Efficient Local Model-Checking for Fragments of the Modal μ-Calculus

Girish Bhat* **Rance Cleaveland***

ABSTRACT This paper develops efficient local model-checking algorithms for expressive fragments of the modal μ-calculus. The time complexity of our procedures matches that of the best existing global algorithms; however, in contrast to those routines, ours explore a system's state space in a need-driven fashion and do not require its *a priori* construction. Consequently, our algorithms should perform better in practice. Our approach relies on a novel reformulation of the model-checking problem for the modal mu-calculus in terms of checking whether certain linear-time temporal formulas are satisfied by generalized Kripke structures that we call *and-or* Kripke structures.

1 Introduction

Over the last decade *model checking* has emerged as a useful technique for automatically verifying concurrent systems. [1, 4, 8, 15, 19]. In this approach, one attempts to determine whether or not a system satisfies a formula that typically comes from a temporal logic. A variety of different temporal logics have been proposed for this purpose [4, 10, 18]; one particularly expressive one is the modal μ-calculus [13], which is capable of encoding numerous existing temporal logics [12].

When systems are finite-state, mu-calculus model checking becomes decidable; for such systems, a variety of model-checking algorithms have been developed. Two major approaches may be identified. *Global* routines [6, 12, 14] require the *a priori* construction of the entire state space of the system being analyzed; a subsequent pass over the state space then determines the truth or falsity of the formula. Such algorithms typically exhibit good worst-case behavior; however, in practice, the overhead of computing the whole state space is often unnecessary, as the truth or falsity of the property can be deduced from an investigation of a small part

*Research supported by NSF/DARPA grant CCR-9014775, NSF grant CCR-9120995, ONR Young Investigator Award N00014-92-J-1582, NSF Young Investigator Award CCR-9257963, NSF grant CCR-9402807, and AFOSR grant F49620-95-1-0508.

of it. *Local*, or *on-the-fly*, algorithms [1, 5, 16, 20] attempt to remedy this shortcoming by exploring the state space in *demand-driven* fashion. The procedures that have been proposed for the full mu-calculus, however, have uniformly had very poor worst-case behavior in comparison to the global approaches. An efficient algorithm for the alternation-free fragment has been given in [1], but this fragment is incapable of expressing certain fairness constraints that are often needed in practice.

In this paper we present efficient local model-checking algorithms for fragments of the mu-calculus introduced by Emerson et al. in [11]. One of these fragments contains more expressive power than CTL* [10] and hence is capable of encoding properties involving subtle fairness constraints. Our algorithms also have worst-case behavior that matches that of the best existing global model-checking algorithms for these logics [11]. However, since our routines explore state spaces in a need-driven fashion, we expect them to perform much better in practice than the global approaches.

The remainder of this paper is organized as follows. The next section presents the syntax and semantics of μ-calculus and defines the fragments L_1 and L_2 that we consider. In Section 3 we define a variant of traditional Kripke structures that we call *and-or Kripke structures* and give the semantics of a linear-time temporal logic with respect to them. The section following then shows how the model-checking problem for the modal mu-calculus may be reduced to one of model-checking and-or Kripke structures against temporal formulas of a restricted form. Section 5 presents our algorithms; in particular, we explain how the restricted form of L_1 and L_2 may be exploited to give very efficient on-the-fly procedures, and we briefly discuss some implementation issues. The last section contains our conclusions and directions for future research.

2 The Modal μ-calculus

This section presents the syntax and semantics of the modal μ-calculus and defines the sublogics L_1 and L_2. Throughout this section we fix a set $(A, B \in)\mathcal{A}$ of *atomic propositions*.

2.1 Syntax

The syntax of μ-calculus formulas is parameterized with respect to a set $(X, Y \in)\mathcal{V}$ of propositional variables and a set $(a, b \in)Act$ of actions. Formulas are given by the following grammar.

$$\phi \quad ::= \quad A \mid X \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid [a]S \mid \langle a\rangle S \mid \mu X.\phi \mid \nu X.\phi$$

Formulas must also obey the following syntactic restriction: in $\mu X.\phi$ or $\nu X.\phi$, all free occurrences of X in ϕ must fall within the scope of an even

number of negations. We use ϕ, ψ, γ to range over formulas. We refer to $[a]$ and $\langle a \rangle$ operators as *modalities* and to μ and ν as the least and greatest fixpoint operators, respectively, and we call a formula of the form A or $\neg A$ a *literal* and use $(L \in)\mathcal{L}$ to stand for the set of all literals. If a formula has form $\mu X.\phi$ then we sometimes call it a μ-formula, while if it has form $\nu X.\phi$ we refer to it as a ν-formula. In what follows we also assume the usual definitions for (proper, maximal) subformula, for free and bound variable, and for closed formulas, and we write $\phi[\gamma/X]$ to represent the capture-free simultaneous substitution of γ for all free occurrences of X in ϕ. We also introduce the following syntactic normal forms.

Definition 2.1 *Let ϕ be a closed μ-calculus formula.*

1. *ϕ is in positive normal form (PNF) if the only negated subformulas it has are literals.*

2. *ϕ is L_2 normal form (LNF) if every variable is bound at most once and the only negated subformulas it has are closed.*

It is trivial to show that for any closed formula there are semantically equivalent PNF and LNF formulas.

Alternation Levels and the Fischer-Ladner Closure.

We now introduce the notions of *alternation level* and *Fischer-Ladner closure* for LNF formulas. The former notion is defined for the fixpoint subformulas of a given formula; intuitively, it records the number of "alternations" in interdependent fixpoint constructors encountered in the path in the parse tree from the root of the formula to the root of the subformula. To define it precisely, we introduce the following. For formulas ϕ_1 and ϕ_2 let $\phi_1 \preceq \phi_2$ iff ϕ_1 is a subformula of ϕ_2. We also write $\phi_1 \prec_M \phi_2$ iff ϕ_1 is a *maximal* proper subformula of ϕ_2. We use σ to range over $\{\mu, \nu\}$ and $\bar{\sigma}$ to represent the dual of σ.

Definition 2.2 *Let ϕ be a mu-calculus formula in L_2 normal form, with $\psi \equiv \sigma X.\psi' \preceq \phi$. Then $al_\phi(\psi)$ is defined as follows.*

- *If ψ is closed then $al_\phi(\psi) = 1$.*

- *If ψ is not closed then let $\gamma \equiv \sigma' Y.\gamma' \preceq \phi$ be such that $\psi \prec_M \gamma$. If $\sigma' = \bar{\sigma}$ and there exists a $\xi \equiv \sigma''.\xi'$ with $\sigma'' = \bar{\sigma}$, $\gamma \preceq \xi$ and Z appearing free in γ', then $al_\phi(\psi) = 1 + al_\phi(\gamma)$. Otherwise $al_\phi(\psi) = al_\phi(\gamma)$.*

The notion of alternation level can be used to define the more usual one of alternation depth as follows.

Definition 2.3 *Let ϕ be a formula. Then the alternation depth, $ad(\phi)$, of ϕ is given as follows. If ϕ contains fixpoint subformulas $\psi_1, \ldots, \psi_n (n \geq 1)$ then $ad(\phi) = max\{al_\phi(\psi_1), \ldots, al_\phi(\psi_n)\}$. If ϕ contains no fixpoint subformulas, then $ad(\phi) = 0$.*

We refer to a formula ϕ as *alternation-free* when $ad(\phi) \leq 1$. It should be noted that our definition is a slight variant of the usual one given in [12] that corrects a minor anomaly in the treatment of open formulas. The above definition of alternation depth always returns a value less than or equal to the one produced by [12].

We now present the Fisher-Ladner closure [13] for a μ-calculus formula and extend the notion of alternation level to the elements of the closure.

Definition 2.4 *Let ϕ be an μ-calculus formula. Then the Fisher-Ladner closure of ϕ, denoted by $Cl(\phi)$, is the smallest set for which the following hold.*

- $\phi \in Cl(\phi)$.
- *If* $\neg\psi \in Cl(\phi)$ *then* $\psi \in Cl(\phi)$.
- *If* $\psi_1 \vee \psi_2 \in Cl(\phi)$ *or* $\psi_1 \wedge \psi_2 \in Cl(\phi)$ *then* $\psi_1, \psi_2 \in Cl(\phi)$.
- *If* $[a]\psi \in Cl(\phi)$ *or* $\langle a\rangle\psi \in Cl(\phi)$ *then* $\psi \in Cl(\phi)$.
- *If* $\sigma X.\psi \in Cl(\phi)$ *then* $\psi[\sigma X.\psi/X] \in Cl(\phi)$.

The next lemma establishes that there is a one-to-one correspondence between the subformulas of a LNF formula ϕ and $Cl(\phi)$.

Lemma 2.5 *Let ϕ be in LNF, with $\psi \preceq \phi$ and $\gamma \in Cl(\phi)$. There there is a unique $\psi' \in Cl(\phi)$ and substitution ρ such that $\psi' \equiv \psi[\rho]$, and there is a unique $\gamma' \preceq \phi$ and subsitution ρ' such that $\gamma \equiv \gamma'[\rho']$.*

Proof. Follows by induction on the definitions of $\preceq$ and Cl and the fact that variables are bound at most once in LNF formulas.
If $\gamma \in Cl(\phi)$ then we use $s(\phi)$ to denote the subformula of ϕ whose existence is guaranteed by the lemma. We may now define $al_\phi(\gamma)$ as follows.

Definition 2.6 *Let ϕ be in LNF, and let $\gamma \in Cl(\phi)$. Then $al_\phi(\gamma) = al_\phi(s(\gamma))$.*

2.2 Semantics

Labeled transitions systems are used to interpret μ-calculus formulas.

Definition 2.7 *A labeled transition system is a quadruple $\langle \mathcal{S}, Act, \rightarrow, I\rangle$, where*

$$
\begin{aligned}
[\![A]\!]_{\mathcal{T}}e &= \mathcal{V}(A) \\
[\![X]\!]_{\mathcal{T}}e &= e(X) \\
[\![\neg\phi]\!]_{\mathcal{T}}e &= \mathcal{S} - [\![\phi]\!]_{\mathcal{T}}e \\
[\![\phi_1 \vee \phi_2]\!]_{\mathcal{T}}e &= [\![\phi_1]\!]_{\mathcal{T}}e \cup [\![\phi_2]\!]_{\mathcal{T}}e \\
[\![\phi_1 \wedge \phi_2]\!]_{\mathcal{T}}e &= [\![\phi_1]\!]_{\mathcal{T}}e \cap [\![\phi_2]\!]_{\mathcal{T}}e \\
[\![\langle a\rangle\phi]\!]_{\mathcal{T}}e &= \{\, s \mid \exists s'.\ s \xrightarrow{a} s' \wedge s' \in [\![\phi]\!]_{\mathcal{T}}e \,\} \\
[\![[a]\phi]\!]_{\mathcal{T}}e &= \{\, s \mid \forall s'.\ s \xrightarrow{a} s' \Rightarrow s' \in [\![\phi]\!]_{\mathcal{T}}e \,\} \\
[\![\mu X.\phi]\!]_{\mathcal{T}}e &= \bigcup\{\, S \subseteq \mathcal{S} \mid S \subseteq [\![\phi]\!]_{\mathcal{T}}e[X \to S] \,\} \\
[\![\nu X.\phi]\!]_{\mathcal{T}}e &= \bigcap\{\, S \subseteq \mathcal{S} \mid [\![\phi]\!]_{\mathcal{T}}e[X \mapsto S] \subseteq S \,\}
\end{aligned}
$$

FIGURE 1. Semantics of formulas for $\mathcal{T} = \langle \mathcal{S}, Act, \rightarrow, I\rangle$.

- $\mathcal{S}$ *is a set of states;*
- *Act is a set of actions;*
- $\rightarrow \subseteq \mathcal{S} \times Act \times \mathcal{S}$ *is the transition relation; and*
- $I \in \mathcal{A} \rightarrow 2^{\mathcal{S}}$ *is the interpretation.*

If $\mathcal{T} = \langle \mathcal{S}, Act, \rightarrow, I\rangle$ *is a labeled transition system and* $s \in \mathcal{S}$*, then we refer to the pair* $\langle \mathcal{T}, s\rangle$ *as a labeled transition* structure*; in this case, we call* s *the start state.*

Intuitively, a labeled transition system encodes the operational behavior of a system, with $\mathcal{S}$ containing the possible system states, *Act* the actions the system may engage in, $\rightarrow$ the transitions between states that occurs as a result of execution of actions, and I indicating which states a given atomic proposition is true in. A labeled transition system additionally contains a designated start state. Following traditional usage we write $s \xrightarrow{a} s'$ in lieu of $\langle s, a, s'\rangle \in \rightarrow$ and call s' an *a-derivative* of s. When $|S| < \infty$ and $|Act| < \infty$ we call labeled transition system $\langle S, Act, \rightarrow, I\rangle$ *finite-state.*

The semantics of μ-calculus formulas shown in Figure 2.2 is given with respect to a labeled transition system $\mathcal{T} = \langle S, Act, \rightarrow, I\rangle$ and an environment $e : \mathcal{V} \rightarrow 2^{\mathcal{S}}$. The environment $e[X \mapsto S]$ is the environment obtained from e by updating X to S. Intuitively, the semantics maps a formula to a set of states for which the formula holds. Accordingly, the meaning of an atomic proposition is given by I, and the meaning of a propositional variable is the set of states bound to it by the environment e. The boolean constructs are interpreted in the usual fashion. The meaning of $[a]\phi$ contains the set of states all of whose the a-derivatives satisfy ϕ. Similarly, $\langle a\rangle\phi$ represents the set of states for which there is some a-derivative satisfying ϕ.

The semantics of $\mu X.\phi$ and $\nu X.\phi$ are taken to be the least and greatest fixpoints of the function $\phi_e(S) = [\![\phi]\!]_{\mathcal{T}} e[X \mapsto S]$ repectively. The existence of these fixpoints is guaranteed by the monotonicity of ϕ_e over the lattice of sets of states and the Tarski Fixpoint Theorem [17].

We now define what it means for a labeled transition structure to satisfy a formula.

Definition 2.8 *Let $\langle T, s\rangle$ be a labeled transition structure and e an environment. Then $s \models^e_T \phi$ iff $s \in [\![\phi]\!]_{\mathcal{T}} e$.*

If ϕ is closed then the environment e does not influence $[\![\phi]\!]_{\mathcal{T}} e$. In this case we write $s \models_{\mathcal{T}} \phi$ when $s \models^e_T \phi$ holds for some (hence any) e.

2.3 The L_1 and L_2 Sublogics

We now present the syntax of two fragments of μ-calculus. The first, L_1, is the set of formulas formed by the following rules.

1. All atomic propositions and variables are elements of L_1.
2. If ϕ_1,ϕ_2 are in L_1 then
   (a) $\phi_1 \vee \phi_2$, $\langle a\rangle\phi_1$, $\mu X.\phi$ and $\nu X.\phi$ are in L_1.
   (b) $\neg\phi$ is in L_1 provided that ϕ is atomic.
   (c) $\phi_1 \wedge \phi_2$ is in L_1 provided ϕ_1 is a literal.
   (d) $[a]\phi_1$ is in L_1 provided ϕ_1 is a literal.

It should be noted that this definitions differs slightly from the one given in [11]; the difference, however, is insignificant for the purposes of this paper.

To obtain L_2 we modify rules 2(b), 2(c) and 2(d) by replacing "atomic" and "literal" with "closed formula". Note that L_1 is a sublogic of L_2. It is also straightforward to establish that for any formula in L_2, there is an equivalent LNF formula that is also in L_2 (hence the motivation for LNF). The same does not hold for PNF; in general, L_2 formulas do not have PNF equivalents that are also in L_2.

The expressiveness of L_2 has been studied by Emerson et al. [11]; in particular they have shown that it has the same expressive power as Wolper's ECTL* [18] and hence is strictly more expressive than CTL* [10].

3 And-Or Kripke Structures and Temporal Logic

In this paper we wish to present algorithms for solving the model-checking problem for closed formulas in L_1 and L_2 interpreted over finite-state labeled transition structures. This problem may be phrased as follows.

Given L_1/L_2 formula ϕ and labeled transition structure $\langle T, s\rangle$, does $s \models_T \phi$?

Our approach uses this general strategy.

1. From $\langle T, s\rangle$ and ϕ, generate an intermediate structure representing the possible "attempted proofs" that $s \models_T \phi$.
2. Check whether one of the attempted proofs is valid.

It turns out that the construction of the intermediate structure can be combined with the check for validity, thereby yielding an on-the-fly algorithm.

In this section, we introduce the intermediate structures used in our methodology. They resemble the traditional Kripke structures used in defining the semantics of temporal logics; the main difference is that the underlying graph structure is an *and-or* graph. Hence we call these structures *and-or Kripke structures*. We also show how a linear-time temporal logic may be interpreted over these structures; we use this logic to define the "validity check" referred to above.

3.1 And-Or Kripke Structures

And-or Kripke structures may be defined formally as follows.

Definition 3.1 *An and-or Kripke structure is a tuple* $\langle \mathcal{Q}, \mathcal{A}, R, L, P, q_0\rangle$ *where*

- $\mathcal{Q}$ *is a set of* states*;*
- $\mathcal{A}$ *is a set of* atomic propositions*;*
- $R \subseteq \mathcal{Q} \times \mathcal{Q}$ *is the* transition relation*, which is total: for every* $q \in \mathcal{Q}$ *there must exist a* $q' \in \mathcal{Q}$ *with* $\langle q, q'\rangle \in R$*;*
- $L : \mathcal{Q} \to 2^{\mathcal{A}}$ *is the* propositional labeling*;*
- $P : \mathcal{Q} \to \{\vee, \wedge\}$ *is the* and-or labeling*; and*
- $q_0 \in \mathcal{Q}$ *is the start state.*

If $P(s) = \vee$ then we sometimes refer to s as a $\vee$-state, and similarly for $\wedge$.

And-or Kripke structures differ from traditional Kripke structures in the inclusion of the and-or labeling P. In a traditional Kripke structure, an execution, or *run*, of the system is typically defined as a maximal sequence of states $q_0 q_1 \ldots$ where $\langle q_i, q_{i+1}\rangle \in R$. In an and-or Kripke structure, a run will instead be a *tree* of states, with $\wedge$-states having multiple successors, in general. This intuition is captured by the following definition.

Definition 3.2 *Let* $\mathcal{K} \equiv \langle \mathcal{Q}, \mathcal{A}, R, L, P, q_0\rangle$ *be an and-or Kripke structure. Then a* run *of* $\mathcal{K}$ *is a maximal (hence infinite) tree with nodes labeled by elements of* $\mathcal{Q}$ *that satisfies the following properties.*

- *The root of the tree is labeled by* q_0.
- *Let* σ *be a node labeled by* q.
  - *If* $P(q) = \wedge$ *and* $\{q' \mid \langle q, q' \rangle \in R\} = \{q_1, \ldots q_m\}$ *then* σ *has exactly* m *successors* $\sigma_1, \ldots, \sigma_m$, *with* σ_i *labeled by* q_i.
  - *If* $P(q) = \vee$ *then* σ *has exactly one successor,* σ', *and* σ' *is labeled by some* q' *such that* $\langle q, q' \rangle \in R$.

We use $R(\mathcal{K})$ *to represent the set of all runs of* $\mathcal{K}$.

Note that if an and-or Kripke structure contains only $\vee$-states, then its runs are sequences; in this case, the notion of run coincides exactly with the one found for traditional Kripke structures.

We also use the following notions in the rest of the paper.

Definition 3.3 *Let* $\mathcal{K} \equiv \langle \mathcal{Q}, \mathcal{A}, R, L, P, q_0 \rangle$ *be an and-or Kripke structure.*

- *A* path *through* $\mathcal{K}$ *is a maximal sequence* $q'_0 q'_1 \ldots$ *such that* $\langle q'_i, q'_{i+1} \rangle \in R$.
- *Let* $r \in R(\mathcal{K})$. *Then* $\pi(r)$, *the* paths through r, *contains all sequences of the form* $q'_0 q'_1 \ldots$, *where* q'_0 *labels the root of* r *and* q'_{i+1} *labels a successor* σ_{i+1} *of* σ *in* r *if* q'_i *labels node* σ_i *in* r.

And-or Kripke structures may also be viewed as variations on amorphous alternating tree automata [2], the main difference being that tree automata have an explicit acceptance condition used for defining runs and have propositional labelings on their transitions rather than states.

3.2 A Linear Temporal Logic

We now introduce a simple linear-time temporal logic and show how formulas may be interpreted with respect to and-or Kripke structures. The semantics of the logic, which we call LTL, is given as follows, where $(A \in)\mathcal{A}$ is assumed to be a set of atomic propositions.

$$\phi \ ::= \ A \mid \phi \wedge \phi \mid \phi \vee \phi \mid F\phi \mid G\phi$$

The boolean operations are interpreted in the usual manner, while F and G represent the standard "eventually" and "henceforth" operators.

Traditionally, LTL formulas are interpreted with respect to paths in Kripke structures. We recall the definition here.

Definition 3.4 *Let* $\mathcal{K} \equiv \langle \mathcal{Q}, \mathcal{A}, R, L, P, q_0 \rangle$ *be an and-or Kripke structure, let* $x \equiv q'_0 q'_1 \ldots$ *be a path in* $\mathcal{K}$, *and let* ϕ *be an LTL formula. Then* $x \models_{\mathcal{K}} \phi$ *is defined inductively as follows.*

- $x \models_{\mathcal{K}} A$ *iff* $A \in L(q'_0)$.
- $x \models_{\mathcal{K}} \phi_1 \wedge \phi_2$ *iff* $x \models_{\mathcal{K}} \phi_1$ *and* $x \models_{\mathcal{K}} \phi_2$.
- $x \models_{\mathcal{K}} \phi_1 \vee \phi_2$ *iff* $x \models_{\mathcal{K}} \phi_1$ *or* $x \models_{\mathcal{K}} \phi_2$.
- $x \models_{\mathcal{K}} F\phi$ *iff there is a suffix* $x^i = q'_i q'_{i+1} \ldots$ *of* x *such that* $x^i \models_{\mathcal{K}} \phi$.
- $x \models_{\mathcal{K}} G\phi$ *iff for every suffix* $x^i = q'_i q'_{i+1} \ldots$ *of* x, $x^i \models_{\mathcal{K}} \phi$.

We may now extend the notion of $\models_{\mathcal{K}}$ to *runs* of $\mathcal{K}$ as follows.

Definition 3.5 *Let $\mathcal{K}$ be an and-or Kripke structure with $r \in R(\mathcal{K})$, and let ϕ be an LTL formula. Then $r \models_{\mathcal{K}} \phi$ iff for every $x \in \pi(r), x \models_{\mathcal{K}} \phi$.*

Finally, we may identify two different ways in which an and-or Kripke structure satisfies an LTL formula.

Definition 3.6 *Let $\mathcal{K}$ be an and-or Kripke structure, and let ϕ be an LTL formula. Then:*

- $\mathcal{K} \models_\exists \phi$ *iff there is an* $r \in R(\mathcal{K})$ *such that* $r \models_{\mathcal{K}} \phi$.
- $\mathcal{K} \models_\forall \phi$ *iff for every* $r \in R(\mathcal{K})$, $r \models_{\mathcal{K}} \phi$.

4 μ-Calculus Model Checking via And-Or Kripke Structures

We now show how model-checking for the general μ-calculus can be reduced to the model-checking problem for LTL interpreted over and-or Kripke structures. The reduction proceeds as follows.

1. We give a set of "proof rules" for establishing that a labeled transition structure satisfies a μ-calculus formula in PNF.
2. We then show how one may use the rules to generate an and-or Kripke structure from a labeled transition structure and μ-calculus formula.
3. Finally, we describe how to build a formula in LTL that is satisfied by the and-or Kripke structure if and only if the labeled transition structure satisfies the original μ-calculus formula.

The proof rules for inferring if a labeled transition structure satisfies a PNF formula are given in Figure 5. They work on *assertions* of form $s \vdash_{\mathcal{T}} \phi$; intuitively, $s \vdash_{\mathcal{T}} \phi$ represents the statement that transition structure $\langle \mathcal{T}, s \rangle$ satistifies ϕ. In what follows we use $(\sigma, \sigma' \in)\Sigma$ to refer to the set of all assertions. The proof rules are also goal-directed, meaning that given an assertion to be proved, an application of a proof rule yields subassertions to be proved. The following lemma establishes the soundness of the rules.

$$\vee \frac{s \vdash_{\mathcal{T}} L}{true} \quad (s \models_{\mathcal{T}} L)$$

$$\wedge \frac{s \vdash_{\mathcal{T}} \phi_1 \wedge \phi_2}{s \vdash_{\mathcal{T}} \phi_1 \quad s \vdash_{\mathcal{T}} \phi_2} \qquad \vee \frac{s \vdash_{\mathcal{T}} \phi_1 \vee \phi_2}{s \vdash_{\mathcal{T}} \phi_1 \quad s \vdash_{\mathcal{T}} \phi_2}$$

$$\wedge \frac{s \vdash_{\mathcal{T}} [a]\phi}{s_1 \vdash_{\mathcal{T}} \phi, .., s_m \vdash_{\mathcal{T}} \phi} \{s_1, \ldots, s_m\} = \{s' | s \xrightarrow{a} s'\}$$

$$\vee \frac{s \vdash_{\mathcal{T}} \langle a \rangle \phi}{s_1 \vdash_{\mathcal{T}} \phi, .., s_m \vdash_{\mathcal{T}} \phi} \{s_1, \ldots, s_m\} = \{s' | s \xrightarrow{a} s'\}$$

$$\vee \frac{s \vdash_{\mathcal{T}} \mu X.\phi}{s \vdash_{\mathcal{T}} \phi[\mu X.\phi / X]} \qquad \vee \frac{s \vdash_{\mathcal{T}} \nu X.\phi}{s \vdash_{\mathcal{T}} \phi[\nu X.\phi / X]}$$

FIGURE 2. Proof rules for the μ-calculus, where $\mathcal{T} = \langle \mathcal{S}, Act, \rightarrow, I \rangle$.

Lemma 4.1 *Let $\sigma \equiv s \vdash_{\mathcal{T}} \phi$ be an assertion.*

1. *If the subgoals resulting from applying a rule to σ have the form $s_1 \vdash_{\mathcal{T}} \phi_1, \ldots, s_m \vdash_{\mathcal{T}} \phi_m$, then $s \models_{\mathcal{T}} \phi$ iff $s_i \models_{\mathcal{T}} \phi_i$ for each i.*
2. *If the subgoal resulting from applying a rule to σ is* true *then $s \models_{\mathcal{T}} \phi$.*
3. *If no rule can be applied to σ then $s \not\models_{\mathcal{T}} \phi$.*

Proof. Follows from the semantics of μ-calculus formulas.
We now introduce the notion of *proof structure*.

Definition 4.2 *Let $V \subseteq \Sigma \cup \{true\}$, $E \subseteq V \times V$ and $\sigma \in \Sigma$. Then $\langle V, E \rangle$ is a proof structure for σ if $\sigma \in V$ and V and E are maximal sets satisfying the following for every $\sigma' \in V$: σ' is reachable from σ using edges in E, and the set $\{\sigma'' | \langle \sigma', \sigma'' \rangle \in E\}$ is the result of applying a rule from Figure 5 to σ'.*

Note that a proof structure for a given σ is unique, since at most one rule is applicable to any assertion. Intuitively, a proof structure for σ in intended to encode all possible ways of "proving" that σ holds. A "candidate proof" may be obtained from a proof structure by removing all but one outgoing edge from all assertions to which a rule labeled by $\vee$ has been applied. It is also the case that proof structures may contain cycles, owing to the presence of the fixpoint operators in the logic; this fact complicates a determination of when a proof structure for σ contains a "valid proof" of σ. In traditional proof theory, circular reasoning is always deemed incorrect; in such a setting, a proof for σ that contains a cycle could not be used as evidence of the truth of σ. However, in the μ-calculus, while one may not

use such "circular reasoning" to establish that $s \vdash_{\mathcal{T}} \mu X.\phi$ holds, one may use it in order to prove $s \vdash_{\mathcal{T}} \nu X.\phi$. Consequently, in order to determine if a proof structure contains a valid proof, one should permit cycles in which the "top-level" formula in a ν-formula. More specifically, a cycle is allowed in a proof if the formulas with the lowest alternation level on the cycle are ν-formulas. Our approach to checking this condition is as follows.

1. Represent the proof structure for an assertion σ as an and-or Kripke structure whose runs are attempted proofs of σ; and

2. devise an LTL formula that holds of runs whose paths satisfy the "good cycles" condition.

In order to determine if a proof structure contains a proof for σ, one would then check whether the resultant and-or Kripke structure satisfies the LTL formula.

To convert a proof structure $\langle V, E \rangle$ for assertion $\sigma \equiv s \vdash_{\mathcal{T}} \phi$ into an and-or Kripke structure, we must specify the set of states $\mathcal{Q}$, the set of atomic propositions $\mathcal{A}$, the transition relation R, the propositional labeling L, the and-or labeling P, and the start state q_0. Most of these are straightforward.

- For the state set, take $\mathcal{Q} = V$.

- For the transition relation R it is tempting to take E. However, R is required to be total, while E may not be (there may be "leaves" in the proof structure). To handle this, we extend E to a total relation by adding self loops to every leaf. Formally,

$$R = E \cup \{ \langle \sigma', \sigma' \rangle \mid \forall \sigma'' \in V. \langle \sigma', \sigma'' \rangle \notin R \}$$

- As remarked above, for any given assertion at most one rule is applicable. So we define $P(\sigma)$ to be the label of the rule in Figure 5 applicable to σ, if such a rule exists; if no such rule exists, we take $P(\sigma)$ to be $\vee$.

- For the start state, we take $q_0 = \sigma$.

In order to complete our definition, we need to specify $\mathcal{A}$ and L. Our intention is to use the atomic propositions in order to record the alternation level of fixpoint formulas contained in assertions; accordingly, we the atomic propositions to be of form ν_i and $\mu_{>i}$, where $i \leq ad(\phi)$ (recall that ϕ is the formula in the "root assertion" in the proof structure). Formally, if ϕ is the μ-calculus formula being checked then the set of atomic propositions $\mathcal{A} = \{true\} \cup \{ \nu_i \mid i \leq ad(\phi) \} \cup \{ \mu_{>i} \mid i \leq ad(\phi) \}$. The function L can now be defined as follows, where $\sigma' \equiv s' \vdash_M \phi'$. $\nu_i \in L(\sigma')$ iff ϕ' is a ν-formula and $al_\phi(\phi') = i$. $\mu_{>i} \in L(\sigma')$ iff either ϕ' is a non-μ-formula or ϕ' is a μ-formula and $al_\phi(\phi') > i$. If σ' is of form $true$, then $L(\sigma) = \{true\}$.

$$\vee \frac{s \vdash_T \phi_1 \wedge \phi_2}{s \vdash_T \phi_2} (s \models_T \phi_1)$$

$$\vee \frac{s \vdash_T [a]\phi}{true} (\forall s'' \in \{ s' \mid s \xrightarrow{a} s' \}. s'' \models_T \phi)$$

FIGURE 3. Modified proof rules for L_1.

The following theorem states that the μ-calculus model-checking problem may be reduced to checking specific LTL formulas on and-or Kripke structures.

Theorem 4.3 *Let $\mathcal{K}_\sigma$ be the and-or Kripke structure corresponding to $\sigma \equiv s \vdash_T \phi$. Then $s \models_M \phi$ iff $\mathcal{K}_\sigma \models_\exists F(true) \vee \bigvee_{i=1}^{ad(\phi)} (GF\nu_i \wedge FG\mu_{>i})$.*

The proof of this theorem appears in an appendix, but the intuition is as follows. Suppose that $\mathcal{K}_\sigma$, where $\sigma \equiv s \vdash_T \phi$, contains a run r satisfying $GF\nu_i \wedge FG\mu_{>i}$ for some $i \leq ad(\phi)$. Suppose further that the run contains a cycle. It then follows that the run contains a path that traverses this cycle an infinite number of times. In order for this path to satisfy this formula, it must follow that some ν-formula of level i appears infinitely often while μ-formulas of level $\leq i$ can only appear finitely often. This implies that the cyclic reasoning implicit in the cycle is being used in support a ν-formula and hence that the cycle is allowable. Similarly, if all cycles in a run satisfy the "good cycle" condition then given LTL formula must hold the infinite paths in the run.

5 Model Checking for L_1 and L_2

The previous characterization of μ-calculus model checking in terms of and-or Kripke structures does not by itself suggest efficient algorithms for determining whether states satisfy formulas in the full calculus. However, when the formulas are in L_1 or L_2, it turns out that the and-or Kripke structures have a special structure that permits them to be manipulated efficiently. In the remainder of this section we use these facts to develop local model-checking algorithms for these logics.

5.1 Efficient L_1 Model Checking

Recall that the syntactic restrictions on the L_1 sublogic stipulate that in formulas of the form $\phi_1 \wedge \phi_2$ and $[a]\phi$, ϕ_1 and ϕ must be literals. These facts imply that proof structures involving formulas of these types have a restricted form. In particular, in a structure built for assertion $s \vdash_T \phi_1 \wedge \phi_2$, the left child of the root ($s \vdash_T \phi_1$, where ϕ_1 is atomic) is either a leaf if

$s \not\models_{\mathcal{T}} \phi_1$ or has the leaf *true* as its only child otherwise. A similar property holds for all children of $s \vdash_{\mathcal{T}} [a]\phi$ when ϕ is atomic; by only looking at the a-derivatives of s and no other states, one may determine if a successful proof structure may be constructed for this assertion.

Using these observations, we may alter the proof rules for the full μ-calculus to obtain the ones given in Figure 3. What is noteworthy about these is the absence of rules labeled by $\wedge$; instead, side conditions are used to handle the left conjuncts in conjunctions and formulas using the $[a]$ modality.

From the definition of the procedure for constructing and-or Kripke structures from models and μ-calculus formulas, it immediately follows that if ϕ is in L_1 then the and-or Kripke structure for assertion $s \vdash_{\mathcal{T}} \phi$ contains no $\wedge$-states. This implies that all the runs of this structure degenerate to sequences of states, as branching in a run arises only from $\wedge$-states. Therefore, checking if there exists a run which satisfies an LTL formula is equivalent to checking if there exists a path satisfying the formula. As we observed before, this is the LTL model-checking problem for traditional Kripke structures, for which an efficient on-the-fly algorithm has been developed [3]. As the proof rules permit us to construct the Kripke structure on-the-fly, we therefore obtain an on-the-fly model-checking routine for L_1.

Time Complexity

In order to quantify the time complexity of this algorithm we first characterize the size of the proof structure yielded by applying the rules in Figure 3. First, note that in any assertion $s' \vdash_{\mathcal{T}} \phi'$ found in a proof structure for $s \vdash_{\mathcal{T}} \phi$, $\phi' \in Cl(\phi)$. This leads the following result.

Theorem 5.1 *Let $\langle \mathcal{T}, s \rangle$ be a labeled transition structure, let ϕ a L_1 formula, and let $\langle V, E \rangle$ be a proof structure for $s \vdash_{\mathcal{T}} \phi$. Then $|V| + |E| \leq |\phi| * |\mathcal{T}|$.*

Also observe that the formula we give in Theorem 4.3 is $O(ad(\phi))$ in size. Consequently, as the LTL model-checking algorithm in [3] has complexity $2^{O(|\psi|)} * |\mathcal{K}|$ for a Kripke structure $\mathcal{K}$ and LTL formula ψ, one would expect the complexity of our algorithm for L_1 formula ϕ and labeled transition structure $\langle \mathcal{T}, s \rangle$ to be $2^{O(|ad(|\phi|)|)} * |\phi| * |\mathcal{T}|$. However, recall that formula under consideration has the form $Ftrue \vee \bigvee_{i=1}^{ad(\phi)} (GF\nu_i \wedge FG\mu_{>i})$, where *true* and the ν_i and $\mu_{>i}$ are atomic propositions. It can be shown that for formulas in this form the algorithm in [3] takes time in $O(ad(\phi) * c)$ where c is constant. Intuitively, this is due to the observation that to check whether a there is a path in a Kripke structure satisfying $(\bigvee_{i=1}^{n} (GFp \wedge FGq))$, it suffices to check each disjunct in isolation. As a result our model-checking algorithm for the logic L_1 has time complexity $O(ad(\phi) * |\phi| * |\mathcal{T}|)$.

$$\vee \frac{s \vdash_{\mathcal{T}} \neg\phi}{true} (s \not\vdash_{\mathcal{T}} \phi)$$

FIGURE 4. Additional rule for L_2.

5.2 *Efficient L_2 Model Checking*

The model checker for L_2 uses the same essential observations as the one given above for L_1. Indeed, the same proof rules are used (with one rule, given in Figure 4, to handle negation). However, the side conditions in the rules for propositions of the form $\neg\phi$, $\phi_1 \wedge \phi_2$ and $[a]\phi$ are nontrivial to handle because ϕ_1 and ϕ are no longer assumed to be atomic; in L_2 they are only required to be closed. We may nevertheless exploit the following observations. Suppose we wish to build a proof structure for an assertion σ using the proof rules for the full μ-calculus, and suppose further $s' \vdash_{\mathcal{T}} \phi_1 \wedge \phi_2$ is an assertion in the proof structure and that ϕ_1 is closed. It then follows that the substructure computed for $s' \vdash_{\mathcal{T}} \phi_1$ will have no edges leading to any other part of the proof structure; in other words, $s' \vdash_{\mathcal{T}} \phi_1$ may be checked independently. Our L_2 model checker thus handles the side conditions for the negation rule and for the former $\wedge$-rules by invoking itself recursively on them. The resulting algorithm may be shown also to have time complexity $O(ad(\phi) * |\phi| * |\mathcal{T}|)$ using a simple induction on the structure of formulas.

5.3 *Implementation Concerns*

We have implemented our algorithm in the NCSU Concurrency Workbench, which is a re-implementation of the tool for analyzing concurrent systems described in [7], and have experimented with the implementation using several small and medium-sized (up to 5,000 states) examples, including a train signalling scheme. In its current prototype form, and running on a Sun SparcStation 20 with 512 MB of memory, the implementation seems to be capable of processing roughly 1,000 states per minute.

However, a number of tricks can be used to improve the performance of the algorithm. Firstly, we are explicitly calling an implementation of the LTL model checker of [3] in our μ-calculus model checker. However, since the LTL formulas we use have an extremely restricted form, ***partially evaluating*** the LTL model checker with respect to these formulas would yield a substantial time and space savings. Secondly, the LTL formulas being checked can be "optimized" to yield formulas for which the (partially evaluated) model checker exhibits better behavior. For example, the LTL formula $FGp \wedge GFq$, where p and q are atomic, is logically equivalent to $FG(p \wedge Fq)$; the latter, however, is much more efficient to process in the scheme of [3] than the former.

6 Conclusion

In this paper we have presented efficient on-the-fly model-checking algorithms for fragments of the modal μ-calculus. In contrast with existing algorithms for these logics [11] our routines construct the system state space in a demand-driven manner; the fragments are also capable of expressing fairness constraints that are beyond the expressive power of fragments of the μ-calculus for which efficient on-the-fly algorithms have been developed [1]. Our approach relies on a reduction of the model-checking problem for the full μ-calculus to the model checking problem for LTL interpreted over novel structures that we call and-or Kripke structures. We have also implemented our algorithms in the Concurrency Workbench [7], a tool for analyzing concurrent systems.

7 References

[1] H.R. Andersen. Model checking and boolean graphs. In *Proceedings of the European Symposium on Programming*, volume 582 of *Lecture Notes in Computer Science*, pages 1–19, Rennes, France, March 1992. Springer-Verlag.

[2] O. Bernholtz, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In Dill [9], pages 142–155.

[3] G. Bhat, R. Cleaveland, and O. Grumberg. Efficient on-the-fly model checking for CTL*. In *Tenth Annual Symposium on Logic in Computer Science (LICS '95)*, pages 388–397, San Diego, July 1995. IEEE Computer Society Press.

[4] 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, April 1986.

[5] R. Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27(8):725–747, September 1990.

[6] R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal mu-calculus. In G.v. Bochmann and D.K. Probst, editors, *Computer Aided Verification (CAV '92)*, volume 663 of *Lecture Notes in Computer Science*, pages 410–422, Montréal, June/July 1992. Springer-Verlag.

[7] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: A semantics-based tool for the verification of finite-state systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.

[8] R. Cleaveland and B. Steffen. A linear-time model-checking algorithm for the alternation-free modal mu-calculus. *Formal Methods in System Design*, 2:121–147, 1993.

[9] D.L. Dill, editor. *Computer Aided Verification (CAV '94)*, volume 818 of *Lecture Notes in Computer Science*, Stanford, California, June 1994. Springer-Verlag.

[10] E.A. Emerson and J.Y. Halpern. 'Sometime' and 'not never' revisited: On branching versus linear time temporal logic. *Journal of the Association for Computing Machinery*, 33(1):151–178, 1986.

[11] E.A. Emerson, C. Jutla, and A.P. Sistla. On model-checking for fragments of μ-calculus. In C. Courcoubetis, editor, *Computer Aided Verification (CAV '93)*, volume 697 of *Lecture Notes in Computer Science*, pages 385–396, Elounda, Greece, June/July 1993. Springer-Verlag.

[12] E.A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Symposium on Logic in Computer Science (LICS '86)*, pages 267–278, Cambridge, Massachusetts, June 1986. IEEE Computer Society Press.

[13] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[14] D.E. Long, A. Browne, E.M. Clarke, S. Jha, and W.R. Marrero. An improved algorithm for the evaluation of fixpoint expressions. In Dill [9], pages 338–350.

[15] O. Sokolsky and S. Smolka. Incremental model-checking in the modal mu-calculus. In Dill [9], pages 352–363.

[16] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. In *TAPSOFT*, volume 352 of *Lecture Notes in Computer Science*, pages 369–383, Barcelona, March 1989. Springer-Verlag.

[17] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 25(2):285–309, 1955.

[18] M. Vardi and P. Wolper. Yet another process logic. In E.M. Clarke and D. Kozen, editors, *Workshop on Logics of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 501–512, Pittsburgh, June 1983. SV.

[19] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Symposium on Logic in Computer Science (LICS '86)*, pages 332–344, Cambridge, Massachusetts, June 1986. IEEE Computer Society Press.

[20] G. Winskel. A note on model checking the modal ν-calculus. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Automata, Languages and Programming (ICALP '89)*, volume 372 of *Lecture Notes in Computer Science*, pages 761–772, Stresa, Italy, July 1989. Springer-Verlag.

8 Appendix : Proof of Theorem 4.3

We use the result in [5] to prove our theorem. First, we say that an and-or Kripke structure is *successful* if satisfies the temporal formula in Theorem 4.3. Our goal is to show that an assertion $s \models_M \phi$ is true iff the corresponding and-or Kripke structure is successful. In [5], necessary and sufficient condition for the truth of $s \vdash_\mathcal{T} \phi$ is given. Therefore, our goal now can be reduced to showing that the and-or Kripke Structure corresponding an assertion $\sigma \equiv s \vdash_\mathcal{T} \phi$ is successful iff σ satisfies this condition.

We first give a brief description of the technique in [5]. The model checking technique in [5] is also a goal oriented approach based on the use of *proof rules*. But the assertions they use are of a slightly different form. Their assertions are of form $H : s \vdash_\mathcal{T} \phi$ where H is referred to as the hypothesis set whose elements are of form $s' \vdash_\mathcal{T} \phi'$. Intuitively, the hypothesis set records information about the assertions seen on the path from the root. The proof rules in [5] with some minor modifications is shown in Fig 5.

A leaf σ is successful iff

- σ is of form *true*.
- σ is of form $H : s \vdash_\mathcal{T} [a]\phi$ and s does not have any a transitions.
- σ is of form $H : s \vdash_\mathcal{T} \nu X.\phi'$

A tableau for an assertion is *successful* iff all the leaves in the tableau are successful.

Theorem 8.1 *For an assertion $\sigma \equiv \{\} : s \vdash_\mathcal{T} \phi$, $s \models_\mathcal{T} \phi$ iff σ has a successful tableau.*

Before we present our proof, we introduce what we refer to as *proof trees*. We use these structures as means to connect the notions of and-or Kripke structures and tableaus.

Let $\mathcal{K} \equiv \langle \mathcal{Q}, \mathcal{A}, R, L, P, q_0 \rangle$ be the and-or Kripke structure corresponding to the assertion $s \vdash_\mathcal{T} \phi$. Let $\alpha \in \{\mu, \nu\}$. Also let $\bar{\alpha} = \mu$ iff $\alpha = \nu$ and vice versa.

Definition 8.2 *A proof tree for an assertion σ is a finite tree satisfying the following.*

1. *Every node in the tree is labelled with an element of Q.*

2. *The root of the tree is labelled with σ.*

3. *Let n be a node labeled by σ.*

   - *If $P(\sigma) = \wedge$ and $\{\sigma' \mid \langle \sigma, \sigma' \rangle \in R\} = \{\sigma_1, \ldots \sigma_m\}$ then n has exactly m successors $n_1, \ldots, n_m$, with n_i labeled by σ_i.*
   - *If $P(\sigma) = \vee$ then n has exactly one successor, n', and n' is labeled by some σ' such that $\langle \sigma, \sigma' \rangle \in R$.*

4. *Every leaf σ in the tree is of the form*

   - *true or*
   - *$s \vdash_T L$ where $s \not\vdash_T L$ or*
   - *$s \vdash_T \alpha X.\phi$ and there exists an internal no de σ' such that $\sigma' = \sigma$,σ' is an ancestor of σ and for every assertion σ'' of form $s' \vdash_T \bar{\alpha} Y.\phi'$ on the path from σ' to σ, $al(\bar{\alpha} Y.\phi') > al(\alpha X.\phi)$.*

A leaf n labelled with σ is successful iff $\sigma \equiv s \vdash_T \nu X.\phi'$ or $\sigma \equiv true$. A proof tree is successful iff all the leaves in the tree are successful.

Our proof now follows from the following two theorems.

Theorem 8.3 *An assertion $\sigma \equiv s \vdash_T \phi$ has a successful proof tree iff the and-or Kripke structure corresponding to the assertion is successful.*

Proof. '$\Rightarrow$' Assume σ has a successful proof tree. Now if we unwind the proof tree into an infinite tree, we get a run of the and-or Kripke structure corresponding to σ. From the definition of a successful proof tree, it is clear that this run satisfies the temporal formula in Theorem 4.3.

'$\Leftarrow$' Assume that the and-or Kripke structure corresponding to σ is successful. Then there exists a run of the Kripke structure that satisfies the LTL formula in Theorem 4.3. We now show how a successful proof tree can be constructed from this run. Since the run satisfies the LTL formula in Theorem 4.3, we know that on every path starting from the root in the run either (i) eventually there is a node labelled with *true* or (ii) an assertion $\sigma' \equiv s' \vdash_T \nu X.\phi'$ occurs infinitely often and no assertion of form $s'' \vdash_T \mu Y.\phi''$, such that $al(\mu Y.\phi'') \leq al(\nu X.\phi')$, occurs infinitely often on the path. In the first case, we terminate the path when we hit node labelled with *true*. In the second case we know there exists an node n_1 on the path labelled with σ' and there exists an ancestor n_2 of n_1 such that for every node labelled with $s'' \vdash_T \mu Y.\phi''$ on the path from n_1 to n_2, $al(\mu Y.\phi'') > al(\nu X.\phi')$. We terminate the path at such a node. It clear that all the leaves the leaves the resulting tree are successful. It remains to be shown that the tree resulting from this construction is finite. Assume that the constructed proof tree is not finite. By König's Lemma, there exists an

infinite path in the tree. This is contradiction since this path is successful and would be terminated by our construction.

Theorem 8.4 *An assertion σ has a successful proof tree iff it has a successful tableau.*

Proof. '$\Leftarrow$' Assume σ has a successful tableau. Replace every assertion $H : s \vdash_{\mathcal{T}} \phi$ by the assertion $s \vdash_{\mathcal{T}} \phi$. Our claim is that the resulting tree is a successful proof tree. It clear that this tree satisfies all requirements except 4. So we need to prove that every leaf n in the constructed tree satisfies the requirements for the leaves in a proof tree. If the leaf is labelled with *true* it obviously does. If the leaf is labelled with $s \vdash_{\mathcal{T}} \nu X.\phi$ then we know that this corresponds to a tableau leaf labelled with $H : s \vdash_{\mathcal{T}} \nu X.\phi$ where $\nu X.\phi \in H$. Now, from the tableau construction it follows that the leaf has an ancestor labelled with $H' : s \vdash_{\mathcal{T}} \nu X.\phi$ and for every node on the path from this ancestor to the leaf there exists no assertion of form $H'' : s' \vdash_{\mathcal{T}} \mu Y.\phi'$ such that $al(\mu Y.\phi') \leq \nu X.\phi$. It follows from the above observation that the leaf n in the constructed proof tree satisfies the necessary requirement.

'$\Rightarrow$' In this case we consider the *smallest* successful proof tree and map it to a successful tableau. Basically, an assertion $\sigma' \equiv s \vdash_{\mathcal{T}} \phi$ in the proof tree is replaced by the assertion $H : s \vdash_{\mathcal{T}} \phi$ in the tableau where $H = \{\sigma'' \mid \sigma'' \equiv \alpha X.\phi'$is an ancestor of σ, and for every assertion of form $\bar{\alpha} Y.\phi''$, $al(\bar{\alpha} Y.\phi'') > al(\alpha X.\phi')\}$. To show that the resulting structure is indeed a successful tableau, we need to show that (i) tableau rules can be applied at each internal node n, and the successors of n correspond to assertions obtained as a result of an application of a rule in Figure 5. (ii) the leaves are indeed leaves and are successful (Recall that a successful leaf in a tableau is labelled with true or of labelled with $H : s \vdash_{\mathcal{T}} \nu X.\phi$ where $s \vdash_{\mathcal{T}} \nu X.\phi \in H$).

The latter case is straightforward. The first case requires an analysis of the structure of the formula appearing in the the assertion labelling node n. Most of the cases are straightforward. The interesting cases are when the assertion has form $H : s \vdash_{\mathcal{T}} \nu X.\phi$ or $H : s \vdash_{\mathcal{T}} \mu X.\phi$. For the first case, assume no tableau rule can be applied at node n. This implies that $s \vdash_{\mathcal{T}} \nu X.\phi \in H$. From our construction it follows that node n' which is the pre-image of n in the original proof tree has an ancestor n'' labelled with $s \vdash_{\mathcal{T}} \nu X.\phi$ and on the path from n' to n'' there is no μ-formula with lower alternation level. Therefore, n' can be a leaf in the proof tree. This would mean there exists a smaller successful proof tree which contradicts our assumption the original proof tree was the smallest successful proof tree.

For the second case, again assume that no tableau rule can be applied at node n. This implies that $s \vdash_{\mathcal{T}} \mu X.\phi \in H$. From our construction it follows that node n' which is the pre-image of n in the original proof tree has an ancestor n'' labelled with $s \vdash_{\mathcal{T}} \mu X.\phi$. Let the subtrees at nodes n'

$$R1\ \frac{H : s \vdash_T L}{true}\ (s \models_T L)$$

$$R2\ \frac{H : s \vdash_T \phi_1 \wedge \phi_2}{H : s \vdash_T \phi_1 \quad H : s \vdash_T \phi_2} \qquad R3\ \frac{H : s \vdash_T \phi_1 \vee \phi_2}{H : s \vdash_T \phi_1}$$

$$R4\ \frac{H : s \vdash_T \phi_1 \vee \phi_2}{H : s \vdash_T \phi_2}$$

$$R5\ \frac{H : s \vdash_T [a]\phi}{H : s_1 \vdash_T \phi, .., H : s_m \vdash_T \phi}\ \{s_1, \ldots, s_m\} = \{s' | s \xrightarrow{a} s'\}$$

$$R6\ \frac{H : s \vdash_T \langle a \rangle \phi}{H : s_1 \vdash_T \phi}\ (s \xrightarrow{a} s_1)$$

$$R7\ \frac{H : s \vdash_T \alpha X.\phi}{H' : s \vdash_T \phi[\mu X.\phi/X]}\ (s \vdash_T \alpha X.\phi \notin H)$$

where $\alpha \in \{\mu, \nu\}$, $\bar{\alpha} = \mu(\nu)$ if $\alpha = \mu(\nu)$
and $H' = H - \{s' \vdash_T \bar{\alpha} X'.\phi' | al(\bar{\alpha} X'.\phi') > al(\alpha X.\phi)\}$

FIGURE 5. Proof rules for the μ-calculus, where $\mathcal{T} = \langle \mathcal{S}, Act, \rightarrow, I \rangle$.

and n'' be T'_n and T''_n respectively. Now, in the original proof tree, if we replace the subtree T''_n at node n' by the T'_n, the resulting tree is a smaller successful proof tree. This again is a contradiction as we assumed that the original proof tree was the smallest successful proof tree.