cycles in $G_{K,f}$.

Lemma 4.1 *A node $(s, g) \in V$ lies on a ν-cycle in $G_{K,f}$ iff there exists an $i \geq alt\_depth(g)$ such that the scc of H_i that contains (s, g) is a ν-scc.*

Proof: In one direction, it is trivial to see that if the scc in H_i that contains (s, g) is a ν-scc then this scc itself gives a ν-cycle in $G_{K,f}$ that contains (s, g). To prove the other direction, assume that (s, g) lies on a ν-cycle L in $G_{K,f}$. Let (s', h) be a node on L such that h is the longest sub-formula. Clearly, h is a ν-sub-formula. From lemma 3.1, we see that g is a sub-formula of h and hence $alt\_depth(h) \geq alt\_depth(g)$. Now consider the graph H_i where $i = alt\_depth(h)$. It should be easy to see that both (s', h) and (s, g) belong to the same scc in H_i. Further more, if h' is the longest sub-formula appearing in this scc then h' has to be ν-sub-formula; this is because h is a sub-formula of h' and $alt\_depth(h') \leq i$. ■

To identify all nodes in $G_{K,f}$ that lie on a ν-cycle we do as follows. For each $i = 0, 1, ..., alt\_depth(f)$, we construct the graph H_i and identify all the nodes in each ν-scc. These are exactly the required nodes. Since, each H_i is of size at most the size of $G_{K,f}$, it is easy to see that this step takes time $O(alt\_depth(f)(|V| + |E|))$. The remainder of step 3 can be implemented in time $O(|V| + |E|)$. Thus the over all complexity of the algorithm is $O(alt\_depth(f)(|V| + |E|))$. Substituting for $|V|$ and $|E|$ in terms of $|S|$ and $|R|$, we get an over all complexity which is $O(|f| \cdot alt\_depth(f) \cdot (|S| + |R|))$.

The above algorithm can be naturally extended to the logic L_2 with the same complexity. Let f be an L_2 formula and $K = (S, R, L)$ be Kripke structure. In order to find all states in K that satisfy f we invoke procedure *check*, as described below, with arguments f and K. This procedure, on input g and the structure K', identifies certain sub-formulas and recursively determines all the states that satisfy these sub-formulas; after this it replaces all such sub-formulas in g by new atomic propositions, called auxiliary propositions; the resulting formula will be a L_1-formula; it model-checks for this formula using the previous algorithm. First, we need the following definition. A strict sub-formula g' of g is called a significant sub-formula if it contains at least one variable and it is a closed sub-formula.

The procedure *check*, with input formula g and input structure $K' = (S', R', L')$ over the set of atomic propositions $\mathcal{P}$, works as follows.

> If g has no significant sub-formulas then g is a L_1-formula. In this case, we use the algorithm for L_1-formulas to determine all states that satisfy g. Otherwise, we do the following. First, we determine all maximal significant sub-formulas. Let $g_1, ..., g_k$ be all such sub-formulas. For each $i = 1, ..., k$, we recursively invoke *check* to determine all states in K' that satisfy g_i. We introduce new atomic propositions $Q_1, ..., Q_k$. Let $\mathcal{P}' = \mathcal{P} \cup \{Q_1, ..., Q_k\}$. We define a new Kripke structure K'' over the set of atomic propositions $\mathcal{P}'$ as follows. $K'' = (S', R', L'')$ where for each state $s \in S'$, $L''(s) = L'(s) \cup \{Q_j : s \in \mathcal{M}_{K',g_j}(\epsilon)\}$. Note that the sets of states and transitions of K'' are same as those of K'; the labeling of each state is extended to include the new atomic proposition; $Q_j \in L''(s)$ iff g_j is satisfied in state s of K'. For each $i = 1, ..., k$, we replace each occurrence of g_i in g by Q_i. Let g' be the resulting formula. It should be easy to see that g' is a L_1-formula. It is also not difficult to see that g' is satisfied at a state s in K'' iff g is satisfied at the same state s in K'. Now, we use the previous model-

checking algorithm for L_1-formulas to determine all states in K'' that satisfy g'. The procedure *check* returns this set of states as the answer.

It is to be noted that procedure *check* eventually terminates since the number of significant occurrences of operators decreases in each recursive invocation.

Using appropriate data structures, it is not hard to see that this model-checking algorithm can be implemented so that it runs in time $O(|f| \cdot alt\_depth(f) \cdot (|S| + |R|))$.

5 Expressive Power of the Restricted Logic

We compare the expressive power of the logics to well known branching time temporal logics. Consider the branching time temporal logic CTL*. Let the ECTL* (given in [19]) denote the extended version of the logic CTL* where each path formula can be as expressive as ω-regular expressions. Below, we define the syntax and semantics of ECTL*. First we define regular expressions over a finite alphabet Σ inductively as follows: the empty set $\emptyset$, the empty string ϵ and every member of Σ are regular expressions; if U, V are regular expressions then $U^*, UV, (U \vee V)$ are regular expressions. As usual, with each regular expression R over Σ, we associate a set, $\mathcal{L}(R)$, of finite strings over Σ defined inductively as follows: $\mathcal{L}(\emptyset)$ is the empty set; $\mathcal{L}(\epsilon)$ is the singleton set containing the empty string; for any $a \in \Sigma$, $\mathcal{L}(a) = \{a\}$; $\mathcal{L}(UV)$ is the set of strings obtained by concatenating some string from $\mathcal{L}(U)$ with some string from $\mathcal{L}(V)$ in that order; $L(U^*)$ is the set of all strings obtained by concatenating zero or more strings belonging to $\mathcal{L}(U)$; $\mathcal{L}((U \vee V)) = \mathcal{L}(U) \cup \mathcal{L}(V)$. For a regular expression U and integer $n \geq 0$, we let U^n denote the regular expression obtained by concatenating U with itself $n-1$; for example $U^1 = U$, $U^2 = UU$; we let U^0 denote the regular expression ϵ.

An ω-regular expression W over a finite alphabet Σ is a finite union $bigcup_{1 \leq i \leq n} U_i(V_i)^\omega$ where, for each $1 \leq i \leq n$, U_i and V_i are regular expressions over Σ. With the ω-regular expression W, we associate a set $\mathcal{L}(W)$ of infinite strings over Σ defined as follows. First, for each $1 \leq i \leq n$, let $\mathcal{L}(U_i), \mathcal{L}(V_i)$ denote the set of finite strings denoted by the regular expressions U_i and V_i, respectively. We require that, for each $1 \leq i \leq n$, the empty string is not in $\mathcal{L}(V_i)$. For each $1 \leq i \leq n$, we let $\mathcal{L}(U_i(V_i)^\omega)$ denote the set of ω-strings obtained by concatenating an infinite sequence of finite strings $\sigma_1, \sigma_2, ..., \sigma_j, ...$ in that order, where $\sigma_1 \in \mathcal{L}(U_i)$ and for each $j \geq 2$, $\sigma_j \in \mathcal{L}(V_i)$. If either U_i or V_i is $\emptyset$ then $\mathcal{L}(U_i(V_i)^\omega)$ is the empty set. Now we define $\mathcal{L}(W)$ to be the set of ω-strings $\bigcup_{1 \leq i \leq n} \mathcal{L}(U_i(V_i)^\omega)$.

Now we define the syntax and semantics of ECTL* formulas. The formulas of ECTL* are formed using the symbols for atomic propositions drawn from $\mathcal{P}$, the propositional connectives $\wedge, vee$ and $\neg$, the path quantifiers E, A and the ω-regular expressions. The set of ECTL* formulas is the smallest set satisfying the following conditions.

- Every $P \in \mathcal{P}$ is a ECTL* formula.
- If f and g are ECTL* formulas then $\neg f$ and $f \vee g$ are also ECTL* formulas.

- If W is a ω-regular expression over a finite alphabet $\Sigma = \{a_1, a_2, ..., a_n\}$ and $f_1, f_2, ..., f_n$ are ECTL* formulas then $E(W(f_1, f_2, ..., f_n))$ is a ECTL* formula.

We define the semantics of ECTL* formulas in a Kripke structure $K = (S, R, L)$ as follows. For a formula f, we let $K, s \models f$ to denote that f is satisfied at state s in the structure K. The relation $\models$ is defined by induction on the structure of f as follows.

- $K, s \models P$ iff $P \in L(s)$.
- $K, s \models f_1 \vee f_2$ iff $K, s \models f_1$ or $K, s \models f_2$.
- Let W be a ω-regular expression over $\Sigma = \{a_1, ..., a_n\}$ and let $f_1, ..., f_n$ be ECTL* formulas. Then $K, s \models E(W(f_1, f_2, ..., f_n))$ if there exists an infinite path $p = (p_0, ..., p_j, ...)$ in the structure K (i.e. for each $j \geq 0$, $(p_j, p_{j+1}) \in R$) starting from the state s (i.e. $p_0 = s$) and there exists an infinite string $\sigma_0, \sigma_1, ..., \sigma_j, ...$ in $\mathcal{L}(W)$ such that the following condition is satisfied: for each $j \geq 0$, if $\sigma_j = a_i$ (for some $1 \leq i \leq n$) then $K, p_j \models f_i$.

The following theorem states that L_2 is at least as expressive as ECTL*.

Theorem 5.1 *Corresponding to every ECTL* formula f, there exists a closed formula $T(f)$ in the logic L_2 such that the following condition is satisfied: for every Kripke structure K and every state s of the Kripke structure, $K, s \models f$ iff $s \in \mathcal{M}_{K,T(f)}(\epsilon)$. (Recall that ϵ is the unique empty evaluation for closed formulas). Further more, if f is of the form $E(W(f_1, ..., f_n))$ where W is a ω-regular expression and $f_1, ..., f_n$ are propositional formulas then $T(f)$ is a L_1-formula.*

Proof: The proof of the theorem is organized as follows. First we introduce some notation and prove some subsequently needed properties in lemma 5.1. After this, we define a function T' that associates a μ-calculus formula $T'(U)$ with each regular expression U. Using T', we define a function T'' that associates a μ-calculus formula $T''(U)$ with an ω-regular expression U of the form $V(W)^\omega$. We prove certain needed properties of $T'(U)$ and $T''(U)$ in lemmas 5.2 and 5.3, respectively. After this, for any ECTL* formula f, we define the μ-calculus formula $T(f)$ inductively using the function T''. From lemma 5.3, it automatically follows that $T(f)$ satisfies the required condition of the theorem.

Now we introduce the following notation. Let g be a μ-calculus formula. Let $(Q_1, ...Q_k)$ be a sequence of distinct symbols such that each Q_i is either an atomic proposition or a variable appearing free in g. Let $(h_1, ..., h_k)$ be a sequence of μ-calculus formulas. Now, we define $g(h_1, h_2, ..., h_k/Q_1, Q_2, ..., Q_k)$ to be the formula obtained by replacing every occurrence of Q_i in g by h_i, for each $i = 1, 2, ..., k$. It is not hard to see that $g(h_1, h_2, ..., h_k/Q_1, Q_2, ..., Q_k)$ is a well-formed formula. The following lemma relates the semantics of a formula of the form $g(h/x)$ with the semantics of g and h. It can be proven by straightforward induction on the structure of g.

Lemma 5.1 *Let g and h be formulas such that g has one free variable x and h has one free variable y. Then, $g(h/x)$ is a formula with free variable y. For any Kripke structure K and any evaluation ρ for $g(h/x)$, $\mathcal{M}_{K,g(h/x)}(\rho) = \mathcal{M}_{K,g}(\rho')$ where ρ' is an evaluation for g such that $\rho'(x) = \mathcal{M}_{K,h}(\rho)$.*

Let U be any regular expression over the alphabet $\Sigma = \{a_1, ..., a_n\}$. We define a μ-calculus formula $T'(U)$ over the atomic propositions $a_1, ..., a_2$ with a free variable x such that the following property is satisfied: $T'(U)$ is satisfied at any state s of a Kripke structure K under an evaluation ρ iff there exists a string δ in the language of U and a finite path p in K starting from s and ending in a state that satisfies x such that the successive atomic propositions in the string δ are satisfied in the successive states of p. $T'(U)$ is defined inductively on the structure of U as follows:

- If $U = a_j$ for some $a_j \in \Sigma$ then $T'(U) = a_j \wedge \texttt{<R>}x$.
- If $U = (U_1 \vee U_2)$ then $T'(U) = T'(U_1) \vee T'(U_2)$.
- If $U = (U_1 U_2)$ then $T'(U) = T'(U_1)(T'(U_2)/x)$.
- If $U = (U_1)^*$ then $T'(U) = \mu y(x \vee T'(U_1)(y/x))$.

Lemma 5.2 *Let U be any regular expression over the alphabet Σ. For any Kripke structure $K = (S, R, L)$ over the set of atomic propositions Σ and for any evaluation ρ for $T'(U)$ the following condition is satisfied: $s \in \mathcal{M}_{K,T'(U)}(\rho)$ iff there exists a finite string $\delta = \delta_0, \delta_1, ..., \delta_m$ in $\mathcal{L}(U)$ and there exists a finite path $p = p_0, p_1, ..., p_{m+1}$ in K such that $p_{m+1} \in \rho(x)$ and for each $i = 0, 1, ..., m$, $\delta_i \in L(p_i)$.*

Proof: The proof is by induction on the structure of U. The proof is trivial for the case when $U = a_j$ for some $a_j \in \Sigma$ and for the case when $U = U_1 \vee U_2$. Now consider the case when $U = (U_1 U_2)$. By definition, $T'(U) = T'(U_1)(T'(U_2)/x)$. From lemma 5.1, we have $\mathcal{M}_{K,T'(U)}(\rho) = \mathcal{M}_{K,T'(U_1)}(\rho')$ where $\rho'(x) = \mathcal{M}_{K,T'(U_2)}(\rho)$. using the induction hypothesis for U_1, we get $s \in \mathcal{M}_{K,T'(U)}(\rho)$ iff there exists a finite path $p' = p'_0, ..., p'_{m'+1}$ starting from s and there exists a string $\delta' = \delta'_0, ..., \delta'_{m'}$ in $\mathcal{L}(U_1)$ such that for each $i = 0, 1, ..., m'$, $\delta'_i \in L(p'_i)$ and $p'_{m'+1} \in \rho'(x)$. By using the induction hypothesis for U_2, we see that $p'_{m'+1} \in \mathcal{M}_{K,T'(U_2}(\rho)$ iff there exists a path $p'' = p''_0, ..., p''_{m''+1}$ starting from $p'_{m'+1}$ (i.e. $p''_0 = p'_{m'+1}$) and a string $\delta'' = \delta''_0, ..., \delta''_{m''}$ in $\mathcal{L}(U_2)$ such that, for each $j = 0, ..., m''$, $\delta''_j \in L(p''_j)$ and $p''_{m''+1} \in \rho(x)$. It should b easy to see that the lemma is satisfied for U by taking p to be the concatenation of $p'_0, p'_1, ..., p_{m'}$ and p'', and δ to be the concatenation of δ' and δ''.

Now consider the case when $U = (U_1)^*$. We define a sequence of languages $Z_0, Z_1, ..., Z_i, ...$ as follows. Z_0 is the set containing the empty string. For $i > 0$, $Z_i = Z_{i-1} \cup \mathcal{L}((U_1)^i)$. Essentially, Z_i consists of all strings obtained by concatenating n strings from $\mathcal{L}(U_1)$ for some $n \leq i$. It is easy to see that $\mathcal{L}((U_1)^*) = \bigcup_{i \geq 0} Z_i$. Let g denote the formula $(x \vee T'(U_1)(y/x))$. Note that g has two free variables x and y. Let $\rho'_0, \rho'_1, ..., \rho'_i, ...$ be evaluations for g defined

as follows: $\rho'_0(y) = \emptyset, \rho'_0(x) =$
$rho(x)$; and for each $i > 0$, $\rho'_i(x) = \rho(x)$, $\rho'_i(y) = \mathcal{M}_{K,g}(\rho'_{i-1})$. From Tarski-Knaster theorem, we know that $\mathcal{M}_{K,T'(U)}(\rho) = \bigcup_{i\geq 0} \mathcal{M}_{K,g}(\rho'_i)$. By induction on i, we show that
(I) for any state s, $s \in \mathcal{M}_{K,g}(\rho'_i)$ iff there exists a path $p = p_0, ..., p_{m+1}$ in K starting from s and a string $\delta = \delta_0, ..., \delta_m$ in Z_i such that $p_{m+1} \in \rho(x)$ and for each $j = 0, ..., m$, $\delta_j \in L(p_j)$.

To see the base case of I, i.e, when $i = 0$, recall that g is the formula $x \vee T'(U_1)(y/x)$. Since $\rho'_0(y) = \emptyset$, it is not hard to see that $\mathcal{M}_{K,g}(\rho'_0) = \rho(x)$. From this observation, it is not hard to see that (I) holds for the base case by using δ to be the empty string and p to be the path of length zero. Now we prove the induction step. Assume that (I) holds for all values of $i \leq k$. Now consider the case when $i = k+1$. Now $s \in \mathcal{M}_{K,g}(\rho'_{k+1})$ iff $s \in \rho'_{k+1}(x)$ or $s \in \mathcal{M}_{K,T'(U_1)(y/x)}(\rho'_{k+1})$. By the induction hypothesis for the lemma, we see that $s \in \mathcal{M}_{K,T'(U_1)(y/x)}(\rho'_{k+1})$ iff there exists a path $p' = p'_0, ..., p'_{m'+1}$ and a string $\delta' = \delta'_0, ..., \delta'_{m'}$ in $\mathcal{L}(U_1)$ such that $p'_{m'+1} \in \rho'_{k+1}(y)$ and for each $j = 0, ..., m'$, $\delta'_j \in L(p'_j)$; since $\rho'_{k+1}(y) = \mathcal{M}_{K,g}(\rho'_k)$, from the induction hypothesis of (I) we see that $p'_{m'+1} \in \rho'_{k+1}(y)$ iff there exists a path $p'' = p''_0, ..., p''_{m+1}$ in K starting from $p'_{m'+1}$ (i.e. $p''_0 = p'_{m'+1}$) and a string $\delta'' = \delta''_0, ..., \delta''_m$ in Z_k such that $p''_{m+1} \in \rho(x)$ and for each $j = 0, ..., m$, $\delta''_j \in L(p''_j)$. Putting all the observations together, it should be easy to see that the induction step for I holds by taking the p to be the path $p'_0, p'_1, ..., p'_{m'}$ followed by the the path p'' and δ to be $\delta' \cdot \delta''$. ■

Now consider the ω-regular expression $U = V(W)^\omega$ where V, W are regular expressions over the alphabet $\Sigma = \{a_1, ...a_n\}$. Corresponding to U, we define a formula $T''(U)$ over the atomic propositions $a_1, ..., a_n$ as follows: $T''(U) = T'(V)((\nu x T'(W))/x)$. Note that $T'(V), T'(W)$ have one free variable x, and $T''(U)$ has no free variables. $T''(U)$ is obtained by substituting $\nu x T'(W)$ for every occurrence of x in $T'(V)$.

Lemma 5.3 *Let U be the ω-regular expression $V(W)^\omega$ over the alphabet Σ. For any Kripke structure $K = (S, R, L)$ over the set of atomic propositions Σ and for any state $s \in S$, the following condition is satisfied: $s \in \mathcal{M}_{K,T''(U)}(\epsilon)$ iff there exists an infinite string $\delta = \delta_0, \delta_1, ..., \delta_i, ...$ in $\mathcal{L}(U)$ and there exists an infinite path $p = p_0, p_1, ..., p_i, ...$ in K starting from s (i.e. $p_0 = s$) such that for each $i \geq 0$, $\delta_i \in L(p_i)$.*

Proof: First we prove property (J) given below. The lemma follows easily from this property.
(J) For any $t \in S$, $t \in \mathcal{M}_{K,\nu x T'(W)}(\epsilon)$ iff there exists an infinite path $p' = p'_0, ..., p'_j, ...$ in K starting from t and there exists an infinite string $\delta' = \delta'_0, ..., \delta'_j, ...$ in $\mathcal{L}(W^\omega)$ such that, for each $j \geq 0$, $\delta'_j \in L(p'_j)$.

Let $\rho_0, \rho_1, ..., \rho_i, ...$ be evaluations for $T'(W)$ defined as follows: $\rho_0(x) = S$; for each $i \geq 0$, $\rho_{i+1}(x) = \mathcal{M}_{K,T'(W)}(\rho_i)$. By induction on i, it can easily be shown that
(J') for any $t \in S$, $t \in \rho_i(x)$ iff there exists a finite path $p'' = p''_0, ..., p''_{m+1}$ in K starting from t and a finite string $\delta'' = \delta''_0, ..., \delta''_m$ in $\mathcal{L}(W^i)$ such that $p''_{m+1} \in \rho_i(x)$ and for each $j = 0, ..., m$, $\delta''_j \in L(p''_j)$.

The proof of (J') is fairly straight forward and is left to the reader. From Tarski-Knaster theorem, we see that $\mathcal{M}_{K,\nu xT'(W)}(\epsilon) = \bigcap_{i\geq 0} \rho_i(x)$. We have $t \in \mathcal{M}_{K,\nu xT'(W)}(\epsilon)$ iff for each $i \geq 0$, $t \in \rho_i(x)$ iff for each $i \geq 0$, there exists a finite path $p'' = p''_0, ..., p''_{m+1}$ in K starting from t and a finite string $\delta'' = \delta''_0, ..., \delta''_m$ in $\mathcal{L}(W^i)$ such that $p''_{m+1} \in \rho_i(x)$ and for each $j = 0, ..., m$, $\delta''_j \in L(p''_j)$. From this and the fact that K is a finite structure (and hence has bounded number of successors for each state), it is easy to see that property (J) holds. ∎

Now we continue with the proof of theorem 5.1. For any ECTL* formula f we define $T(f)$ inductively on the structure of f as follows.

- If $f = P$ for some $P \in \mathcal{P}$ then $T(f) = P$. If $f = \neg f_1$ then $T(f) = \neg T(f_1)$. If $f = f_1 \vee f_2$ then $T(f) = T(f_1) \vee T(f_2)$.
- If $f = E(W(f_1, f_2, ..., f_n))$ where W is a ω-regular expression over an alphabet $\Sigma = \{a_1, ..., a_n\}$ then $T(f)$ is defined as follows. Let $W = \bigcup_{1\leq i\leq k} U_i(V_i)^\omega$ where V_i and W_i are regular expressions over Σ. For each i, such that $1 \leq i \leq k$, let $g_i = T''(U_i(V_i)^\omega)(T(f_1), T(f_2), ..., T(f_n)/a_1, a_2, ..., a_n)$. Recall that the atomic propositions appearing in $T''(U_i(V_i)^\omega)$ are from the set Σ. Note that the formula $T''(U_i(V_i)^\omega)(T(f_1), T(f_2), ..., T(f_n)/a_1, a_2, ..., a_n)$ is obtained from $T''(U_i(V_i)^\omega)$ by replacing every occurrence of a_i by $T(f_i)$ for each $i = 1, ..., k$.

It is not hard to see that $T(f)$ as defined above is a closed formula in the logic L_2. Using earlier lemmas, by induction on the structure of f, it is straightforward to prove that $T(f)$ satisfies the condition of the theorem. Further more, it is easy to see that if f is a formula of the form $E(W(f_1, ..., f_n))$ where W is a ω-regular expression and $f_1, ..., f_n$ are propositional formulas then $T(f)$ is a L_1-formula. ∎

.

The following theorem states that ECTL* is at least as expressive as the logic L_2.

Theorem 5.2 *For every closed L_2-formula f there exists a ECTL* formula $\Theta(f)$ such that, for every Kripke structure K and every state s, $s \in \mathcal{M}_{K,f}(\epsilon)$ iff $K, s \models \Theta(f)$. Further more, if f is a L_1-formula then $\Theta(f)$ is a formula of the form $E(W(f_1, ..., f_n))$ where $f_1, ..., f_n$ are propositional formulas.*

Proof: First we prove the theorem for L_1 formulas and then extend it to L_2 formulas. Let $\Sigma = \{\phi : \phi \subseteq \mathcal{P}\}$. Σ is finite since $\mathcal{P}$ is finite. Let $RE(\Sigma)$ denote the regular expression $\phi_1 \vee \phi_2 \vee ...\phi_m$ where $\phi_1, \phi_2, ..., \phi_m$ are all the elements of Σ. Let $K = (S, R, L)$ be any Kripke structure over the set of atomic propositions $\mathcal{P}$. For any finite or infinite path $p = p_0, p_1, ..., p_i, ...$ in K, we let $L(p)$ denote the sequence $L(p_0), L(p_1), ..., L(p_i), ...$

Now, we have the following lemma.

Lemma 5.4 *Let f be any L_1-formula and $x_1, ..., x_k$ be the set of variables that appear free in f. Then there exist regular expressions $R_{f,1}, R_{f,2}, ..., R_{f,k}$ and an ω-regular expression W_f*

over the alphabet Σ such that the following property is satisfied: for every Kripke structure $K = (S, R, L)$, for every evaluation ρ for f and for all states $s \in S$, $s \in \mathcal{M}_{K,f}(\rho)$ iff one of the following two conditions holds: (i) there exists an infinite path p starting from s (i.e. $p_0 = s$) such that $L(p) \in \mathcal{L}(W_f)$; (ii) there exists a finite path $p = p_0, ..., p_i, p_{i+1}$ starting from s and an integer j, $1 \leq j \leq k$, such that $L(p_0, ..., p_i) \in \mathcal{L}(R_j)$ and $p_{i+1} \in \rho(x_j)$.

Proof: Note that a special case of the above lemma is when f is a closed formula. In this case, $s \in \mathcal{M}_{K,f}(\rho)$ iff condition (i) of the lemma is satisfied. For an arbitrary L_1-formula f, we prove the lemma by induction on the structure of f. The base cases are when f is a variable or an atomic proposition or negation of an atomic proposition. When f is a variable, say x_1, then the lemma is satisfied with $R_{f,1}$ being ϵ (in this case, $\mathcal{L}(R_1)$ is the singleton set containing the empty string) and W_f being the empty set. It is trivial to see the proof for the other base cases. Now we consider the following induction cases.

- Assume $f =$ `<R>`g. Assume that the lemma holds for g. The set of free variables of f is exactly same as the set of free variables of g. Define $W_f = RE(\Sigma)W_g$, and for each j, $1 \leq j \leq k$, let $R_{f,j} = RE(\Sigma)R_{g,j}$. It is straightforward to see that the lemma also holds for f using the above values for W_f and $R_{f,j}$ for each $j = 1, ..., k$.

- Assume $f = \mu x_{k+1}\,(g)$, Let $x_1, ..., x_{k+1}$ be the free variables of g. By the induction hypothesis assume that the lemma holds for g. Define W_f to be $(R_{g,k+1})^* W_g$. For each j, $1 \leq j \leq k$, define $R_{f,j} = (R_{g,k+1})^* R_{g,j}$.

  By using Tarski-Knaster theorem for least fix points, we know that $\mathcal{M}_{K,f}(\rho) = \bigcup_{r \geq 0} \mathcal{M}_{K,g}(\rho'_r)$ where each ρ'_r is an extension of ρ_r such that $\rho'_0(x_{k+1}) = \emptyset$, and for each $r \geq 0$, $\rho'_{r+1}(x_{k+1}) = \mathcal{M}_{K,g}(\rho'_r)$. By induction on r and using the induction hypothesis for the lemma, it is fairly easy to show that $s \in \mathcal{M}_{K,g}(\rho'_r)$ iff one of the following two properties holds: (i) there exists an infinite path p starting from s such that $L(p) \in \mathcal{L}((R_{g,k+1})^m W_g)$ for some $m \leq r$; (ii) there exists a finite path $p = p_0, ..., p_{i+1}$ and some $n \leq k$, such that $p_{i+1} \in \rho'_j(x_n)$ and $L(p_0, ..., p_i) \in \mathcal{L}((R_{g,k+1})^m R_{g,n})$ for some $m \leq r$. From this, it is easy to see that the lemma holds for for f with the above defined values for W_f and $R_{f,m}$ for each $m = 1, ..., k$.

- Assume $f = \nu x_{k+1}\,(g)$. As before, let $x_1, ..., x_{k+1}$ be the free variables of g. Assume that the lemma holds for g. Define W_f to be $(R_{g,k+1})^* W_g \cup (R_{g,k+1})^\omega$. For each j, $1 \leq j \leq k$, define $R_{f,j} = (R_{g,k+1})^* R_{g,j}$.

  Let $\rho'_0, \rho'_1, ..., \rho'_r, ...$ be a sequence of evaluations for g such that each of them is an extension of ρ, $\rho'_0(x_{k+1}) = S$, and for each $r \geq 0$, $\rho'_{r+1}(x_{k+1}) = \mathcal{M}_{K,g}(\rho'_r)$. By using Tarski-Knaster theorem for greatest fix points, we know that $\mathcal{M}_{K,f}(\rho) = \bigcap_{r \geq 0} \mathcal{M}_{K,g}(\rho'_r)$. By induction on r and using the induction hypothesis for the lemma, it is fairly easy to show that $s \in \mathcal{M}_{K,g}(\rho'_r)$ iff one of the following three properties holds: (i) there exists an infinite path p starting from s such that $L(p) \in \mathcal{L}((R_{g,k+1})^m W_g)$ for some $m \leq r$; (ii) there exists a finite path $p = p_0, ..., p_{i+1}$ starting from s and some $n \leq k$ such that

$p_{i+1} \in \rho'_r(x_n)$ and $L(p_0, ..., p_i) \in \mathcal{L}((R_{g,k+1})^m R_{g,n})$ for some $m \leq r$; (iii) there exists a finite path p starting from s such that $L(p) \in R^{r+1}_{g,k+1}$.

Assume that property (i) or property (ii) is satisfied for some r. Consider the smallest value of r, say u, for which either (i) or (ii) is satisfied; in this case, it should be easy to see that (i) or (ii) is satisfied with $m = u$; it is also not hard to see that (i) or (ii) is satisfied for all $r \geq u$; further more, this also implies that there is a finite path p starting from s such that $L(p) \in \mathcal{L}((R_{g,k+1})^u$ and hence property (iii) is satisfied for all $r < u$. Thus, if property (i) or (ii) is satisfied for some r, then for all $r \geq 0$, either (i) or (ii) or (iii) is satisfied. From this and previous observations, we see that $s \in \mathcal{M}_{K,f}(\rho)$, iff for all $r \geq 0$, $s \in \mathcal{M}_{K,g}(\rho'_r)$, iff there exists an $r \geq 0$ such that property (i) or (ii) is satisfied or for all $r \geq 0$ property (iii) is satisfied. Since K is a finite structure, it is not hard to see that property (iii) is satisfied for all $r \geq 0$ iff there exists an infinite path p starting from s such that $L(p) \in \mathcal{L}((R_{g,k+1})^\omega)$. Putting all the above observations together, it is not hard to see that the lemma holds for f using the above definitions for W_f and $R_{f,j}$ for $j = 1, ..., k$.

- The other cases are when f is of the form $f_1 \vee f_2$, and is of the form $P \wedge f_1$ where P is an atomic proposition. Assuming that the lemma holds for f_1 and f_2, it is fairly easy to see that it also holds for f.

■

Now, we continue with the proof of theorem 5.2. Consider any arbitrary closed L_2-formula f. Let $g_1, ..., g_k$ be the maximal, closed, strict sub-formulas of f. (Note that no g_i is a sub-formula of any other g_j). By induction assume that, for each j, $1 \leq j \leq n$, there exists a ECTL* formula $\Theta(g_l)$ such that $K, s \models \Theta(g_j)$ iff $s \in \mathcal{M}_{K,g_j}(\epsilon)$. Let $Q_1, ..., Q_k$ be new atomic proposition symbols not present in $\mathcal{P}$. Let f' be the formula obtained from f by replacing each occurrence g_j by Q_j, for each $j = 1, ..., k$. It is not hard to see that f' is a closed L_1-formula. Now we define a new Kripke K' as follows. $K' = (S, R, L')$ where for each $s \in S$, $L'(s) = L(s) \cup \{Q_j : s \in \mathcal{M}_{K,g_j}(\epsilon)\}$. Note that the states and transitions of K are same as those of K'. It should be easy to see that $\mathcal{M}_{K,f}(\epsilon) = \mathcal{M}_{K',f'}(\epsilon)$.

Define $\mathcal{P}' = \mathcal{P} \cup \{Q_1, ...Q_k\}$ and $\Sigma' = \{\phi : \phi \subseteq \mathcal{P}'\}$. Since f' is a closed formula, from lemma 5.4, we see that there exists a ω-regular expression $W_{f'}$ over the alphabet Σ' such that $s \in \mathcal{M}_{K',f'}$ iff there exists an infinite path p in K' starting from s such that $L'(p) \in \mathcal{L}(W_{f'})$. Let $\phi_1, \phi_2, ..., \phi_u$ be all the elements of Σ'. For each $j = 1, ...u$, define a propositional formula p'_j as follows: $p'_j = p'_{j,1} \wedge p'_{j,2}$ where $p'_{j,1} = \bigwedge_{P \in \phi_j} P$ and $p'_{j,2} = \bigwedge_{P \in \mathcal{P}' - \phi_j} \neg P$. Note that $p'_{j,1}$ is the conjunction of all atomic propositions in ϕ_j and $p'_{j,2}$ is the conjunction of negations of atomic propositions not in ϕ_j, i.e. all those in $\mathcal{P}' - \phi_j$.

Now define $\Theta'(f)$ to be the ECTL* formula $E(W_{f'}(p'_1, .p'_2, ..., p'_k))$. It should be easy to see that for any $s \in S$, $s \in \mathcal{M}_{K',f'}(\epsilon)$ iff the th ECTL* formula $\Theta'(f)$ is satisfied in state s of K', i.e., $K', s \models \Theta'(f)$.

Now, for each $j = 1, ..., u$, define a ECTL* formula $p_j = p_{j,1} \wedge p_{j,2} \wedge p_{j,3} \wedge p_{j,4}$ where each $p_{j,l}$ for $l = 1, ..., 4$ is defined as follows. $p_{j,1} = \bigwedge_{P \in \mathcal{P} \cap \phi_j} P$; $p_{j,2} = \bigwedge_{P \in \mathcal{P} - \phi_j} \neg P$;

$p_{j,3} = \bigwedge_{Q_l \in \phi_j} \Theta(g_l)$; $p_{j,4} = \bigwedge_{Q_l \notin \phi_j} \neg\Theta(g_l)$. Note that $p_{j,3}$ is the conjunction of all $\Theta(g_l)$ such that $Q_l \in \phi_j$, and $p_{j,4}$ is the conjunction of all $\neg\Theta(g_l)$ such that $Q_l \notin \phi_j$.

For each $j = 1, ..., u$, it should be easy to see that $K, s \models p_j$ iff $K', s \models p'_j$. Now, define $\Theta(f)$ to be the ECTL* formula $E(W_{f'}(p_1, p_2, ..., p_u))$. From the previous observation, we see that $K, s \models \Theta(f)$ iff $K', s \models \Theta'(f)$. However, from our earlier observations $K', s \models \Theta'(f)$ iff $s \in \mathcal{M}_{K',f'}(\epsilon)$. Since $\mathcal{M}_{K',f'}(\epsilon) = \mathcal{M}_{K,f}(\epsilon)$, it follows that $K, s \models \Theta(f)$ iff $s \in \mathcal{M}_{K,f}(\epsilon)$. Note that if f is a L_1-formula then $\Theta(f)$ is a formula of the form $E(W(f_1, ..., f_n))$ where $f_1, ..., f_n$ are propositional formulas. This completes the proof of theorem 5.2. ∎

The following corollary is immediate from theorems 5.1 and 5.2.

Corollary 5.1 *The logic L_2 is exactly as expressive as ECTL*, and the logic L_1 is exactly as expressive as the fragment of ECTL* consisting of all formulas of the form $E(W(f_1, ..., f_n))$ where W is a ω-regular expression and $f_1, ..., f_n$ are propositional formulas.*

6 Conclusion

In this paper, we considered the model-checking problem for μ-calculus and have shown it to be equivalent to the emptiness problem for parity tree automata. This shows that there is an efficient algorithm for one if and only there is an efficient algorithm for the other. We have also shown this problem to be in NP∩co-NP.

We also considered two different fragments of μ-calculus, logics L_1 and L_2. We gave model checking algorithms for logics L_1 and L_2 which are of complexity $O(mnp)$ where m is the length of the formula and n is the size of the structure, and p is the alternation depth of the formula. We have shown that the logic L_2 is as expressive as ECTL* given in [19]. In additions to these results, we have shown that the model checking problem for the μ-calculus is equivalent to the non-emptiness problem of parity tree automata.

It will be interesting to investigate if there is a model checking algorithm for the logics L_1 and L_2 which is only of complexity $O(mn)$. Of course, determining if the model checking problem for the full μ-calculus is in P or not, is also an open problem.

References

[1] G. Bhatt, R. Cleaveland, *Efficient Local Model-checking for Fragments of the Modal μ-calculus*, Proceedings of the International Conference on Tools and Algorithms for Construction and Analysis of Systems, Passau, Germany, March 1996; Springer-Verlag Lecture Notes in Computer Science 1055, pp 107-126.

[2] O. Bernholtz, M. Vardi, P. Wolper, *An Automata Theoretic Approach to Branching Time Modelchecking* Proceedings of the 6th International Conference on Computer Aided Verification, CAV '94, Stanford, California, June 1994.

[3] R. Cleaveland, *Tableux-based model checking in the propositional μ-calculus* , Acta Informatica, 27:725-747, 1990.

[4] R. Cleaveland and B. Steffen, *A linear-time model-checking for alternation free modal μ-calculus* , Proceedings of the 3rd workshop on Computer Aided Verification, Aalborg, LNCS, Springer-Verlag, July 1991.

[5] R. Cleaveland and B. Steffen, *Faster model-checking for modal μ-calculus* , Proceedings of the 4th workshop on Computer Aided Verification, Montreal, July 1991.

[6] E. A. Emerson, E. M. Clarke, *Characterizing correctness properties of parallel programs Using Fixpoints*, Proceedings of the International Conference on Automata, Languages and Programming, 1980.

[7] E.A. Emerson and C. S. Jutla, *Tree Automata, Mu-calculus and Determinacy*, Proceedings of the 1991 IEEE Symposium on Foundations of Computer Science.

[8] E. A. Emerson, *Automata, Tableaux, and Temporal Logics*, Proceedings of the Conference on Logics of Programs, Brooklyn College, New York, NY, Springer-Verlag Lecture Notes in Computer Science #193, pp. 79–88, June 1985.

[9] E. A. Emerson, C. Jutla, A. P. Sistla, *On Model-checking for fragments of μ-calculus*, Fifth International Conference on Computer Aided Verification, Elounda, Greece, June/July 1993.

[10] E. A. Emerson and C. Lei, *Efficient model-checking in fragments of μ-calculus* , Proceedings of Symposium on Logic in Computer Science, 1986.

[11] E. A. Emerson and C. Lei, *Modalities for Model Checking*, Science of Computer Programming, 1987.

[12] M. Jurdzinski, *Deciding the winner in parity games is in UP$\cap$co-UP*, Information Processing Letters 68(3), 119-124(1998).

[13] D. Kozen, *Results on the propositional μ-calculus*, Theoretical Computer Science, 27, 1983.

[14] A. Mader, *Verification of Modal Properties Using Boolean Equation Systems*, Munchen Tech-Univ, Dissertation 1997.

[15] A.W. Mostowski, *Regular Expressions for Infinite trees and a standard form of automata*, in: A. Skowron, ed., *Computation Theory*, LNCS, vol 208, 1984, Springer-Verlag.

[16] D. Niwinski, *Fixed-points Vs. Infinite Generation*, Proceedings of the Third IEEE Symposium on Logic in Computer Science, 1988.

[17] C. Stirling, D. Walker, *Local model-checking in modal μ-calculus*, Proceedings of TAPSOFT, 1989.

[18] R. S. Streett and E. A. Emerson, *An automata theoretic decision procedure for Propositional μ-calculus* , Proceedings of the International Conference on Automata, Languages and Programming, 1984.

[19] M. Vardi and P. Wolper, *Yet Another Process Logic*, Proceedings of the workshop on Logics of Programs, Pittsburgh, 1983, also appeared in Lecture Notes in Computer Science.