On model-checking for fragments of μ-calculus

E. A. Emerson A. P. Sistla

February 28, 1995

1 Introduction

In this paper we consider the problem of modelchecking for different fragments of propositional μ-calculus. This logic was studied by many authors [6, 9] for specifying the properties of concurrent programs. It has been shown to be as expressive of automata on infinite trees. Most of the known temporal and dynamic logics can be translated into this logic.

The modelchecking problem for this logic was first considered in [7]. In this paper, the authors presented an algorithm that is $O((mn)^{l+1})$ 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, 10] that were presented. Modelcheckin algorithms for μ-calculus based on Binary Decision Diagrams have been given in [1]. 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 [7, 4] for the case when there is no alternation of least and greatest fixed points in the given formula.

In this paper, we consider the modelchecking problem for different fragments of the μ-calculus. We first consider two fragments called L_1, L_2 and give modelchecking algorithms for thesw fragments which are of complexity $O(m^2n)$ where m is the length of the formula and n is the size of the structure. 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. The fragment L_2 is shown to be exactly as expressive as the branching time temporal logic ECTL* considered in [12]. ECTL* is the extended version of CTL* in which the path formulas have the same expressive power as ω-regular expressions.

We also consider the model checking problem for another fragment L_3. Formulas in L_3 are of the form $\nu y(g)$ where ν is the greatest fixed point operator and g is a formula that does not contain any greatest fixed point operators and in which all negations only apply to atomic formulas only. We show that the modelchecking problem for formulas in L_3 is closely related to the non-emptiness problem for Buchi tree automata. More specifically, we show that the modelchecking problem for this logic is reducible to the non-emptiness problem for Buchi tree automata of size $O(mn)$ where m and n are as defined above. We also show that the non-emptiness problem of Buchi tree automata of size p is reducible to the modelchecking problem for the logic L_3 in which the size of the Kripke structure is $O(p)$ and the length of the formula is a constant. This shows that there is an efficient modelchecking algorithm for L_3 (i.e. an algorithm of complexity less than quadratic complexity in mn) iff there is an efficient algorithm (less than quadratic complexity) for the non-emptiness problem for Buchi tree automata.

The paper is organized as follows. Section 2 contains definitions and notation. Section 3 defines two logics $L1$ and $L2$, and presents the modelchecking algorithms for these logics. Section 4 contains the results for the fragment L3.

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 $P, Q, ...$ 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 well-formed formula in which all the occurences of the variable x are in the scope of an even number of negations then $\mu x(f)$ is also a well-formed formula.

In order to define the semantics of the formulas of μ-calculus, we need the following definitions. For any formula f, we define a finite set of variables, denoted by *free-var*(f), inductively as follows. Intuitively, free-var(f) are all the variables that appear free in f.

- free-var$(P) = \emptyset$ where P is an atomic proposition;
- free-var$(x) = \{x\}$ where x is a variable;
- free-var$(f \wedge g)$= free-var$(f)\cup$free-var(g);
- free-var(<R>f) =free-var$(\neg f)$ =free-var(f);
- free-var$(\mu x f)$ =free-var$(f) - \{x\}$.

If $x \in$free-var(f) then we say that x is a free variable 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 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 $L : S \rightarrow 2^{\mathcal{P}}$. With each state s, L associates a set of atomic propositions that are true in that state. 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 $(s, s') \in R\}$;

- $\mathcal{M}_{(K,\mu x f)}(\rho) = \bigcap \{S' \subseteq S : S' = \mathcal{M}_{(K,f)}(\rho')$ where $\rho'(x) = S'$ and for all other $y \in$ free-var(f), $\rho'(y) = \rho(y)\}$.

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 finite kripke structures, this least fixed point can also be computed by iteration starting with an empty set and iterating until a fixed point is reached. 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 pont operator. It is well known that on finite structures $\nu x(f)$ can be computed by iteration starting with value x equal to S, the set of all states.

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. In our paper we will be interested in these types of formulas. In addition, 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 subformulas 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 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. This is formally stated in the following lemma due to Tarski/Knaster.

Lemma 2.1 *Let $K = (S, R, L)$ be a finite Kripke structure and g be a formula having a free variable x. Let ρ be be an arbitrary evaluation on free-var$(g) - \{x\}$. Define sequences of subests X_j, Y_j as follows: $X_0 = \emptyset, Y_0 = S$; for any $j > 0$, $X_j = \mathcal{M}_{K,g}(\rho'_j), Y_j = \mathcal{M}_{K,g}(\rho''_j)$ where ρ'_j and ρ''_j are extensions of ρ to the free-var(g) such that $\rho'_j(x) = X_{j-1}$ and $\rho''_j(x) = Y_{j-1}$. Then $\mu x(g)$ and $\nu x(g)$ satisfy the following properties.*

- *$M_{K,\mu x(g)}(\rho) = X_i$ where i is the smallest value of j such that $X_j = X_{j+1}$.*
- *$M_{K,\nu x(g)}(\rho) = Y_i$ where i is the smallest value of j such that $X_j = X_{j+1}$.*
- *$M_{K,\mu x(g)}(\rho) \subseteq M_{K,\nu x(g)}(\rho)$.*
- *$M_{K,\nu x(g)}(\rho) = \bigcup\{S : S \subseteq \mathcal{M}_{(K,f)}(\rho')$ where $\rho'(x) = S$ and for all other $y \in$ free-var(f), $\rho'(y) = \rho(y)\}$.*

Now, we define two fragments of the μ-calculus L_1 and L_2 defined as follows. The set of L_1 formulas are exactly those that are formed using the following rules:

1. All the members of $\mathcal{P} \cup \mathcal{X}$ are L_1-formulas; i.e. all atomic propositions and all variables are L_1-formulas.
2. If f is a L_1-formula that does not have any variables appearing in it then $\neg f$ is also an L_1-formula.
3. If f and g are L_1-formulas then $f \vee g$, `<R>`f, $\mu x(f)$ and $\nu x(f)$ are L_1-formulas.
4. If f and g are L_1-formulas such that at most one of them has variables appearing in it, then $f \wedge g$ is a L_1-formula.

Let L_2 be the set of formulas obtained by using rules (a) and (b) in place of 2 and 4.

(a). If f is a closed L_2-formula then $\neg f$ is also an L_2-formula.
(b). If f and g are L_2-formulas such that at most one of them is an open formula then $f \wedge g$ is a L_2-formula.

It is easily seen that rules 2 and 4 are special cases of rules (a) and (b) respectively. As a consequence, L_1 is a subset of L_2.

3 Modelchecking 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 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.

Let f be a closed formula in the logic L_1. Let $SF(f)$ denote the set of subformulas of f. The set $SF(f)$ can be defined inductively. Let $K = (S, R, L)$ be a given kripke structure. We define a 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 subformula of f. The set of edges leaving the node (s, g) are, defined according to the outermost connective of the subformula g, as follows.

- If $g = P$ or $g = \neg P$ where P is an atomic proposition then there are no edges leaving (s, g).
- If $g = x$ where x is variable and g' is the largest subformula 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 $g = \texttt{<R>}g'$, then for each state s' such that $(s, s') \in R$, there is an edge from (s, g) to (s', g').

The following observation follows from the definition of $G_{K,f}$.

Observation 3.1 *Assume that there is an edge from (s, g) to (s', g') in $G_{K,f}$. Then,*

- *If $g = \texttt{<R>}g'$ then $(s, s') \in R$. Otherwise, $s' = s$.*
- *If g is not a variable then g' is a subformula of g. If g is a variable then g is a subformula of g'.*

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 subformula $g \in SF(f)$ is called a μ-subformula (respectively, a ν-subformula) if g is of the form $\mu x(f)$ (respectively, $\nu x(f)$). We say that a cycle C in $G_{K,f}$, is a ν-cycle (respectively, μ-cycle) if the longest subformula appearing in a node on C is ν-subformula (respectively, μ-subformula). A subformula that has no variables appearing in it will be called a *constant*. A node (s, g) in $G_{K,f}$ is called a $\wedge$-node if the outer most connective of g is $\wedge$, i.e. g is of the form $g_1 \wedge g_2$. A node (s, g) is called a $\vee$-node if g is of the form $g_1 \vee g_2$ or is of the form $\texttt{<R>}g_1$.

Lemma 3.1 *1. 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).*

2. Let C be a cycle and (s, g) be a node on it such that g is the longest formula appearing in all the nodes on C. Then, g is a μ-subformula or a ν-subformula. In addition, all other subformulas appearing in some node on C themselves are subformulas of g.

Now, we label the nodes of $G_{K,f}$ as follows. With each node $u \in V$, we maintain a variable $label(u)$ that denotes the label of the node 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 $label(s, g)$ as follows.

   - $g = P$:
     If $P \in L(s)$ *then* $label(s, g) \leftarrow \mathsf{true}$ *else* $label(s, g) \leftarrow \mathsf{false}$.

   - $g = \neg P$:
     If $P \notin L(s)$ *then* $label(s, g) \leftarrow \mathsf{true}$ *else* $label(s, g) \leftarrow \mathsf{false}$.

   - $g = g' \wedge g''$:
     If $label(s, g') = \mathsf{true}$ and $label(s, g'') = \mathsf{true}$ *then* $label(s, g) \leftarrow \mathsf{true}$;
     If $label(s, g') = \mathsf{false}$ or $label(s, g'') = \mathsf{false}$ *then* $label(s, g) \leftarrow \mathsf{false}$.
     In other cases, $label(s, g)$ is unchanged (i.e. $= NIL$).

   - $g = g' \vee g''$:
     If $label(s, g') = \mathsf{true}$ or $label(s, g'') = \mathsf{true}$ *then* $label(s, g) \leftarrow \mathsf{true}$;
     If $label(s, g') = \mathsf{false}$ and $label(s, g'') = \mathsf{false}$ *then* $label(s, g) \leftarrow \mathsf{false}$.
     In other cases, $label(s, g)$ is unchanged.

   - $g = \mathsf{<R>}g'$:
     If $\exists s'$ such that $(s, s') \in R$ and $label(s', g') = \mathsf{true}$ *then* $label(s, g) \leftarrow \mathsf{true}$;

If $\forall s'$ such that $(s,s') \in R$ $label(s',g') =$ false *then* $label(s,g) \leftarrow$ false.
In all other cases, $label(s,g)$ is unchanged ($= NIL$).

- None of the above: $label(s,g)$ is unchanged.

3. *For* each unlabeled node $u \in V$, *if* u lies on an unlabeled ν-cycle *then* $label(u) \leftarrow$ true.
4. *For* each unlabeled node $u \in V$, *if* there exists an unlabeled path from u to a node v satisfying condition (a) or (b) given below *then* $label(u) \leftarrow$ true.
(a) v is a $\wedge$-node and if the two edges leaving v are to the nodes w' and w'' then $label(w') = label(w'') =$ true.
(b) v is not a $\wedge$-node and there exists an edge from v to some node w such that $label(w) =$ true.
5. *For* each unlabeled node u, $label(u) \leftarrow$ false.

It is to be noted that after the execution of step 2 of the above algorithm, the following conditions are satisfied. For all nodes u in $G_{K,f}$ such that there is no path connecting u to a node on a cycle, $label(u) \neq NIL$. From this and lemma 3.1, it automatically follows that, for each node (s,g) in $G_{K,f}$ where g is a constant, $label(s,g) \neq NIL$. Also, for every node (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, $label(u) =$ true and for the other successor v, $label(v) = NIL$.

Theorem 3.1 *After the execution of the above algorithm, for any node $u = (s,g)$ in $G_{K,f}$ where g is a closed subformula, $label(u) =$ true iff $K, s \models g$.*

In order to prove the above theorem, we need some lemmas. 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 (s,g) in $G_{K,f}$ where g is a constant, $label(s,g) \neq NIL$. For the case when g is a constant, it should be easy to see that $label(s,g) =$ true iff $K, s \models g$. Consider any node (s,g) such that $label(s,g) \neq NIL$. For any evaluation ρ over the free variables of g, the following property holds: $s \in \mathcal{M}_{K,g}(\rho)$ iff $label(s) =$ true. For any node (s,g) where g is a μ-formula or a ν-formula, $label(s,g) = NIL$. Also, for every node (s,g) where $g = g' \wedge g''$, for one successor u of the node (s,g), $label(u) =$ true and for the other successor v, $label(v) = NIL$.

Now, we prove that step 3 of the above algorithm is sound. To do this, we need the following lemmas.

Lemma 3.2 *Let $u_0, u_1, ..., u_k$ be a path in $G_{K,f}$ after execution of step 2 of the above algorithm where for each $i = 0, 1, ..., k-1$, u_i is a $\wedge$-node or a $\vee$-node. Also, let $u_i = (s_i, g_i)$. Then, each g_i is a subformula of g_0, and for any evaluation ρ on the free-var(g_0), if $s_k \in \mathcal{M}_{K,g_k}(\rho')$ where ρ' is a restriction of ρ to free-var(g_k), then $s_0 \in \mathcal{M}_{K,g_0}(\rho)$.*

The above lemma can be proved by a simple induction on the length of g_0.

Lemma 3.3 *Let $u_0, u_1, ..., u_k$ be an unlabeled path in $G_{K,f}$ after the execution of step 2 of the above algorithm satisfying the following conditions: for each $i > 0$, $u_i = (s_i, g_i)$, g_i is a strict subformula of g_0 and $g_k \in$free-var(g_0). Then, for any evaluation ρ over free-var(g_0), if $s_k \in \rho(g_k)$ then $s_0 \in \mathcal{M}_{K,g_0}(\rho)$.*

Proof Let $d(g_0)$ be the depth of nesting of the fixed point operators in g_0. We prove the lemma by induction on $d(g_0)$. Note that if g_0 has no μ- or ν-subformulas then $d(g_0) = 0$. The base case for induction is when $d(g_0) = 0$. In this case, g_0 has only the propositional connectives and the lemma follows trivially (note that for any $\wedge$-node on the path, the successor of the node which is not on the path is labeled true). To prove the inductive step, assume that the lemma holds for all cases when $d(g_0) \leq p$. Now, consider the case when $d(g_0) = p + 1$. Let I be the set of all values of i such that $0 \leq i < k$ and g_i is a μ-subformula or a ν-subformula such that $d(g_i) = p + 1$. If I is empty then there is no i such that g_i is a μ- or a ν-subformula and $d(g_i) = p + 1$. In this case, it is easily seen that the inductive hypothesis can be directly applied to prove the induction step. The other case is when I is non-empty. In this case it should be easy to see that for any $i, j \in I$, it has to be the case that $g_i = g_j$. Let $i_0 < i_1 ... < i_q$ be the set of all integers in I and $h = g_{i_0}$. Now, h is of the form $\mu y(h')$ or is of the form $\nu y(h')$ where $d(h') = p$. By applying the induction hypothesis to the path segment between u_{i_q} and u_k, we see that $s_{i_q} \in \mathcal{M}_{K,h'}(\rho')$ where ρ' is an extension of ρ such that $\rho'(y) = \emptyset$. By repeatedly applying the induction hypothesis to each of the path segments between u_{i_j+1} and $u_{i_{j+1}}$ and by using lemma 2.1, it is easy to see that $s_{i_j} \in \mathcal{M}_{K,h}(\rho)$ for j such that $0 \leq j < q$. Also, for each j such that $0 \leq j < i_0$, it is the case that u_j is an $\wedge$-node or $\vee$-node. Using this and the fact that $s_{i_0} \in \mathcal{M}_{K,h}(\rho)$, we conclude from lemma 3.2 that $s_0 \in \mathcal{M}_{K,g_0}(\rho)$. ■

The following lemma shows that step 3 of the above algorithm is sound.

Lemma 3.4 *Let*

- *$u_0, u_1, ..., u_k = u_0$ be an unlabeled ν-cycle in $G_{K,f}$ after execution of step 2 of the above algorithm;*
- *$u_i = (s_i, g_i)$ for $0 \leq i < k$, where $g_0 = \nu x(g')$ is the longest subformula appearing in any node of the cycle;*
- *$S = \{s_i : 0 \leq i < k, g_i = g_0\}$.*

Then, for any evaluation ρ on the free-var(g_0), $S \subseteq \mathcal{M}_{K,g_0}(\rho)$.

Proof Let I be the set of all i such that $0 \leq i < k$ and $g_i = g_0$. Let $0 = i_0 < i_1 < ... < i_p$ be all the members of I. Let ρ' be an extension of rho to free-var(g') such that $\rho'(x) = S$. By applying lemma 3.4 to the path segment between u_{i_r+1} to $u_{i_{r+1}-1}$, for each $r = 0, ..., p-1$, we see that $S \subseteq \mathcal{M}_{K,g'}(\rho')$. Now, applying lemma 2.1, we see that $S \subseteq \mathcal{M}_{K,g_0}(\rho)$. ■

In the full paper we will give the remainder of the proof of theorem 3.1.

Complexity and Expressiveness

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 $O(|f|(|V| + |E|))$. This algorithm works as follows:

For each ν-subformula g of f and in the increasing lengths of g, consider the restriction of $G_{K,f}$ to unlabeled nodes of the form (s, g') where g' is a subformula of g. For each strongly connected component C of the restricted graph, find the type of the longest subformula in any node of C; if this formula is a ν-formula then mark all the nodes of C as nodes lying on a ν-cycle.

In the full paper we will show that the above algorithm correctly identifies all the nodes that lie on unlabelled ν-cycles. Each iteration of the above algorithm can be implemented using an algorithm of complexity $O(|V| + |E|)$. Thus, the overall complexity of step 3 of the main algorithm is $O(|f|(|V| + |E|))$. This will also be the complexity of the overall algorithm. Substituting for $|V|$ and $|E|$ in terms of $|S|$ and $|R|$, we see that the overall complexity of the above algorithm is $O(|f|^2(|S| + |R|))$.

The above algorithm can be naturally be extended to the logic L_2 with the same complexity. We will present this in the full paper. Thus, modelchecking for L_2 can also be done in time $O(|f|^2(|S| + |R|))$.

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* denote the extended version of the logic CTL* where each path formula can be as expressive as ω-regular expressions. The following theorem will be proved in the full paper.

Theorem 3.2 • *The logic L_1 is as expressive as the fragment of ECTL* consisting of formulas of the form $E(p)$ where E is the existential path quantifier and p is a path formula which has no further state formulas with path quantifiers.*

- *The logic L_2 is as expressive as ECTL*.*

We can also use the following alternate approach for model-checking for formulas in L_2. However, this approach will have complexity $O(|f|^3(|S| + |R|))$ which is worse than the complexity of the above method. We briefly describe this procedure. Theorem 3.2 shows that for each formula f in L_1 there exists a formula of the form $E(p)$ in ECTL*, where p is a path formula, such that f is equivalent to $E(p)$. In fact, from f, we can construct a Streett string automaton A_f (see [11]) with the following property: f is satisfied at a node s_0 in a Kripke structure iff there exists an infinite path from s_0 that is accepted by A_f. The number of states in the automaton A_f will be $O(|f|)$ and the number of pairs in the accepting condition will be $O(|f|)$. Now to check if the formula f is satisfied at state s_0 of K, we simply consider K as a string automaton and construct the product automaton of A_f and K and check for non-emptiness of this product automaton. The size of the product automaton ,which is a Streett automaton, will be $O((|S|+|R|)|f|)$ and the number of pairs in the accepting condition will be $O(|f|)$. Checking non-emptiness for this product can be done (using the approach of [8]) and the complexity of the procedure will be $O(|f|^3(|S|+|R|))$.

4 Modelchecking for Other Fragments

In this section we consider other fragments of μ-calculus and explore the relationship between modelchecking for these logics and the checking for emptiness of tree automata.

Specifically, let L_3 be the set of all formulas of μ-calculus which are of the form $\nu y(g)$ where g is in normal form, i.e. all negations are applied to atomic propositions only, and μ is the only fixed point operator that appears in g, i.e. the fixed point operator ν does not appear in g. No further restrictions are placed on the formula g. Note that the model operators `[R]`, `<R>` and the boolean connectives $\wedge$ and $\vee$ can all appear in the formula g without any restrictions.

We will show in this section that the modelchecking problem for the logic L_3 is equivalent under linear reductions to the emptiness problem of Buchi automata on infinite trees. This result shows that there is an efficient model-checking algorithm, i.e. an algorithm that is less than quadratic complexity, iff there an efficient algorithm for checking non-emptiness of Buchi automata on infinite trees. It is to be noted that there is an algorithm for the later problem which has quadratic complexity in the size of the automaton (i.e. the number of states + the number of transitions). There is no known algorithm of better complexity for this problem.

A Buchi automaton A on infinite binary tree 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, $\delta : (Q \times \Sigma) \rightarrow 2^{Q\times Q}$ is the next move relation and $F \subseteq Q$ is the set of final states. 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''. We denote the infinite binary tree by the set $\{0,1\}^*$. An input to the automaton is a marked infinite binary tree which is a function $\tau : \{0,1\}^* \rightarrow \Sigma$. A run of r of A on input τ is a function $r : \{0,1\}^* \rightarrow Q$, associating a state of the automaton with each node of the tree, such that $r(\epsilon) = q_0$, and for any $x \in \{0,1\}^*$ $(r(x0), r(x1)) \in \delta(r(x), \tau(x))$. The run r is accepting if for every infinite sequence σ in $\{0,1\}^*$, there exists infinite number of prefixes of σ, say $\sigma_0, \sigma_1, ...$ such that for each $i \geq 0$ $r(\sigma_i) \in F$.

Lemma 4.1 *Given a kripke structure $K = (S,R,L)$ and a formula $f \in L_3$ and a state s_0, we can obtain a Buchi automaton automaton A of size $O((|S|+|R|)|f|)$ such that A*

accepts at least one input iff $K, s_0 \models f$; in addition, this automaton can be obtained in time $O((|S| + |R|)|f|)$.

Proof We briefly sketch the proof here. Let $f = \nu y(g)$ be the given formula which is in L_3 and K be the given Kripke structure. First we construct the graph $G_{K,f} = (V, E)$ as given in the previous section. Recall that each node in V is of the form (s, h) where $s \in S$ and h is a subformula of f. The edge set E is as defined in the previous section. For example, when h is of the form $\texttt{[R]}h'$, then for each $(s, s') \in R$ there is an edge from (s, h) to (s, h') in E. 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 $\texttt{[R]}h_1$; we call it to be an atomic node if h is P or $\neg P$ for some atomic proposition P; all other nodes in V are called $\vee$-nodes. We make the following assumption. Any non-atomic node has at most two successors, i.e. two edges leaving it in E. If this condition is not satisfied, we can introduce new intermediate nodes and edges so that this property is satisfied; actually, for each node u with k successors, if $k > 2$ then we introduce $k - 2$ additional new nodes and $k - 2$ additional edges. As a consequence, the size of $G_{K,f} = |V| + |E|$ at most doubles. The type of a new node that is introduced in the previous step is same as that of u, i.e. it is a $\wedge$-node if u is a $\wedge$-node, etc.

The states set of the automaton A is simply V, the initial state is (s_0, f), the input alphabet has only one symbol, say symbol a. The transitions and final states of A are defined as follows. The set of final states F is exactly the set of all nodes (s, h) such that $h = y$, or $h = P$ and $P \in L(s)$, or $h = \neg P$ and $P \notin L(s)$. For any node $u = (s, h)$, $\delta(u, a)$ consists of the following pairs: if $h = P$ or $h = \neg P$ then $\delta(u, a) = \{(u, u)\}$; if h is $\vee$-node then $\delta(u, a) = \{(v, v) : (u, v) \in E\}$; if h is a $\wedge$-node then $\delta(u, a) = \{(v, v') : v$ and v' are the successors of $u\}$. It can easily be shown that the automaton accepts at least one input iff $K, s_0 \models f$. It is also not difficult to see that the size of the automaton which is the total number of states plus the number of transitions is $O((|S| + |R|)|f|)$. ■

Lemma 4.2 *Given a Buchi automaton A over infinite binary trees we can obtain a Kripke structure K whose size is linear in the size of A and a formula f in L_3 of constant size and a state s_0 in K, such that A accepts at least one input iff $K, s_0 \models f$.*

Proof First we assume that the alphabet of A is a singleton consisting of the symbol a. We give the informal description of the Kripke structure $K = (S, R, L)$. S has the following elements. We call each element of S as a node and each element of R as an edge. Corresponding to each automaton state s, there is one node in S which is also denoted by s. Corresponding to each pair of states (s_1, s_2), such that $(s_1, s_2) \in \delta(s, a)$ for some state s, S has a node which we denote by the pair (s_1, s_2). R has the following edges. For each s, s_1, s_2 such that $(s_1, s_2) \in \delta(s, a)$, there is an edge from the node s to the node (s_1, s_2) and there are edges from (s_1, s_2) to s_1 and to s_2. There are three atomic propositions denoted by E, A and F. The atomic proposition is E holds exactly in all nodes of the form s, i.e. single states. The atomic proposition A holds exactly in all those nodes that are of the form

(s_1, s_2), i.e. in pairs of states. The atomic proposition F holds exactly in all nodes s where s is a final state of the automaton A.

Let s_0 be the initial state of the automaton. Let f be the formula
$\nu y[E \wedge \texttt{<R>}\mu x(F \wedge y \vee E \wedge \texttt{<R>}x \vee A \wedge \texttt{[R]}x)]$.
It can be shown that $K, s_0 \models f$ iff A accepts at least one input.

It is easy to see that the size of K is linear in the size of A. ■

References

[1] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill and J. Hwang, *Symbolic Modelchecking* 10^{20} *states and beyond*, Proceddings of 5th Annual Symposium on Logic in Computer Science, June 1990.

[2] E. M. Clarke, E. A. Emerson, and A. P. Sistla, Automatic Verification of finite-state Concurrent Systems Using Temporal Logic Specifications, ACM Transactions on Programming Languages and Systems, 8(2):244-263, 1986.

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

[4] R. Cleaveland, 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, 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, *Fixed point characterization properties of parallel programs*, Proceedings of the International Conference on Automata, Languages and Programming, 1980.

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

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

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

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

[11] 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.

[12] 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.