MODEL CHECKING IN THE
PROPOSITIONAL MU-CALCULUS

E. Allen Emerson and Chin-Laung Lei

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

TR-86-06  February 1986
Model Checking in
the Propositional Mu-Calculus

E. Allen EMERSON and Chin-Laung LEI

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

31 January 86 16:26

Abstract: In this paper we address the problem of automatically verifying that a given finite state concurrent program meets a correctness assertion specified in the propositional Mu-Calculus $\mu_\nu$ defined by Kozen in [Ko83]. The propositional Mu-Calculus provides a framework for handling fairness, past-tense temporal modalities, and extended temporal modalities such as those of [Wo81] in a uniform way. We give an efficient model checking algorithm for specifications given in a fragment, $\mu_\nu$, of the propositional Mu-Calculus, where the alternation depth of least and greatest fixpoint operators is restricted to be at most 2. We show that $\mu_\nu$ can succinctly encode (and is, in fact, strictly more expressive than) many of the commonly used logics of programs such as PDL ([FL79]), PDL-$\Delta$ ([St82]), CTL ([EC82], [CES83]), FCTL and GFCTL ([EL85]). In practice, we therefore have a small polynomial time model checker for a most useful portion of the propositional Mu-Calculus.

1. Introduction

Due to the rapid development of multiprocessor computer systems and communication networks, a tremendous amount of concurrently programmed software will be needed to take advantage of the fast, improved hardware. Because of the high degree of nondeterminism inherent in concurrency, software engineering techniques such as modular design and systematic debugging are not, by themselves, adequate for ensuring the development of error-free concurrent programs. Program verification thus plays an especially important role during the life cycle of concurrent program development. To improve the productivity of concurrent program development, automatic verification tools become more and more desirable. Recently Clarke, Emerson and Sistla ([CES83]) have suggested an automatic verification method for finite state concurrent systems: The global state graph of such a system can be viewed as a finite (Kripke) structure, and an efficient (small polynomial time) model checking algorithm is described for determining if a given structure is a model of a specification expressed in a propositional branching time temporal logic. Since the algorithm is very efficient, this method is readily mechanizable. Moreover, the approach is potentially of wide applicability since a large class of concurrent programming problems have finite state solutions, and their interesting properties can be specified in a propositional Temporal Logic (cf. [Va85], [LP85]).

Indeed, the role of Temporal Logic as a feasible approach to the specification and verification of concurrent systems is now widely accepted ([Pn84], [La83]). However, several recent developments indicate

---

1This work was supported in part by NSF Grants MCS8302878 and DCR8511354. A summary of these results will be presented at the IEEE Conference on Logic in Computer Science to be held 16-18 June 86 in Cambridge, Mass.
that enriching Temporal Logic can prove most advantageous. For example, it is known (cf. [La80], [EH83]) that the temporal logic CTL used in [CES83] does not permit reasoning under certain fairness assumptions. An extension of CTL, which we call Fair Computation Tree logic (FCTL), is thus introduced in [EL85]. Similarly, in [LPZ85] and [Pn84] it is argued that inclusion of past-tense temporal operators as well as the extended temporal operators of Wolper [Wo81] (e.g., "at all even moments, P holds") can facilitate the task of modular temporal reasoning about concurrent programs. Again, these sorts of temporal operators are not accommodated in the CTL logic of [CES83] (nor in the FCTL logic of [EL85].)

In this paper we investigate another way of extending model checking for CTL. We view CTL not as a sublanguage of the full branching time logic CTL* ([EH83]), but, rather, as a sublanguage of the propositional Mu-Calculus Lμ (cf. [Ko83], [Pr81], [EC80], [deB80]). The propositional Mu-Calculus provides a least fixpoint operator (μ) and a greatest fixpoint operator (ν) which make it possible to give fixpoint characterizations of the branching time modalities. Intuitively, the Mu-Calculus makes it possible to characterize the modalities in terms of recursively defined tree-like patterns. For example, the branching time temporal assertion EFp (along some computation path p will become true eventually) can be characterized as μZ.p ∨ EXZ, the least fixpoint of the functional p ∨ EXZ where Z is an atomic proposition variable (intuitively ranging over sets of states) and EX denotes the existential nexttime operator.

The propositional Mu-Calculus provides a uniform framework for extending the CTL model checking approach of [CES83] to handle fairness constraints as in [EL85], plus past tense temporal operators similar to those in [LPZ85] (cf. [PW84]), and extended temporal operators like those in [Wo81]. For example, the temporal assertion EFp ("along some computation path, p occurs infinitely often"), which is associated with fairness, can be characterized as νZ1.μZ2.EX[(P ∧ Z1) ∨ Z2]. The extended temporal operator "along all paths, P holds of all even moments" (cf. [Wo81]) is captured by νZ.P ∧ AXAXZ. We can accommodate the (branching time) past tense through use of the converse relations.

We show how model checking can be done when correctness assertions are formulated in the propositional Mu-Calculus Lμ. For the full Mu-Calculus Lμ our algorithm has exponential worst case time complexity. However, we consider model checking for restricted fragments Lμk of Lμ where the depth of alternating nestings of μ's and ν's is bounded by k-1. For example, Lμ1 consists of formulae obtained by composing modalities such as μZ.P ∨ EXZ (equivalent to EFP) and νZ.P ∧ AXZ (equivalent to AGP) which contain only μ's or only ν's. The fragment Lμ2 contains modalities such as νZ1.μZ2.EX[(P ∧ Z1) ∨ Z2] (equivalent to EFP) and μZ1.νZ2.AX[(P ∨ Z1) ∧ Z2] (equivalent to AGP, the dual of EFP), where one alternation of μ's inside ν's (or vice-a-versa) is allowed. We will show that the model checking problem for Lμk can be solved in polynomial time (with degree of the polynomial = k+1).
We go on to show that these restricted fragments of the propositional Mu-Calculus have considerable expressive power. We establish that the propositional Mu-Calculus, even restricted to alternation depth 2, is adequate for formalizing most of the reasoning about concurrent programs that can be done in a propositional Temporal Logic by proving that it subsumes many of the commonly used logics of programs such as PDL ([FL79]), PDL-$\Delta$ ([St82]), CTL ([EC82], [CES83]), and FCTL ([EL85]). In particular, we show that PDL and CTL can be succinctly translated into $L_{\mu_1}$ while PDL-$\Delta$ and FCTL can be succinctly translated into $L_{\mu_2}$.\footnote{CTL$^*$ can also be translated into $L_{\mu_2}$. Because the translation is via PDL-$\Delta$, however, it is not succinct.} We thus have a model checker of small polynomial time complexity for a specification language able to handle almost any correctness assertion arising in practice including all of the examples cited above.

The rest of the paper is organized as follows: In section 2 the syntax and semantics of the propositional Mu-Calculus, $L_{\mu}$, are defined. Section 3 describes how to do efficient model checking in the propositional Mu-Calculus $L_{\mu_k}$, and analyzes the complexity of our model checking algorithm. In section 4 we present our results on translating program logics into $L_{\mu_1}$ and $L_{\mu_2}$. Finally, some concluding remarks are given in section 5.

2. Preliminaries

2.1. Syntax.

The formulae of the propositional Mu-Calculus $L_{\mu}$ are:

(1) Atomic propositional constants $P$, $Q$, $R$, . . .
(2) Atomic propositional variables $X$, $Y$, $Z$, . . .
(3) $<A>p$, where $A$ is a member of a set of program letters $A$, $B$, $C$, . . . and $p$ is any formula.
(4) $\neg p$, the negation of formula $p$.
(5) $p \land q$, the conjunction of formulae $p$, $q$.
(6) $\mu X.p(X)$, where $p(X)$ is any formula syntactically monotone in the propositional variable $X$, i.e., all free occurrences of $X$ in $p(X)$ fall under an even number of negations.

The set of formulae generated by the above rules forms the language $L_{\mu}$. The other connectives are introduced as abbreviations in the usual way: $p \lor q$ abbreviates $\neg(\neg p \land \neg q)$, $[A]p$ abbreviates $\neg<A>\neg p$, $\nu X.p(X)$ abbreviates $\neg \mu X.\neg p(\neg X)$, etc. Intuitively, $\mu X.p(X)$ ($\nu X.p(X)$) stands for the least (greatest, resp.) fixpoint of $p(X)$, $<A>p$ ($[A]p$) means $p$ is true at some (every) next state of all reachable by executing atomic program $A$, $\land$ means "and", etc. We use $|p|$ to denote the length (i.e., number of symbols) of $p$.\footnote{Alternatively, we can define $|p|$ as the size of the syntax tree for $p$; see section 4.}

We say that a formula $q$ is a subformula of a formula $p$ provided that $q$, when viewed as a sequence of symbols, is a substring of $p$. A subformula $q$ of $p$ is said to be proper provided that $q$ is not $p$ itself. A top-level (or immediate) subformula is a maximal proper subformula. We use SF($p$) to denote the set of
subformulae of \( p \).

The fixpoint operators \( \mu \) and \( \nu \) are somewhat analogous to the quantifiers \( \exists \) and \( \forall \). Each occurrence of a propositional variable \( X \) in a subformula \( \mu X.p(X) \) (or \( \nu X.p(X) \)) of a formula is said to be bound. All other occurrence are free. By renaming variables if necessary we can assume that the expression \( \mu X.p(X) \) (or \( \nu X.p(X) \)) occurs at most once for each \( X \).

A sentence (or closed formula) is a formula that contains no free propositional variables, i.e., every variable is bound by either \( \mu \) or \( \nu \). A formula is said to be in positive normal form (PNF) provided that no variable is quantified twice and all the negations are applied to atomic propositions only. Note that every formula can be put in PNF by driving the negations in as deep as possible using DeMorgan's Laws and the dualities \( \neg\mu Y.p(Y) = \nu Y.\neg p(\neg Y) \), \( \neg\nu Y.p(Y) = \mu Y.\neg p(\neg Y) \) (This can at most double the length of the formula). Subsentences and proper subsentences are defined in the same way as subformulae and proper subformulae.

Let \( \sigma \) denote either \( \mu \) or \( \nu \). If \( Y \) is a bound variable of formula \( p \), there is a unique \( \mu \) or \( \nu \) subformula \( \sigma Y.p(Y) \) of \( p \) in which \( Y \) is quantified. Denote this subformula by \( \sigma Y \). \( Y \) is called a \( \mu \)-variable if \( \sigma Y = \mu Y \); otherwise, \( Y \) is called a \( \nu \)-variable. A \( \sigma \)-subformula (\( \sigma \)-subsentence, resp.) is a subformula (subsentence) whose main connective is either \( \mu \) or \( \nu \). We say that \( q \) is a top-level \( \sigma \)-subformula of \( p \) provided \( q \) is a proper \( \sigma \)-subformula of \( p \) but not a proper \( \sigma \)-subformula of any other \( \sigma \)-subformula of \( p \). Finally, a basic modality is a \( \sigma \)-sentence that has no proper \( \sigma \)-subsentences.

2.2. Semantics.

We are given a set \( AP \) whose elements are called atomic (or propositional) constants and a set \( P_0 \) whose elements are (atomic) program letters. Sentences of the propositional Mu-Calculus \( L\mu \) are interpreted with respect to a structure \( M = (S, \mathcal{R}, L) \), where

- \( S \) is a nonempty set of states,
- \( \mathcal{R}: P_0 \to 2^{S \times S} \) assigns a binary relation on \( S \) to each atomic program,
- \( L: S \to 2^{AP} \) is a labelling which assigns to each state a set of atomic propositions true in the state.

The size of a structure \( M = (S, \mathcal{R}, L) \), written \( |M| \), is defined to be \( |S| + \sum_{A \in P_0} |\mathcal{R}(A)| \), i.e., the sum of the number of states in \( S \) and the number of transitions in \( \mathcal{R} \).

The power set of \( S \), \( 2^S \), may be viewed as the complete lattice \( (2^S, S, \emptyset, \subseteq, \cup, \cap) \). Intuitively, we identify a predicate with the set of states which make it true. Thus, \( false \) which corresponds to the empty set is the bottom element, \( true \) which corresponds to \( S \) is the top element, and implication \((\forall s \in S[P(s) \Rightarrow Q(s)])\) which corresponds to simple set-theoretic containment \((P \subseteq Q)\) provides the partial ordering on the lattice.
Let \( \tau : 2^S \to 2^S \) be given; then we say that \( \tau \) is monotonic provided that \( P \subseteq Q \) implies \( \tau(P) \subseteq \tau(Q) \), \( \tau \) is \( \cup \)-continuous provided that \( P_0 \subseteq P_1 \subseteq P_2 \subseteq \ldots \) implies \( \tau(\bigcup_i P_i) = \bigcup_i \tau(P_i) \), and that \( \tau \) is \( \cap \)-continuous provided that \( P_0 \supseteq P_1 \supseteq P_2 \supseteq \ldots \) implies \( \tau(\bigcap_i P_i) = \bigcap_i \tau(P_i) \). Either type of continuity can easily be shown to imply monotonicity. Furthermore, if \( S \) is finite, then any monotonic functional is also both \( \cup \)-continuous and \( \cap \)-continuous.

A monotonic functional \( \tau \) always has both a least fixpoint, \( \mu_X.\tau(X) \), and a greatest fixpoint \( \nu_X.\tau(X) \):

**Theorem 2.1** [Tarski-Knaster]. Let \( \tau : 2^S \to 2^S \) be a given monotonic functional. Then

(a) \( \mu_X.\tau(X) = \bigcap \{ X : \tau(X) = X \} = \bigcap \{ X : \tau(X) \subseteq X \} \),
(b) \( \nu_X.\tau(X) = \bigcup \{ X : \tau(X) = X \} = \bigcup \{ X : \tau(X) \supseteq X \} \),
(c) If \( \tau \) is \( \cup \)-continuous then \( \mu_X.\tau(X) = \bigcup_i \tau^i(\text{false}) \), and
(d) If \( \tau \) is \( \cap \)-continuous then \( \nu_X.\tau(X) = \bigcap_i \tau^i(\text{true}) \).

A formula \( p \) with free variables \( X_1, \ldots, X_n \) is thus interpreted as a mapping \( p^M \) from \( (2^S)^n \) to \( 2^S \), i.e., it is interpreted as a predicate transformer. We write \( p(X_1, \ldots, X_n) \) to denote that all free variables of \( p \) are among \( X_1, \ldots, X_n \). A valuation \( \mathcal{V} = (V_1, \ldots, V_n) \) is an assignment of subsets of \( S \) (\( V_1, \ldots, V_n \)) to free variables (\( X_1, \ldots, X_n \)). We use \( p^M(\mathcal{V}) \) to denote the value of \( p \) on the (actual) arguments \( V_1, \ldots, V_n \) (cf. [EC80], [Pr81], [Ko83]). The operator \( p^M \) is defined inductively as follows:

1. \( p^M(\mathcal{V}) = \{ s : s \in S \text{ and } P \in L(s) \} \) for any atomic propositional constant \( P \in AP \)
2. \( X_i^M(\mathcal{V}) = V_i \)
3. \( (p \land q)^M(\mathcal{V}) = p^M(\mathcal{V}) \land q^M(\mathcal{V}) \)
4. \( (\neg p)^M(\mathcal{V}) = S \setminus (p^M(\mathcal{V})) \)
5. \( (\langle A \rangle p)^M(\mathcal{V}) = \{ s : \exists t \in p^M(\mathcal{V}), (s,t) \in R(A) \} \)
6. \( \mu X.p(X)^M(\mathcal{V}) = \bigcap \{ S' \subseteq S : p(S')^M(\mathcal{V}) \subseteq S' \} \)

Note that our syntactic restrictions on monotonicity ensure that least and greatest fixpoints are well-defined.

Usually we write \( M, s \vdash p \) (\( M, s \vdash p(\mathcal{V}) \), resp.) instead of \( s \in p^M \) (\( s \in p^M(\mathcal{V}) \)) to mean that sentence (formula) \( p \) is true in structure \( M \) at state \( s \) (under valuation \( \mathcal{V} \)). When \( M \) is understood, we write simply \( s \vdash p \).

**2.3. Alternation Depth.**

We introduce the notion of alternation depth of formulas in \( L_\mu \) which plays an important role in the complexity analysis of our model checking algorithm. Although the intuition for alternation depth is quite obvious, it turns out that the formal definition is rather subtle. In the definition below we assume that \( f \) is in PNF. (For any formula \( f \) not in PNF, \( A(f) \) is defined as \( A(f') \), where \( f' \equiv f \) is in PNF.)
Definition 2.2: The alternation depth of a formula \( f \), \( A(f) \), is defined recursively as follows:

AD1. If \( f \) contains proper \( \sigma \)-subsentences \( p_1', \ldots, p_n' \), then \( A(f) = \max(A(p_1), \ldots, A(p_n), A(f')) \), where \( f' \) is obtained from \( f \) by substituting fresh atomic propositional constants \( P_1, \ldots, P_n \) for \( p_1, \ldots, p_n \) respectively, in \( f \).

AD2. \( A(P) = 0 \), for any atomic proposition constant \( P \)

AD3. \( A(Y) = 0 \), for any atomic proposition variable \( Y \)

AD4. \( A(p \land q) = \max(A(p), A(q)) \)

AD5. \( A(p \lor q) = \max(A(p), A(q)) \)

AD6. \( A(\neg p) = A(p) \)

AD7. \( A(<A>p) = A(p) \), for any program letter \( A \)

AD8. \( A([A]p) = A(p) \), for any program letter \( A \)

AD9. \( A(\mu X.p) = \max(1, A(p), 1+A(\nu Y_1.q_1), \ldots, 1+A(\nu Y_n.q_n)) \), where \( \nu Y_1.q_1, \ldots, \nu Y_n.q_n \) are the top-level \( \nu \)-subformulae of \( p \).

AD10. \( A(\nu X.p) = \max(1, A(p), 1+A(\mu Y_1.q_1), \ldots, 1+A(\mu Y_n.q_n)) \), where \( \mu Y_1.q_1, \ldots, \mu Y_n.q_n \) are the top-level \( \mu \)-subformulae of \( p \).

Examples:

1. \( A(\mu Y.P \lor <A>Y) = 1 \), by AD9
2. \( A(\mu Y.(\mu Z.P \lor [A]Z) \lor <A>Y)) = 1 \), by AD1 then AD9
3. \( A(\mu Y.(\nu Z.P \land [A]Z) \lor <A>Y)) = 1 \), by AD1 then AD9 and AD10
4. \( A(\nu Z_1.\neg\nu Z_2.[A]((\neg P \lor \neg Z_1) \land Z_2)) = A(\nu Z_1.\mu Z_2.\neg A>((P \land Z_1) \lor Z_2)) \), put it in PNF
   \[ = 1 + A(\mu Z_2.\neg A>((P \land Z_1) \lor Z_2)) \]
   by AD10
   \[ = 2 \], by AD9
5. \( A(\nu Z_1.\mu Z_2.\neg A>((\nu Y_1.\mu Y_2.\nu Y_1)((P \land Y_1) \lor Y_2)) \land Z_1) \lor Z_2) \)
   \[ = \max\{A(\nu Z_1.\mu Z_2.((\nu Y_1.\mu Y_2.\nu Y_1)((P \land Y_1) \lor Y_2)) \land Z_1) \lor Z_2)\}
   by AD1
   \[ = 2 \], by AD9 and AD10
6. \( A(\mu X.\nu Y.(P \lor ((\mu Z.(X \lor <A>Y)) \land <B>Y))) = 3 \), by AD9 then AD10 and then AD9 again

Remark: One possibly helpful way to understand our definition of alternation depth is to use the notion of active variables introduced in [Ko83]. Let \( p_0 \) be a basic modality of \( L_\mu \), i.e., a sentence with no proper \( \sigma \)-subsentences. Draw the syntax tree for \( p_0 \) in the usual way; then, for each node corresponding to an occurrence of a \( \sigma \)-variable \( X \), draw a directed arc from the node for \( X \) to the node for \( \sigma X \). (See Figure 2-1) We say that variable \( X \) is active in subformula \( q \) in sentence \( p_0 \) provided that there is a directed path from (the root node for) \( q \) to (the node for) \( \sigma X \). Then, for example, the formula \( \mu X.\nu Y.(P \lor ((\mu Z.(X \lor <A>Y)) \land <B>Y)) \) from above is of alternation depth 3 because it contains a chain of 2 consecutive "active" alternations of \( \sigma \)-variables: In the chain \( \mu X \ldots \nu Y \ldots \mu Z \), the alternation \( \mu X \ldots \nu Y \) is active because \( X \) is active in \( \nu Y \) (indeed, it occurs free in \( \nu Y \)) while the alternation \( \nu Y \ldots \mu Z \) is active because \( Y \) is active in \( \mu Z \) (even though \( Y \) does not occur free in \( \mu Z \)).
Definition 2.3: We denote by $L\mu_k$ the fragment of $L\mu$ restricted to alternation depth $k$.


In this section we develop a model checking algorithm for $L\mu$. We also show that the complexity of our algorithm for $L\mu_k$ is a polynomial of degree $k+1$ in both of the input structure and the input specification.

The algorithm of Figure 3-1 for model checking in the Mu-Calculus is a straightforward implementation of the semantics and the Tarski-Knaster Theorem (Theorem 2.1) for evaluating fixpoints.

We can improve the performance of the algorithm by observing that subexpressions need only be evaluated once. Moreover, we can exploit the simultaneous monotonicity induced by $\sigma$-variables of the same type by using the following variant of the Tarski-Knaster Theorem:

**Theorem 3.1:** If $p(Y): 2^S \rightarrow 2^S$ is $\cup$-continuous, then $\mu Y.p(Y) = \bigcup_{i>0} p^i(Y_0)$ for any initial set $Y_0 \subseteq \mu Y.p(Y)$, and if $p(Y)$ is $\cap$-continuous then $\nu Y.p(Y) = \bigcap_{i>0} p^i(Y_0)$ for any initial set $Y_0 \supseteq \nu Y.p(Y)$. \hfill $\square$

For example, consider $p = \mu Y_1.p_1(Y_1)$ where $p_1(Y_1)$ contains the subformula $\mu Y_2.p_2(Y_1, Y_2)$. Based on Theorem 3.1, to evaluate $\mu Y_1.p_1(Y_1)$, we successively compute $p^i_Y(\text{false})$, $p^{2i}_Y(\text{false})$, (i.e., $p_1(p^\text{false}_Y)$), $p^3_Y(false)$, ... until stabilization. In order to compute $p^i_Y(\text{false})$ we must first evaluate the subformula $\mu Y_2.p_2(\text{false}, Y_2)$. We can then again use Theorem 3.1 to calculate $p^i_{Y_2}(false, false)$, $p^{2i}_{Y_2}(false, false)$, (i.e., $p_2(p^\text{false}_{Y_2}, p^\text{false}_{Y_2})$), $p^{3i}_{Y_2}(false, false)$, ... until stabilization. The calculation of $\mu Y_2.p_2(\text{false}, Y_2)$ can involve up to $|M|$ steps.

Similarly, to compute $p_Y^i(\text{false})$, we compute subformula $\mu Y_2.p_2(p^i_{Y_2}Y_{false}, Y_2)$ by means of a subcomputation $p_2(p^{i-1}_{Y_2}(false), false)$, $p^{2i}_{Y_2}(false, false)$, $p^{3i}_{Y_2}(false, false)$, ... of up to $|M|$ steps. Thus $O(|M|^2)$ evaluations of $p^i_Y(\text{false})$ are performed. In general, if we have a $\sigma$-formula nested $k$ deep, the body of the innermost $\sigma$-subformula will be evaluated $O(|M|^k)$ times. Since the depth $k$ of nesting $\sigma$-subformulae can be $\geq \Omega(|p_0|)$, we get an algorithm of superexponential complexity, $O(|M|.|p_0|)^{c|p_0|}$, for some constant $c > 0$.

However, whenever the nested $\sigma$-subformulae are all of the same type (all $\mu$'s or all $\nu$'s) we can do better. In the above example, we must compute $p_1(\text{false})$, $p^{2i}_Y(\text{false})$, $p^{3i}_Y(\text{false})$, ... and for each $p^i_Y(\text{false})$ we need to compute $\mu Y_2.p_2(p^{i-1}_Y(\text{false}), false)$. But, it is not necessary to restart the $\mu Y_2$ subcomputation with the $Y_2$ variable $= false$ each time. Since $p^{i-1}_Y(\text{false}) \subseteq p_1(\text{false})$ and $p_2(Y_1, Y_2)$ is monotonic in $Y_1$, we have that $\mu Y_2.p_2(p^{i-1}_Y(\text{false}), Y_2) \subseteq \mu Y_2.p_2(p^i_Y(\text{false}), Y_2)$. Thus we can start the computation for $\mu Y_2.p_2(p^i_Y(\text{false}), Y_2)$ with the $Y_2$ variable equal to the previously computed $\mu Y_2.p_2(p^{i-1}_Y(\text{false}), Y_2)$. Only about $O(|M|)$ calculations of $p^i_Y(\text{false})$ are needed.

In general, instead of reinitializing the $\mu$-variable $Y$ each time the subformula $\mu Y.p(Y)$ is evaluated, it
suffices to reinitialize Y just when the \( \mu \)-variable \( Z \) of the closest surrounding \( \nu \)-formula \( \nu Z \cdot q \) changes\(^4\). We can use the following rule, inserted at the beginning of the evaluation of \( \mu Y \cdot p(Y) \).

if the surrounding \( \sigma \)-formula of \( \mu Y \) is \( \nu Z \)
then \( Y = \emptyset \);
for each open subformula \( \mu X \) of \( \mu Y \) such that there is no
subformula \( \nu W \) of \( \mu Y \) containing \( \mu X \), set \( X = \emptyset \).\(^5\)

A symmetric rule is used for reinitializing \( \nu \)-variables. The improved algorithm is shown in Figure 3-2.

**Theorem 3.2:** The algorithm in Figure 3-2 is of time complexity \( O([|p_0|\cdot|M|]^{A(p_0)+1}) \) where \( A(p_0) \) is the alternation depth of \( p_0 \) (defined above).

**proof:** It suffices to consider the case where \( p_0 \) does not contain any proper \( \sigma \)-subsentence (i.e., is a basic modality), since each subsentence is evaluated only once.

If \( p_0 \) is of alternation depth 0, so it contains no \( \sigma \)-subformulae, then it is easy to see that the algorithm takes time \( O([|M|\cdot|p_0|]) \). Let us next consider \( p_0 \) of alternation depth 1 so that it contains only \( \sigma \)-subformula of the same type, say, \( \mu \)-subformula (A symmetric argument applies when it consists only of \( \nu \)-subformulae). Define the \( \mu \)-depth in \( p_0 \) of subformula \( \mu Y_i \), \( D(\mu Y_i) \), to be the number of \( \mu \)-subformulae surrounding \( \mu Y_i \). Suppose we have \( \mu Y_k \) is a proper subformula of \( \mu Y_{k-1} \ldots \) is a proper subformula of \( \mu Y_1 \) is a proper subformula of no \( \mu \)-subformula of \( p_0 \). Write \( \#_{\text{evals}}(p_i) \) for the total number of times \( \text{eval}(p_i) \) is called when evaluating \( p_0 \). Now,

1. \( \#_{\text{evals}}(p_1) \leq |S| + 1 \) since each time, except the last, \( p_1 \) is evaluated the size of \( Y_1 \) increases, and

2. \( \#_{\text{evals}}(p_{i+1}) \leq |S| + \#_{\text{evals}}(p_i) \) since there can be up to \( |S| \) calls to eval\((p_{i+1})\) which increase \( Y_{i+1} \) in addition to 1 call to eval\((p_{i+1})\) for each call to eval\((p_i)\).

Thus, for any subformula \( q \) of \( p_0 \), \( \#_{\text{evals}}(q) \leq |p_0|\cdot|M| \). Now, the total cost to evaluate \( p_0 \) can be calculated as:

\[
\text{total cost} \leq \sum_{q \in SF(p_0)} (\text{local cost to evaluate } q \text{ once}) \cdot (\text{total number of times } q \text{ is evaluated})
\]

Here, the local cost to evaluate \( q \) once is the time required for a single evaluation of \( q \) exclusive of the time spent (in recursive calls) to evaluate the immediate subformulae. Plainly, when \( q \) is not a \( \sigma \)-subformula, the local cost of \( q \) is at most \( c_1 \cdot |M| \), for some constant \( c_1 \). For example if \( q = p_1 \land p_2 \), it is essentially the time required to intersect the sets of states associated with \( p_1 \) and \( p_2 \); the time required to evaluate \( p_1 \) and \( p_2 \) is

\( \phantom{\text{total cost}} \)

\( ^4 \)Actually, we only need to reset \( Y \) when a surrounding \( \nu \)-variable \( Z \) which is active in \( p \) changes. (See the remark near the end of subsection 2.3.)

\( ^5 \)For future reference, call such an open formula \( \mu X \) of \( \mu Y \) an *unguarded* open subformula.
not charged to the local cost of q, but rather to \( p_1 \) and \( p_2 \) respectively.

For \( q = \sigma Y_i \cdot p_i \), the local cost can include the charges for resetting \( S_i \) and any \( S_j \) corresponding to open unguarded \( \sigma \)-subformulae, and for storing (the old value of) \( S_i \) in \( S_i' \) and comparing \( S_i' \) with (the new value of) \( S_i \) (obtained after evaluating the body \( p_i \)). This can yield a local cost of \( O(|q| \cdot |M| + |M|^2) \). However, the total cost estimate will not decrease if we calculate it on the basis an appropriately modified local cost.

Observe that it is, for example, allowable to modify the local costs of formulae \( q \) and \( q' \) by subtracting an amount \( a \cdot b \) from \( q \) and adding amount \( a \) to \( q' \), provided that \( q' \) is evaluated at least \( b \) times for each time \( q \) is evaluated. Therefore, for, say, a \( \mu Y_i \) formula such that the surrounding \( \sigma \)-formula is a \( \nu \)-formula (or has no surrounding \( \sigma \)-formula), we charge to its (modified) local cost the cost to reinitialize \( S_i \) to \( \emptyset \), which is at most \( c_2 |M| \) for some constant \( c_2 \). However, the cost to reinitialize \( S_j \) for \( \mu Y_j \), an open unguarded subformula of \( \mu Y_i \), is charged to the (modified) local cost of \( \mu Y_j \). Since \( \mu Y_j \) will be evaluated at least once for each \( \mu Y_i \) evaluation, this is allowable. Also, for an evaluation of \( \mu Y_i \cdot p_i \) we store \( S_i \) in and compare \( S_i \) with \( S_i' \) some number \( b \) times, where \( b \) is the number of times the body \( p_i \) is evaluated during this evaluation of \( \mu Y_i \cdot p_i \). The cost of a single such store and subsequent comparison is some amount \( a \leq c_3 |M| \), for some \( c_3 \). The total contribution of all storing and comparing of \( S_i \) to the local cost of an evaluation of \( \mu Y_i \cdot p_i \) is thus \( a \cdot b \). By our observation above, we can therefore modify the local costs of \( \mu Y_i \cdot p_i \) and \( p_i \) by subtracting \( a \cdot b \) from \( \mu Y_i \cdot p_i \) and adding \( a \) to \( p_i \). We can similarly modify the local costs for \( \nu \)-subformulae.

Now the modified local cost of each subformula, whatever its main connective, is \( \leq c \cdot |M| \) for some constant \( c \). Thus the total cost of evaluating \( p_0 \leq \sum_{q \in \text{SF}(p_0)} (\text{modified local cost to evaluate } q \text{ once}) \cdot (\text{total number of times } q \text{ is evaluated}) \leq |p_0| \cdot (\text{maximum of the modified local cost of any } q \in \text{SF}(p_0)) \cdot (\text{maximum of the total number of times any } q \in \text{SF}(p_0) \text{ is evaluated}) \leq |p_0| \cdot (c \cdot |M|) \cdot (|p_0| \cdot |M|) \leq c \cdot (|p_0| \cdot |M|)^2 \).

We can now argue by induction on the alternation depth, that a formula of alternation depth \( k \) can be evaluated in time \( c \cdot (|p_0| \cdot |M|)^{k+1} \), where \( c \) the the constant mentioned in the previous paragraph. The basis step for \( k \leq 1 \) is as above. For the induction step, let \( p_0 \) be a sentence of alternation depth \( k + 1 \). We assume that all \( \sigma \)-subformula in \( p_0 \) of alternation depth \( k + 1 \) are \( \mu \)-subformulae (The case where they are \( \nu \)-subformulae is symmetric). Then \( p_0 \) can be expressed as a composition of the form \( p' [\nu Y_1, ..., \nu Y_l] \) where the \( \nu Y_i \)'s are the top-level \( \nu \)-subformulae, each of alternation depth \( \leq k \), and \( p' \) is the formula obtained from \( p_0 \) by replacing each \( \nu Y_i \) with a "fresh" variable \( Z_i \); \( p' \) is of alternation depth 1.

In a manner similar to that used for the basis case, we can calculate the total cost for evaluating \( p_0 \) as follows: total cost \( \leq \sum_{q \in \text{SF}(p')} (\text{total number of times } q \text{ is evaluated}) \cdot (\text{modified local cost of evaluating } q \).
once), where the modified local cost of evaluating $Z_i$ is the cost of evaluating $\nu Y_i$ --- which is $c(|\nu Y_i|^l |M|)^{k+1}$ by induction hypothesis plus the additional cost due to the "modification" discussed above which is $c|\nu Y_i|$. Since $p'$ is of alternating depth 1, we have, as above, that the maximal number of times say subformula $q$ of $p'$ is evaluated is $\leq |p'| |M|$. Hence the total cost

$\leq \left(\sum_{q \in \text{SF}(p')} \left(\text{modified local cost of evaluating } q \text{ once)}\right) + \sum_{i=1}^{1} c \cdot (|\nu Y_i|^l |M|)^{k+1}\right)$

$\leq \left(6 |p_0|^l |M|\cdot |p'| \cdot (c |M|) + c \cdot (|p_0|^l |p'| |M|)^{k+1}\right)$

$\leq \left(|p_0|^l |M| \cdot c \cdot |p_0|^l |M|)^{k+1}\right)$

$\leq c (|p_0|^l |M|)^{k+2}$

\square

4. Expressiveness

Since $A(p_0) \geq \Omega |p_0|$, our improved algorithm still has exponential worst case complexity $(|M| |p_0|)^c |p_0|$, for some constant $c > 0$. However, almost all useful modalities can be expressed in the Mu-Calculus with alternation depth 1 or 2. Indeed, the expressive power of the Mu-Calculus of limited alternation depth is underscored by the results below which establish that PDL-$\Delta$ (Propositional Dynamic Logic with Looping, cf. [St82]) and FCTL (Fair Computation Tree Logic, cf. [EL85]) can be succinctly encoded into $L_{\mu_2}$ while (ordinary) PDL (cf. [FL79]) and CTL (cf. [CE81]) can be succinctly encoded into $L_{\mu_1}$. In practice, we therefore have a model checker for the useful portion of Mu-Calculus of small polynomial time complexity.

4.1. Mu-Calculus and Propositional Dynamic Logic

We denote by $p^t$ the Mu-Calculus translation of PDL-$\Delta$ formula $p$, where the translation $t$ is defined by the following rules (cf. [Pr79], [Ko83])

T1. $P^t = P$ for atomic propositional constants $P$
T2. $Y^t = Y$ for atomic propositional variables $Y$
T3. $(\neg p^t = \neg p^t$
T4. $(<A>p)^t = (<A>p)^t$
T5. $(<a \lor b>p)^t = (<a>p)^t \lor (<b>p)^t$
T6. $(<p?q)^t = p^t \land q^t$
T7. $(<a;b>p)^t = (<a><b>p)^t$
T8. $(<a^*>p)^t = \nu Y.(p^t \lor (<a>Y)^t)$, for "fresh" variable $Y$
T9. $(\Delta a)^t = \nu Y.(<a>Y)^t$, for "fresh" variable $Y$

\[\text{A brief description of PDL-$\Delta$ and FCTL is given in Appendix.}\]
Since p is duplicated in translating \(<a \cup b,p, \mid p^t\mid \) the length of p^t considered as a string of symbols in the usual way, can be exponential in \(|p|\). However, if we have an intelligent representation scheme which consolidates common subformulae (so that, e.g., the parse "graph" of \(<a> p \lor <b>p\) would have the form of Figure 4-1), then \(||p^t||\), the size of the representation of p^t (e.g., the size of its parse graph) is linear in |p|. In the sequel, we will assume that the "size" of a formula is measured by the size of its parse graph instead of number of symbols. We thus have

**Proposition 4.1:** For every PDL-\(\Delta\) formula \(f(X_1, \ldots , X_n)\) having free variables \(X_1, \ldots , X_n\), \(t(X_1, \ldots , X_n)\) is an L\(\mu\) formula also having free variables \(X_1, \ldots , X_n\), which is equivalent to \(f\); i.e., \(\forall \) structure \(M = (S, R, L)\), \(\forall \) state s of M, and for every valuation \(\nu = (V_1, \ldots , V_n)\), \(M, s \models f(\nu) \iff M, s \models t(\nu)\). Moreover, \(||t||\) = \(O(|f|)\).

**Proof:** The proof is given in the Appendix. \(\square\)

**Theorem 4.2:** For any PDL sentence \(p\), \(p^t\) is an equivalent Mu-Calculus sentence of alternation depth \(\leq 1\). Thus, PDL is succinctly translatable into L\(\mu_1\) in linear time.

**Proof:** Correctness and succinctness of the translation follow from Proposition 4.1, and the observation that translation rules T1-9 can plainly be implemented in linear time.

To see that \(p^t\) is of alternation depth \(\leq 1\), note that the translation rules T1-9 above define a mapping from PDL sentences to Mu-Calculus sentences. Since we only introduce least fixpoint operators (\(\mu\)) during the translation and negations are only applied to subsentences, it is clear that \(p^t\), the Mu-Calculus counterpart of the the PDL sentence \(p\), is of alternation depth at most 1 according to definition 2.2. \(\square\)

Since the L\(\mu_1\) formula \(\mu x. [A] <\Delta> x\) is not expressible in PDL (cf. [Ko83]), we have the following:

**Corollary 4.3:** L\(\mu_1\) is strictly more expressive than PDL. \(\square\)

Let \(p\) be a PDL-\(\Delta\) formula and \(q\) be a subsentence of \(p\). We observe that the translation \(t\) has the following property: \([p(q/Q)]^t(Q^t/q^t) = [p(q/Q)]^t(Q/q^t) = p^t\), where \(Q\) is a fresh atomic variable not occurring in \(p\) and \(p(x/y)\) denotes the substitution of \(y\) for \(x\) in \(p\).

**Theorem 4.4:** For any PDL-\(\Delta\) sentence \(p\), \(p^t\) is a Mu-Calculus sentence of alternation depth \(\leq 2\). Thus, PDL-\(\Delta\) is succinctly translatable into L\(\mu_2\) in linear time.

**Proof:** The correctness and succinctness of the translation follow immediately from Proposition 4.1 and it is clear that the translation can be implemented in linear time complexity. To see that the alternation depth is \(\leq 2\), we argue as follows: Let \(p\) be an arbitrary PDL-\(\Delta\) sentence, we will prove by induction on \(#\Delta(p)\), the number of \(\Delta\) operators in \(p\), that \(\mathcal{A}(p^t)\leq 2\).

**Basis:** If \(#\Delta(p) = 0\) then \(\mathcal{A}(p^t)\leq 1\) by Theorem 4.2. If \(#\Delta(p) = 1\), let \(\Delta a\) be the unique \(\Delta\)-sentence of \(p\), then \((\Delta a)^t = \nu Y. (<a>Y)^t\). Let \(q_1, \ldots , q_n\) be all the top-level subsentences of \(<a>Y\). Let \(<a'>Y =\)
<a>Y[q_1/Q_1, ..., q_n/Q_n]\). By the above observation, we have \((<a>Y)^t = (<a'>Y)^t(Q_1/q_1^t, ..., Q_n/q_n^t). By definition 2.2, \(A((<a>Y)^t) = \max(A((<a'>Y)^t), A(q_1^t), ..., A(q_n^t)). Since, \(q_i\) is a sentence of PDL, \(A(q_i^t) \leq 1\) by Theorem 4.2. Note that the only negations in \(<a>Y\) are applied to subsentences (because \(Delta a\) is a sentence). Since any negation of a subsentence is a subsentence and we replace all top-level proper subsentences of a by atomic propositional constants to get \(a', a'\) is negation free. Therefore, \((<a'>Y)^t\) contains only \(mu\) operators (if it contains any \(sigma\)-operators) and is in PNF. Applying definition 2.2, \(A((<a'>Y)^t) \leq 1\). Hence, \(A((<a>Y)^t) = \max(A((<a'>Y)^t), A(q_1^t), ..., A(q_n^t)) \leq \max(1, 1, ..., 1) = 1\). By definition 2.2, \(A((Delta a)^t) \leq 1 + A((<a>Y)^t) \leq 2\). Now, we can evaluate \(A(p^t). A(p^t) \leq \max(2, A((p')^t)), where p' is obtained from p by replacing \(Delta a\) with a fresh atomic propositional constant. Since p' is an ordinary PDL formula, \(A((p')^t) \leq 1\) by Theorem 4.2. Therefore \(A(p^t) \leq 2\).

Induction step: Assume \(#Delta(p) = k + 1 \geq 2\. Let \(Delta b\) be a \(Delta\)-sentence of p with \(#Delta(Delta b) \leq k\). By induction hypothesis, \(A((Delta b)^t) \leq 2\). By rule AD1 of Definition 2.2, \(A(p^t) \leq \max(2, A((p')^t)), where p' is the formula p with \(Delta b\) replaced by a "fresh" atomic proposition. Again by induction hypothesis, \(A(p^t) \leq 2\). Therefore, \(A(p^t) \leq 2\).

In [Ni84] it has been shown that \(nu X.(<A>X \land <B>X)\) is not expressible in PDL-\(Delta\), we thus have the following corollary:

Corollary 4.5: \(L_{mu_2}\) is more expressive than PDL-\(Delta\). □

Theorem 4.6: There exists an algorithm for model checking in PDL-\(Delta\) which runs in time \(O([|M|,|p_0|,3])\) for input structure M and sentence p_0.

Proof: If we modify the model checking algorithm given in the previous section so that it does not evaluate common subformulae twice, it is then straightforward to see that the modified model checking algorithm runs in time \(O([|p_0|,|M|,Delta(p_0) + 1])\) instead of \(O([|p_0|,|M|,Delta(p_0) + 1])\). The theorem thus follows from Theorem 4.4 and the modified algorithm. □

4.2. Mu-Calculus and FCTL.

To compare the Mu-Calculus with endogenous Temporal Logics such as CTL and FCTL we only have to consider \(L_{mu}\) over one program letter. In the sequel, we therefore assume that \(pi_0 = \{A\}\).

Theorem 4.7. There is a succinct, efficient (i.e., linear size and computable in linear time) translation of CTL into \(L_{mu_1}\) over 1 program letter. Moreover, \(L_{mu_1}\) over 1 program letter is strictly more expressive than CTL.

Proof sketch: The existence of the desired translation follows from the following fixpoint characterizations of the basic CTL modalities: \(EXP = <A>P, EFP = mu Y.P \lor <A>Y, EGP = nu Y.P \land <A>Y, and E[P \cup Q] = mu Y.[Q \lor (P \land <A>Y)]. To see that CTL is strictly less expressive than \(L_{mu_1}\)
over 1 program letter, observe that the formula $\nu Y. [P \land <A> <A> Y]$ which asserts that $p$ occurs every other time is not expressible in CTL (cf. [Wo81]).

It can be shown that CTL cannot express correctness properties such as $\text{EF}^P$ (along some path $P$ occurs infinitely often) which are important in dealing with fairness (cf. [EH83]). Thus, in [EL85] we extended CTL to Fair CTL (FCTL). An FCTL specification $(p_0, \Phi_0)$ consists of a functional assertion $p_0$, which is a state formula, and an underlying fairness assumption $\Phi_0$, which is a pure path formula. The functional assertion $p_0$ is expressed in essentially CTL syntax with basic modalities of the form either $A_\phi$ ("for all fair paths") or $E_\phi$ ("for some fair path") followed by one of the linear time operators $F$, $G$, $X$, or $U$. We subscript the path quantifiers with the symbol $\phi$ to emphasize that they range over paths meeting the fairness constraint $\Phi_0$, and to syntactically distinguish FCTL from CTL. A fairness constraint $\Phi_0$ is a boolean combination of the infinitary linear time operators $\mathcal{F}^i_p$ ("infinitely often $p$") and $\mathcal{G}^i_p$ ("almost always $p$"), applied to propositional arguments (cf. [EC80], [Ha86]). We can then view a subformula such as $A_{\phi} \mathcal{F}^i_p$ of functional assertion $p_0$ as an abbreviation for the CTL* formula $A[\Phi_0 \Rightarrow F_p]$. Similarly, $E_{\phi} \mathcal{G}^i_p$ abbreviates $E[\Phi_0 \land G_p]$.

One key result of [EL85] was that almost all practical types of fairness notions can be succinctly expressed using the canonical form $\Phi = \bigwedge_{i=1}^n \left( \mathcal{F}^i_{p_i} \lor \mathcal{G}^i_{q_i} \right)$. We will show, in this section, how the fairness canonical form can be succinctly expressed in $L_{\mu_2}$. In the following discussion we take the liberty of slightly mixing CTL and $L_{\mu}$ operators.

**Proposition 4.8:** $E\Phi = E\nu Y. \pi(Y)$.

**Proof:** We first establish the following lemmas:

**Lemma 4.9:** $\nu Y. \pi(Y) \subseteq E\Phi$.

**Proof:** We will argue that if $Y = \pi(Y)$ then $Y \subseteq E\Phi$, from which it will follow that $\nu Y. \pi(Y) \subseteq E\Phi$.

Suppose $s_0 \in Y$. Then for each $j \in [1:n]$, $s_0 = \text{EXE}(Y \cup (p_j \land Y)) \lor (q_j \land \text{EXY})$. Consider a fixed $j \in [1:n]$. We have two cases:

Case (a): $s_0 = \text{EXE}(Y \cup (p_j \land Y))$, in which case there exists a finite sequence $(s_0 = t_0, t_1, \ldots, t_k = s_1)$ of states, each satisfying $Y$, which starts at $s_0$ and ends at state $s_1$ such that $s_1 \models p_j \land Y$. Denote this sequence by $s_0 \xrightarrow{i_0:a} \cdots \xrightarrow{i_k:a} s_1$.

Case (b): $s_0 = (q_j \land \text{EXY}) \land \neg\text{EXE}(Y \cup (p_j \land Y))$, in which case $s_0$ has a successor $s_1$ such that $s_1 \models Y$. Denote this sequence $(s_0, s_1)$ by $s_0 \xrightarrow{i_0:b} s_1$.

---

7 By Theorem 4.7 we can view CTL formulae as abbreviations for $L_{\mu_1}$ formulae if we wish
In either case, there is a finite sequence from \( s_0 \) to \( s_1 \), which we can denote by \( s_0 \rightarrow s_1 \), where for simplicity we suppress which case applies.

We will construct an infinite path \( x \) of the form \( s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots \) by splicing together segments of the form \( s_i \rightarrow s_{i+1} \) in such a way as to ensure that \( x \models \Phi \). The ground rules are (1) at time \( i \), try to handle \( p_k \), \( q_k \) where \( k = 1 + i \mod n \), and (2) always favor Case (a) to Case (b), if possible. So initially the path is just \( s_0 \). Inductively, the path is of the form \( s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_i \). Let \( k = 1 + i \mod n \). If \( s_i \models \text{EXE}[Y \cup (p_k \land Y)] \) (case (a)) then take \( s_i \rightarrow s_{i+1} \) to be \( s_i \rightarrow_{k,b} s_{i+1} \); otherwise (case (b)) take \( s_i \rightarrow s_{i+1} \) to be \( s_i \rightarrow_{k,a} s_{i+1} \). Let \( x \) be the infinite path \( s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots \) obtained by repeating this process for each \( i \in \mathbb{N} \). Observe that, since \( s_0 \models Y \) and \( Y \) holds of each state of each \( s_i \rightarrow s_{i+1} \), (1) \( Y \) holds of every state along \( x \).

We claim that if \( s_i \models \text{EXE}[Y \cup (p_k \land Y)] \) then \( q_k \) holds of every state along the infinite suffix \( x' = s_i \rightarrow s_{i+1} \rightarrow s_{i+2} \ldots \) of \( x \). To see this, assume \( s_i \models \text{EXE}[Y \cup (p_k \land Y)] \) and suppose the claim is false. Let \( t \) be the first state along \( x' \) after \( s_i \) such that \( q_k \) is false at \( t \). Since \( Y \) holds at every state along \( x \), \( t \models Y \). Since \( Y = r(\tau) \), \( t \models \text{EXE}[Y \cup (p_k \land Y)] \). However, every state between \( s_i \) and \( t \) satisfies \( Y \). This implies that \( s_i \models \text{EXE}[Y \cup (p_k \land Y)] \) contradicting our assumption that \( s_i \models \text{EXE}[Y \cup (p_k \land Y)] \). Thus, each state along \( x' \) satisfies \( q_k \), thereby establishing the Lemma.

It is now easy to verify that the path \( x \) thus constructed satisfies \( \Phi \): For each \( j \in [1:n] \), either Case (a) applied at every time \( i \) for which \( 1 + i \mod n = j \) -- so that \( x \models \text{F}^\omega p_i \) -- or eventually there is a time \( i \), such that \( 1 + i \mod n = j \) and Case (b) applies -- so that, by the claim, \( x \models \text{G}^\omega q_i \). Hence, \( Y \subseteq \text{E} \Phi \) for any fixpoint \( \tau \) of \( r \). This completes our proof.

\[ \text{Lemma 4.10:} \ E\Phi' \subseteq \nu \cdot Y \cdot r(Y). \]

**Proof:** It suffices to show that \( E\Phi' \subseteq r(E\Phi') \) since \( \nu \cdot Y \cdot r(Y) = \bigcup \{Z : Z \subseteq r(Z)\} \). Suppose \( s_0 \models E\Phi' \). Then there exists a path \( x = (s_0, s_1, s_2, \ldots) \) such that \( x \models \Phi' \). Moreover, for each state \( s_j \) on \( x \), \( s_j \models E\Phi' \). We will show that \( s_0 \models r(E\Phi') \).

For each \( i \in [1:n] \), if \( x \models \text{G} \cdot q_i \) then \( s_0 \models q_i \land \text{EXE} \Phi' \). Otherwise, \( x \models \text{F}^\omega p_i \). In this case, \( s_0 \models \text{EXE}[E\Phi' \cup (p_i \land E\Phi')] \). Thus, \( s_0 \models \text{EXE}[E\Phi' \cup (p_i \land E\Phi')] \lor (q_i \land \text{EXE} \Phi') \) for each \( i \in [1:n] \). I.e., \( s_0 \models \bigwedge_{i=1}^n \text{EXE}[E\Phi' \cup (p_i \land E\Phi')] \lor (q_i \land \text{EXE} \Phi') \) = \( r(E\Phi') \). Therefore, \( E\Phi' \subseteq \nu \cdot Y \cdot r(Y) \).

**Lemma 4.11:** \( \text{E} \Phi \cdot \text{E} \Phi = \text{E} \Phi \).

**Proof:** Obvious. Left to the reader. \( \square \)

**Lemma 4.12:** \( \text{E} \Phi \cdot \text{E} \Phi = \text{E} \Phi \).

**Proof:** Since \( E\Phi' \subseteq E\Phi \), \( \text{E} \Phi \cdot \text{E} \Phi' \subseteq \text{E} \Phi \Phi \) by monotonicity of \( \text{E} \Phi \) operator. Hence, \( \text{E} \Phi \cdot \text{E} \Phi' \subseteq E\Phi \) by Lemma 4.11.
If \( s_0 \models E\Phi \), then there exists a fullpath \( x = (s_0, s_1, s_2, \ldots) \) such that \( x \models \Phi \). It is quite straightforward to verify that there exists a state \( s_j \) on \( x \) such that \( s_j \models E\Phi' \). Thus \( s_0 \models EFE\Phi' \). Therefore, \( EFE\Phi' \supseteq E\Phi \).

To complete the proof of Proposition 4.8, we use the following "squeezing" argument to show that \( E\Phi \models Y.\pi(Y) = E\Phi \). Since \( \phi \subseteq \nu Y.\pi(Y) \) (Lemma 4.10) \( \subseteq E\Phi \) (Lemma 4.9), we have \( E\Phi = EFE\Phi' \) (Lemma 4.12) \( \subseteq E\nu Y.\pi(Y) \) (by monotonicity of \( EF \) \( \subseteq EFE\Phi = E\Phi \) (Lemma 4.11).

Thus \( E\Phi \subseteq E\nu Y.\pi(Y) \subseteq E\Phi \), and the theorem is proved.

**Remark:** We feel that the existence of a succinct fixpoint characterization in \( \mu_2 \) of \( E\Phi \) is a rather nonobvious point. Now, in the case where \( \Phi \) contains only a single conjunct, then \( E\Phi \) pretty obviously has the fixpoint characterization \( \nu Y.EGEF((\Phi_1 \land Y) \lor E\Phi(p_1 \land Y)) \). But this fixpoint characterization does not appear to generalize to \( n > 1 \) conjuncts. An alternative approach would be to multiply out the \( n \) conjuncts, but then the resulting formula is of length about \( 2^n \) and the fixpoint characterization will not be succinct.

We also point out that our succinct fixpoint characterization is correct for arbitrary sets of states \( p_1 \), \( q_1 \), and not just in the (easier) case when, e.g., \( p_1 = executed_i \), and \( q_i = \neg enabled_i \) (cf. [EC80], [deR81].)

**Theorem 4.13:** There is a succinct, efficient (i.e., linear size and computable in linear time) translation of FCTL into \( \mu_2 \) over 1 program letter. Moreover, \( \mu_2 \) over 1 program letter is strictly more expressive than FCTL.

**Proof Sketch:** Since all temporal operators in FCTL can be defined in terms of the basic modalities \( E\Phi Xq, E\Phi Gq \) and \( E\Phi[p U q] \), it suffices to show how to encode these formulae in \( \mu_2 \). For \( E\Phi Xq \) and \( E\Phi[p U q] \), it is quite straightforward to get the fixpoint characterizations because \( E\Phi Xq \equiv EX(q \land E\Phi) \), \( E\Phi[p U q] \equiv E[p U (q \land E\Phi)] \), and we have the fixpoint characterization of \( E\Phi \) in \( \mu_2 \) by Proposition 4.8.

It remains to show how \( E\Phi Gq \) is expressed in \( \mu_2 \). It is not hard to verify that \( E\Phi Gq \equiv E[q U E(Gq \land \Phi')] \). Using the same squeezing technique used to establish Proposition 4.8 above, we can show that the corresponding \( \mu_2 \) formula for \( E[q U E(Gq \land \Phi')] \) is \( E[q U \nu Y.\bigwedge_{i=1}^{n} [EXE(Y U (p_i \land Y)) \lor (q_i \land EXY)] \land q] \).

Finally, it can be shown that the \( \mu_2 \) (in fact, \( \mu_1 \)) formula \( \nu Y.p \land AXAXY \) is not expressible in CTL*, and hence not in FCTL.

Note that since the canonical form FCTL is as expressive as the full FCTL (but less succinct), we conclude that \( \mu_2 \) is more expressive than FCTL.

---

[8] The reader who is familiar with GFCTL will note that Theorem 4.13 also holds for GFCTL as well as FCTL (cf. [EL85]).
5. Conclusion

In this paper we have considered the problem of model checking in the propositional Mu-calculus $L_\mu$. It provides a least fixpoint operator ($\mu$) and greatest fixpoint operator ($\nu$) which make it possible to give fixpoint characterizations of temporal modalities. For the full logic $L_\mu$ our model checking algorithm has exponential time worst case complexity. However, we show that for each $k$, for the restricted fragment $L_{\mu_k}$ where the depth of alternating nestings of $\mu$'s and $\nu$'s is at most $k$, our algorithm runs in polynomial time with the degree of the polynomial $= k+1$. It turns out that these restricted fragments of the Mu-Calculus have considerable expressive power. Indeed, we show that PDL and CTL can be succinctly translated into (and are strictly less expressive than) $L_{\mu_1}$ while PDL-$\Delta$ and FCTL can be succinctly translated into (and are strictly less expressive than) $L_{\mu_2}$. (CTL$^*$ can be translated into $L_{\mu_2}$ also, but our translation is not succinct.) In practice, we therefore have an efficient model checking algorithm for a language $L_{\mu_2}$ which subsumes a number of commonly used program logics. An additional advantage is that $L_{\mu_2}$ provides a uniform framework for handling fairness constraints as in [EL85], extended temporal operators similar to those in [Wo83], and (branching) past-tense modalities (through the use of converse relations.)

Appendix.
A.1: PDL-$\Delta$ and FCTL.

For the convenience of the reader, we briefly describe the syntax and semantics of PDL-$\Delta$ and FCTL in this appendix.

For PDL-$\Delta$, we are given a set $\Psi_0$ whose elements are called atomic propositions and a set $\Pi_0$ whose elements are (atomic) program letters. The set of programs, $\Pi$, and the set of formulae, $\Psi$, of PDL-$\Delta$ are then defined inductively as follows:

(P1) $\Pi_0 \subseteq \Pi$
(P2) if $a, b \in \Pi$ then $a;b, a\cup b, a^* \in \Pi$
(P3) if $p \in \Psi$ then $p? \in \Pi$
(F1) $\Psi_0 \subseteq \Psi$
(F2) if $p \in \Psi$ then $\neg p \in \Psi$
(F3) if $a \in \Pi$ and $p \in \Psi$ then $<a>p, \Delta a \in \Psi$

Sentences of PDL-$\Delta$ are interpreted with respect to a structure $M = (S, R, L)$ as defined in section 2.3. Intuitively, $\Pi$ is a set of regular languages over $\Pi_0$ and tests. If $p$ is a formula, the test $p?$ is a program equivalent to 'if $p$ then skip else abort'. Thus $<p?q$ is true at a state $s$ iff $p \land q$ is true at $s$. If $a \in \Pi$, the formula $<a>p$ is true at a state $s$ when there is a a path labelled by some word matching the regular expression $a$ leading from $s$ to a state satisfying $p$, and the formula $\Delta a$ is true at a state $s$ when there is an infinite chain of edges labelled with an infinite word from the $\omega$-regular language $a^\omega$. 

16
Basic modalities of FCTL are formed by fair path quantifiers $A_\phi$ (along every fair computation path satisfying the underlying fairness constraint $\phi_0$) and $E_\phi$ (along some fair path) followed by one of the usual linear time temporal operators $X$ (nexttime), $F$ (sometime), $G$ (always), or $\mathcal{U}$ (until). Formally, formulae of FCTL are formed as follows:

1. Any atomic proposition $P$ is a formula.
2. If $p$, $q$ are formulae then so are $\neg p$, and $(p \land q)$.
3. If $p$, $q$ are formulae then so are $A_\phi X p$, $E_\phi X p$, $A_\phi [p \mathcal{U} q]$, and $E_\phi [p \mathcal{U} q]$.

A propositional formula is one formed by rules 1, 2 above. A fairness constraint is then formed by the following rules:

4. If $p$, $q$ are propositional formulae then $\forall^\infty p$ (infinitely often $p$), $\exists^\infty p$ (almost always $p$) are 
   fairness constraints.
5. If $p$, $q$ are fairness constraints then so are $\neg p$, and $(p \land q)$.

The other connectives can then be defined as abbreviations in the usual way: $A_\phi F q$ abbreviates $A_\phi [true \ U q]$, $E_\phi F q$ abbreviates $E_{\phi} [true \ U q]$, $A_\phi G q$ abbreviates $\neg E_\phi F \neg q$, $E_\phi G q$ abbreviates $\neg A_\phi F \neg q$. Again, formulae of FCTL is interpreted in a structure $M = (S, \mathcal{R}, L)$ as defined in section 2.3 except that $\mathcal{R}$ is simply a binary relation over $S$. For example, the formula $E_\phi F p$ is true at a state $s$ when there is a path starting at $s$ which meets the underlying fairness constraint $\phi_0$ and $p$ is true at some state of the path. The formulae $A_\phi [p \mathcal{U} q]$ is true at a state $s$ iff along every fair path starting at $s$, $p$ holds until $q$ becomes true. The reader is referred to [EL85] for more details.

We define the language $LX$ to be the set of formulae formed by the rules of $L_\mu$, FCTL, and PDL-$\Delta$. Since FCTL and PDL-$\Delta$ are just sublanguages of $L_\mu$, $LX$ is not more expressive than $L_\mu$. However, it is clear that by allowing mixing FCTL, PDL-$\Delta$, and $L_\mu$ modalities the task of specification will be much more easier. Moreover, it is straightforward to modify our model checking algorithm to accept formulae in LX.

A.2: Proof of Theorem 4.1.

The proof is by induction on the lexicographic order of $(\#\Delta(f), |f|, ld(f))$, where $\#\Delta(f)$ is the number of $\Delta$-operators in $f$, and $ld(f)$ is the length of the leading diamond of $f$ (i.e., $ld(f) = |a|$ if $f = <a> p$, otherwise $ld(f) = 0$).

In the following, we will use $p$ instead of $p(\mathcal{V})$ if the valuation $\mathcal{V}$ is obvious.

1. If $f = P$ (an atomic propositional constant), clearly $M, s \models P$ iff $M, s \models P^t$.
2. If $f = X_i$ (an atomic propositional variable), $M, s \models X_i(\mathcal{V})$ iff $M, s \models X_i^t(\mathcal{V})$.
3. $f = \neg p$: By induction hypothesis, $M, s \models p$ iff $M, s \models p^t$. Therefore, $M, s \models \neg p$ iff $M, s \models \neg p^t$ iff $M, s \models (\neg p)^t$. 

17
4. \( f = \langle A \rangle p \): By induction hypothesis, \( M, s \models \langle A \rangle p \iff \exists (s, t) \in \mathbb{R}(A)[M, t \models p] \iff \exists (s, t) \in \mathbb{R}(A)[M, t \models p^t] \) iff \( M, s \models \langle A \rangle p^t \) iff \( M, s \models \langle A \rangle p^t \).

5. \( f = \langle a \cup b \rangle p \): By induction hypothesis, \( M, s \models \langle a \rangle p \) iff \( M, s \models \langle a \rangle p^t \), and \( M, s \models \langle b \rangle p \) iff \( M, s \models \langle b \rangle p^t \). Therefore, \( M, s \models \langle a \cup b \rangle p \) iff \( M, s \models \langle a \rangle p \) or \( M, s \models \langle b \rangle p \) iff \( M, s \models \langle a \rangle p^t \) or \( M, s \models \langle b \rangle p^t \) iff \( M, s \models \langle a \cup b \rangle p^t \).

6. \( f = \langle p? \rangle q \): By induction hypothesis, \( M, s \models p \) iff \( M, s \models p_t \), and \( M, s \models q \) iff \( M, s \models q_t \). Therefore, \( M, s \models \langle p? \rangle q \) iff \( M, s \models p \land q \) iff \( M, s \models p^t \) and \( M, s \models \langle p \land q \rangle \) iff \( M, s \models \langle p \rangle \) by induction hypothesis.

7. \( f = \langle a; b \rangle p \): \( M, s \models \langle a; b \rangle p \) iff \( M, s \models \langle a \rangle \langle b \rangle p \). Note that \( \#\Delta(\langle a; b \rangle p) = \#\Delta(\langle a \rangle \langle b \rangle p), |\langle a; b \rangle p| = |\langle a \rangle \langle b \rangle p|, \) and \( \text{ld}(\langle a \rangle \langle b \rangle p) < \text{ld}(\langle a; b \rangle p) \). Therefore, \( M, s \models \langle a; b \rangle p \) iff \( M, s \models \langle a \rangle \langle b \rangle p \) by induction hypothesis.

8. \( f = \langle a^* \rangle p \): Since \( \langle a^* \rangle p \) is the least fixpoint of \( p \lor \langle a \rangle Y, M, s \models \langle a^* \rangle p \) iff \( M, s \models \mu Y.[p \lor (\langle a \rangle Y)] \). By induction hypothesis, \( M, s \models p \) iff \( M, s \models p^t \) and \( M, s \models \langle a \rangle Y \) iff \( M, s \models (\langle a \rangle Y)^t \). Therefore, \( M, s \models \langle a^* \rangle p \) iff \( M, s \models \mu Y.[p^t \lor (\langle a \rangle Y)^t] \)

9. \( f = \Delta a \): Since \( \Delta a \) is the greatest fixpoint of \( \langle a \rangle Y, M, s \models \Delta a \) iff \( M, s \models \nu Y.\langle a \rangle Y \). By induction hypothesis, \( M, s \models \langle a \rangle Y \) iff \( M, s \models (\langle a \rangle Y)^t \). Therefore, \( M, s \models \Delta a \) iff \( M, s \models \nu Y.(\langle a \rangle Y)^t \).

References


Acknowledgement We would like to thank Amir Pnueli for helpful discussions which helped refine our understanding of the significance of the Mu-calculus.
Figure 2-1: Active Variables in Mu-Calculus
Algorithm 3-1: model checking for a $\mu$-calculus sentence

Input: given a structure $M = (S, \mathcal{R}, L)$, and a sentence $p_0$ which contains variables $Y_1$, $Y_2$, ..., $Y_n$

Output: determine whether $M$ is a model for $p_0$

Step 1. Convert $p_0$ to its equivalent PNF $p'_0$.

Step 2. $S' = \text{eval}(p'_0); /* Compute the set of states at which } p'_0 \text{ holds */}

Step 3. if $S' \neq \emptyset$ then $M$ is a model for $p_0$ else $M$ is not a model for $p_0$

Recursive function \( \text{eval}(f) \); var $S'$, $S_n$;

/* return the set of states which satisfy $f$ */

begin

case $f$ of the form

P (atomic prop.): $S' = \{s \in S: P \in L(s)\}$;

$Y_i$ (\(\sigma\)-variable): $S' = S_i$

$\neg p$: $S' = S \setminus \text{eval}(p)$;

$p \land q$: $S' = \text{eval}(p) \cap \text{eval}(q)$;

$p \lor q$: $S' = \text{eval}(p) \cup \text{eval}(q)$;

$\langle A \rangle p$: $S_n = \text{eval}(p)$; $S' = \{s \in S: \exists t \in S_n[(s, t) \in \mathcal{R}(A)]\}$;

$[A] p$: $S_n = \text{eval}(p)$; $S' = \{s \in S: \forall t \in S[(s, t) \in \mathcal{R}(A) \Rightarrow t \in S_n]\}$;

$\mu Y_i.p_i$:

begin

$S_i = \emptyset$

repeat $S' = S_i$; $S_i = \text{eval}(p_i)$ until $S' = S_i$;

end;

$\nu Y_i.p_i$:

begin

$S_i = S$

repeat $S' = S_i$; $S_i = \text{eval}(p_i)$ until $S' = S_i$;

end;

end;

return($S'$);

end.

---

Figure 3-1: Model Checker for Mu-Calculus

(a) Parse Tree

```
\[ \forall \langle a \rangle \langle b \rangle \]  
\[ \langle a \rangle \langle b \rangle \]  
\[ \langle a \rangle \]  
\[ \langle b \rangle \]  
\[ p \]  
```

(b) Parse Graph

---

Figure 4-1: Representation of $\langle a \rangle p \lor \langle b \rangle p$
Algorithm 3-2: model checking for a μ-calculus sentence

Input: given a structure \( M = (\mathcal{S}, \mathcal{R}, \mathcal{L}) \), and a sentence \( p_0 \) which contains variables \( Y_1, Y_2, \ldots, Y_n \)

Output: determine whether \( M \) is a model for \( p_0 \)

Step 1. Convert \( p_0 \) to its equivalent PNF \( p'_0 \).
Step 2. for \( i = 1 \) to \( n \) do if \( Y_i \) is a μ-variable then \( S_i = \emptyset \) else \( S_i = S \);
   
   /* set appropriate initial conditions for iteration */
   for each subformula \( q \) of \( p'_0 \) do already-eval(\( q \)) = false;
   /* already-eval() is a global boolean array used to prevent redundant evaluation
   of a subformula. When a subformula \( q \) of \( p'_0 \) is evaluated, its value is kept in the global
   array value(). Whenever the value of \( q \) is needed again we simply return value(\( q \)). */

Step 3. \( S' = \text{eval}(p'_0) \); /* Compute the set of states at which \( p'_0 \) holds */
Step 4. if \( S' \neq \emptyset \) then \( M \) is a model for \( p_0 \) else \( M \) is not a model for \( p_0 \)

Recursive function eval(f); var \( S', S'' \);

/* return the set of states which satisfy \( f \) */
begin
   if already-eval(f) then return(value(f));
   case \( f \) of the form
     \( P \) (atomic prop.): \( S' = \{ s \in S : P \in L(s) \} \);
     \( Y_i \) (σ-variable): \( S' = S_i \);
     \( \neg p \): \( S' = S \setminus \text{eval}(p) \);
     \( p \land q \): \( S' = \text{eval}(p) \cap \text{eval}(q) \);
     \( p \lor q \): \( S' = \text{eval}(p) \cup \text{eval}(q) \);
     \( \langle A \rangle p \): \( S'' = \text{eval}(p) ; S' = \{ s \in S : \exists t \in S''[(s,t) \in R(A)] \} \);
     \( [A]p \): \( S'' = \text{eval}(p) ; S' = \{ s \in S : \forall t \in S''[(s,t) \in R(A) \Rightarrow t \in S''] \} \);
     \( \mu Y_i p_i (\ldots, Y_i, \ldots) \):
       begin
         if the surrounding σ-formula of \( \mu Y_i \) is νZ
         then \( S_i = \emptyset \);
           for each open subformula \( \mu Y_j \) of \( \mu Y_i \) such that there is no
           subformula \( \nu W \) of \( \mu Y_i \) containing \( \mu Y_j \), set \( S_j = \emptyset \);
         repeat \( S' = S_i ; S_i = \text{eval}(p_i) \) until \( S' = S_i \);
         end;
     \( \nu Y_i p_i (\ldots, Y_i, \ldots) \):
       begin
         if the surrounding σ-formula of \( \nu Y_i \) is \( \mu Z \)
         then \( S_i = S \);
           for each open subformula \( \nu Y_j \) of \( \nu Y_i \) such that there is no
           subformula \( \mu W \) of \( \nu Y_i \) containing \( \nu Y_j \), set \( S_j = S \);
         repeat \( S' = S_i ; S_i = \text{eval}(p_i) \) until \( S' = S_i \);
         end;
       if \( f \) is a sentence then begin already-eval(f) = true; value(f) = \( S' \) end;
     return(\( S' \));
   end.

Figure 3-2: Improved Model Checker for Mu-Calculus