

Quotient-based Control Synthesis for Non-Deterministic Plants with Mu-Calculus Specifications

Samik Basu and Ratnesh Kumar

Abstract— We study the control of a nondeterministic discrete event system (DES) subject to a control specification expressed in the propositional mu-calculus, under complete observation of events. Given a plant automaton model and a mu-calculus specification we provide a set of rules that computes the “quotient” of the specification against the plant, which is another mu-calculus formula such that a supervisor exists if and only if the quotiented formula is satisfiable. Thus the control problem is reduced to one of mu-calculus satisfiability. We also present a tableau-based satisfiability solving algorithm that identifies a model for the quotiented formula. The resulting model serves as a supervisor. The complexity of supervisor existence verification as well as model synthesis is single exponential in the size of the plant as well as the size of the specification formula.

I. INTRODUCTION

Supervisory control problem for discrete event systems involves computing a supervisor (if one exists) such that when composed with a given plant, the composed system conforms to a desired specification. The framework of DES control was presented in [9] in which the plant and the specification were modeled as deterministic automata. Since then extensions to allow more general nondeterministic models of plant and more general temporal-logic/bisimulation-based specifications have been considered.

The problem of control of a deterministic plant subject to a mu-calculus specification was first addressed by Arnold, Vincent, and Walukiewicz in [2], where the authors also allowed time-varying uncontrollable actions and “projection-type” partial observation function. We study the same problem allowing nondeterminism in the plant model and more general state-based uncontrollable events but assume a complete observation of events. Also there are similarities and differences between the two solution approaches. The similarities are that we both employ the technique of “quotienting”, and reduce the problem of control to that of a satisfiability problem. The differences are: (i) Quotienting is performed at the level of mu-calculus formulas in our setting (this lets us handle the plant nondeterminism in a simple manner), while it is performed at the level of their alternating tree automata representations in the setting of Arnold et al.; (ii) We absorb the control-compatibility requirements as part of the

quotienting (this lets us handle state-based such requirements in a straightforward manner), while Arnold et al. impose this as a requirement of the quotiented automata. Riedweg and Pinchinat [10], [8] also presented an automata-theoretic quotienting approach for supervisor synthesis for enforcing mu-calculus specifications. An extension of mu-calculus, called *quantified mu-calculus*, to express the existence of a supervisor as part of the specification logic was introduced. We avoid capturing the supervisor existence as part of the specification logic and delegate that to the satisfiability of the quotiented property. Furthermore, as with [2], the work in [8] is limited to deterministic plants.

Our quotienting procedure is closely related to the ones described in [1], [3], where quotienting of equational mu-calculus formula expressions against labeled transition systems and CCS processes is described for model checking systems with regular structures (e.g rings) and unbounded subsystems (parameterized systems) respectively. In this paper, we define the quotienting operation for mu-calculus against labeled transition systems in the domain of supervisory controller problem taking into consideration the controllability requirement and also any nondeterminism in the plant model.

In our setting the satisfiability of the quotiented formula is used for determining controller existence, and a model satisfying the quotiented formula serves as a controller. A number of works have discussed the problem of satisfiability checking of mu-calculus formulas; they rely on reducing the satisfiability problem to emptiness problem of alternating tree automata [4] or identifying a winning strategy in parity games [11], [2]. In this paper, we describe a tableau-based mu-calculus satisfiability checking and model identification technique which is inspired from [6]. The authors in [6] developed an equivalent disjunctive form of a mu-calculus formula for which satisfiability is easily verified. In contrast, our satisfiability checking and model identification works directly on a mu-calculus formula. The main reason for presenting our own approach for checking satisfiability and model identification is that it is self-contained in its treatment: Care has been taken to keep track of the appropriate histories that allows construction of the successor tableau nodes from the current tableau nodes, and also ensures the finiteness of the tableau construction.

II. PRELIMINARIES

A. Propositional Mu-calculus Specifications

The Propositional mu-calculus [7], [5] is an expressive temporal logic with explicit greatest and least fixed point

The research was supported in part by the National Science Foundation under the grants NSF-ECS-0218207, NSF-ECS-0244732, NSF-EPNES-0323379, NSF-ECS-0424048, and NSF-ECS-0601570.

Samik Basu and Ratnesh Kumar are with Iowa State University, Dept. of Computer Science and Electrical & Computer Eng., respectively. Email: {sbasu, rkumar}@iastate.edu.

We are thankful to Prof. Andre Arnold who pointed out some errors in an initial draft caused by an improper handling of the alternations in the tableau-rules.

operators. Properties in temporal logics LTL, CTL, and CTL* can be encoded as mu-calculus formulas. The syntax of mu-calculus formulas is defined over a domain (AP, \mathcal{X}, A) consisting of a set of atomic propositions AP , a set of formula variables \mathcal{X} , and a set of actions A , and is given by the following grammar:

$$\phi \rightarrow \text{tt} \mid \text{ff} \mid p \mid X \mid \phi \wedge \phi \mid \phi \vee \phi \mid \langle a \rangle \phi \mid [a] \phi \mid \sigma X. \phi$$

In the above, $p \in AP, X \in \mathcal{X}, a \in A$; and tt and ff denote the formulas that are always “True” and “False” respectively; $\langle a \rangle$ and $[a]$ denote the “diamond” and “box” operators representing “exists a -successor” and “all a -successor” modalities respectively. We also use the notation $\langle \rangle$ to denote either $\langle \rangle$ or $[\]$. $\sigma X. \phi$ represents a fixed point formula expression and in this case X is said to be *bound* in the fixed point formula expression. Here $\sigma \in \{\mu, \nu\}$ and μ and ν denote the least and greatest fixed point operations respectively. The set of variables in a formula that are not bound are called *free*. The set of all mu-calculus formulas defined over the domain (AP, \mathcal{X}, A) is denoted $\Phi[AP, \mathcal{X}, A]$. For a formula φ , $FV(\varphi)$ denotes its set of free-variables, $Sub(\varphi)$ denotes its set of sub-formulas, $|\varphi|$, called length of φ , denotes the number of boolean and modal operators in φ , and $ad(\varphi)$, called alternation depth of φ , denotes the number of nesting between μ and ν in φ . $ad(\varphi)$ is recursively defined as:

- $ad(\text{tt}) = ad(\text{ff}) = ad(X) = 0$
- $ad(\varphi \wedge \psi) = ad(\varphi \vee \psi) = \max\{ad(\varphi), ad(\psi)\}$
- $ad([a]\varphi) = ad(\langle a \rangle \varphi) = ad(\varphi)$
- $ad(\sigma X. \varphi) = \max \left(\begin{array}{l} \{1, ad(\varphi)\} \cup \\ \{ad(\sigma_y Y. \varphi_y) + 1 \mid \\ (\sigma_y Y. \varphi_y \in Sub(\varphi)) \\ \wedge (X \text{ free in } \varphi_y) \\ \wedge (\text{fp}(X) \neq \text{fp}(Y))\} \end{array} \right)$

where for $\sigma X. \varphi$, $\text{fp}(X) = \sigma$. We use $nd(\varphi)$ to denote the nesting depth, i.e., the number of nestings of fixed point expressions in φ . The definition is identical to that for $ad(\varphi)$ except when $\varphi = \sigma X. \varphi$.

$$nd(\sigma X. \varphi) = \max \left(\begin{array}{l} \{1, nd(\varphi)\} \cup \\ \{nd(\sigma_y Y. \varphi_y) + 1 \mid \\ (\sigma_y Y. \varphi_y \in Sub(\varphi))\} \end{array} \right)$$

The semantics of a mu-calculus formula φ is a set of states of a labeled transition system (LTS) M where the formula holds under a given environment e , and is denoted by $\llbracket \varphi \rrbracket_e^M$. Here the LTS M is a 5-tuple, (S, A, T, AP, L) where S is the set of states, $T \subseteq S \times A \times S$ is the set of transitions labeled by actions in A and $L : S \rightarrow 2^{AP}$ is the labeling function which maps states to sets of propositions. e is a map of the form, $e : \mathcal{X} \rightarrow 2^S$ that associates with each of the formula variables a set of states and is used for defining the semantics of the variables. The LTS is typically understood from the context and so instead of writing $\llbracket \varphi \rrbracket_e^M$, we only write $\llbracket \varphi \rrbracket_e$, which is recursively defined in Fig. 1. In Fig. 1, $e[X \mapsto \hat{S}]$ denotes the environment e with the substitution that associates the state set $\hat{S} \subseteq S$ with the variable $X \in \mathcal{X}$. In other words for $Y \in \mathcal{X}$, $e[X \mapsto \hat{S}](Y)$ equals \hat{S} if $Y = X$, and $e(Y)$ otherwise.

1.	$\llbracket \text{tt} \rrbracket_e$	=	S
2.	$\llbracket \text{ff} \rrbracket_e$	=	\emptyset
3.	$\llbracket p \rrbracket_e$	=	$\{s \mid p \in L(s)\}$
4.	$\llbracket X \rrbracket_e$	=	$e(X)$
5.	$\llbracket \varphi_1 \wedge \varphi_2 \rrbracket_e$	=	$\llbracket \varphi_1 \rrbracket_e \cap \llbracket \varphi_2 \rrbracket_e$
6.	$\llbracket \varphi_1 \vee \varphi_2 \rrbracket_e$	=	$\llbracket \varphi_1 \rrbracket_e \cup \llbracket \varphi_2 \rrbracket_e$
7.	$\llbracket \langle a \rangle \varphi \rrbracket_e$	=	$\{s \mid \exists s' \xrightarrow{a} s' \wedge s' \in \llbracket \varphi \rrbracket_e\}$
8.	$\llbracket [a] \varphi \rrbracket_e$	=	$\{s \mid \forall s' \xrightarrow{a} s' \Rightarrow s' \in \llbracket \varphi \rrbracket_e\}$
9.	$\llbracket \mu X. \varphi \rrbracket_e$	=	$\cap \{ \hat{S} \mid \llbracket \varphi \rrbracket_{e[X \mapsto \hat{S}]} \subseteq \hat{S} \}$
10.	$\llbracket \nu X. \varphi \rrbracket_e$	=	$\cup \{ \hat{S} \mid \hat{S} \subseteq \llbracket \varphi \rrbracket_{e[X \mapsto \hat{S}]} \}$

Fig. 1. Semantics of mu-calculus formula

We model a discrete event system to be controlled, called a plant, as well as a supervisor - for controlling it - as LTSs with certain initial or start states, i.e., “initialized LTSs”. An LTS M with initial states $S_0 \subseteq S$ is said to satisfy a formula φ if there exists an environment e such that $S_0 \subseteq \llbracket \varphi \rrbracket_e$.

B. Supervisory Control

An uncontrolled discrete event plant P is modeled as an initialized LTS, $P = (S_P, A, \delta_P, AP, L_P, S_{0,P})$, where S_P is the finite set of states; A is the finite set of actions or events. At each state $s \in S_P$, the actions set is partitioned into a set of controllable actions $A_c(s)$, and a set of uncontrollable actions $A_u(s)$. $\delta_P \subseteq S_P \times A \times S_P$ is the set of transitions; $(s, a, s') \in \delta_P$ is said to be uncontrollable if $a \in A_u(s)$, and otherwise it is said to be controllable. AP is a finite set of atomic proposition; $L_P : S_P \rightarrow 2^{AP}$ is a labeling function; and $S_{0,P} \subseteq S_P$ is the set of initial states.

A supervisor $C = (S_C, A, \delta_C, AP, L_C, S_{0,C})$, is defined as another initialized LTS. Note that P and C share the sets of actions (A) and atomic propositions (AP). The controlled plant is obtained by the strict synchronous composition of P and C , denoted by $P||C$, which is defined as: $(S_{P||C}, A, \delta_{P||C}, AP, L_{P||C}, S_{0,P||C})$, where $S_{P||C} = S_P \times S_C$ is the state set; A and AP are the same sets as given in P ; $\delta_{P||C} \subseteq S_{P||C} \times A \times S_{P||C}$ is the set of transitions of $P||C$ and is given by,

$$\{((s, q), \sigma, (s', q')) \mid (s, \sigma, s') \in \delta_P \wedge (q, \sigma, q') \in \delta_C\},$$

$L_{P||C} : S_{P||C} \rightarrow 2^{AP}$ is the labeling function for $P||C$, which is defined as $L_{P||C}(s, c) := L_P(s) \cap L_C(c)$, and $S_{0,P||C} = S_{0,P} \times S_{0,C}$ denotes the set of initial states of $P||C$. In the rest of the paper, we will use s, s', s'', \dots to represent states in the plant, q, q', q'', \dots to represent states of supervisor and $(s, a, s') \in \delta_P$ (or $(q, a, q') \in \delta_C$) will be written as $s \xrightarrow{a} s'$ (or $q \xrightarrow{a} q'$).

Due to the presence of uncontrollable actions, the supervisor is required to satisfy the following control compatibility requirement: when controlling the transitions defined at a plant state s , the supervisor must not disable any uncontrollable transitions defined at s .

III. QUOTIENTING MU-CALCULUS SPECIFICATIONS

Given a discrete event plant P , and a mu-calculus specification φ° , the control problem is to determine the existence of a control-compatible supervisor C enforcing the specification exists, and to find one when it exists. We view the control problem as a specification transformation problem, which transforms the “obligation” φ° on the controlled plant

$$\begin{aligned}
1. \quad (\text{tt}/_T s) &= \begin{cases} \nu Z_s. (\bigwedge_{\substack{a \in A_u(s) \\ s \xrightarrow{a} s'}} \langle a \rangle (\text{tt}/_{T \cup \{Z_s\}} s') \wedge \\ \bigwedge_{\substack{a \in A_c(s) \\ s \xrightarrow{a} s'}} [a] (\text{tt}/_{T \cup \{Z_s\}} s')) \\ \text{if } Z_s \notin T \\ Z_s \text{ otherwise} \end{cases} \\
2. \quad (\text{ff}/_T s) &= \text{ff}. \\
3. \quad (p/_T s) &= \begin{cases} (\text{tt}/_T s) \wedge p & \text{if } p \in L_P(s) \\ \text{ff} & \text{otherwise} \end{cases} \\
4. \quad (\varphi_1 \wedge \varphi_2/_T s) &= (\varphi_1/_T s) \wedge (\varphi_2/_T s). \\
5. \quad (\varphi_1 \vee \varphi_2/_T s) &= (\varphi_1/_T s) \vee (\varphi_2/_T s). \\
6. \quad (\langle a \rangle \varphi/_T s) &= (\text{tt}/_T s) \wedge \begin{cases} \langle a \rangle (\bigvee_{s' : s \xrightarrow{a} s'} (\varphi/_T s')) \\ \text{if } \exists s' : s \xrightarrow{a} s' \\ \text{ff} \text{ otherwise} \end{cases} \\
7. \quad ([a] \varphi/_T s) &= (\text{tt}/_T s) \wedge \begin{cases} [a] (\bigwedge_{s' : s \xrightarrow{a} s'} (\varphi/_T s')) \\ \text{if } \exists s' : s \xrightarrow{a} s' \\ \text{tt} \text{ otherwise} \end{cases} \\
8. \quad (\sigma X. \varphi_x/_T s) &= \begin{cases} \sigma X_{(s,k+1)}. (\varphi_x/_T [X_{(s,k)}/X_{(s,k+1)}] s) \\ \text{if } X_{(s,k)} \in T \\ \sigma X_{(s,1)}. (\varphi_x/_T [X_{(s,1)}] s) \text{ otherwise} \end{cases} \\
9. \quad (X/_T s) &= \begin{cases} X_{(s,1)} & \text{if } X \in FV^\circ \\ X_{(s,k)} & \text{otherwise if } X_{(s,k)} \in T \\ (\sigma X. \varphi_x/_T s) & \text{otherwise where } \sigma X. \varphi_x \in Sub^\circ \end{cases}
\end{aligned}$$

Fig. 2. Quotienting Rules

$P||C$ to an obligation φ^\ddagger on the supervisor C . Satisfiability of φ^\ddagger ensures the existence of a supervisor C , while a model for a satisfiable φ^\ddagger represents a desired supervisor. This specification transformation is referred to as *quotienting* and is similar in flavor to that in [1], [3] where it is applied for efficient model checking of synchronous systems with replicated sub-systems, and infinite-state parameterized systems.

We use FV° , Sub° , nd° to denote the set of free variables, the set of sub-formulas, and the nesting depth respectively, of φ° . The quotienting operation involves introducing new variables, one for each element in $\mathcal{X} \times S_P \times \mathbb{N}$ (\mathbb{N} is the set of integers), which for $X \in \mathcal{X}$, $s \in S_P$, and $k \in \mathbb{N}$ is denoted as $X_{(s,k)}$. The set $\mathcal{X} \times S_P \times \mathbb{N}$ itself is denoted as $\mathcal{X}_{(S_P \times \mathbb{N})}$. In order to keep track of control compatibility requirement, quotienting also introduces new greatest fixed point variables of the form Z_s , one for each $s \in S_P$. The set of all such variables is denoted by \mathcal{Z} . We use \mathcal{T} , referred to as “tags”, to denote $2^{\mathcal{X}_{(S_P \times \mathbb{N})} \cup \mathcal{Z}}$. The tag set $T \in \mathcal{T}$ maintains a certain history that is needed for quotienting a fixed-point formula or a fixed-point variable (to be explained in more detail below). For each $s \in S$, and $T \in \mathcal{T}$, we define a quotienting function $/_T(s) : \Phi[AP, \mathcal{X}, A] \rightarrow \Phi[AP, \mathcal{X}_{(S_P, \mathbb{N})} \cup \mathcal{Z}, A]$ (see Fig. 2).

Discussion. The quotienting operation, $(\varphi/_T s)$, generates a formula ψ such that a controlled plant state (s, q) satisfies φ if and only if the supervisor state q satisfies ψ .

Rule 1 in Fig. 2 states that, regardless of the controlled plant state (s, q) (the propositional constant tt is satisfied by every state (s, q)), the supervisor state q should satisfy control compatibility. This is represented using a greatest fixed point formula over Z_s which requires that for all uncontrollable transitions $s \xrightarrow{a} s'$ there exists a transition

$q \xrightarrow{a} q'$ such that q' is itself control compatible with respect to actions in $A_u(s')$. Further the controllable actions in $A_c(s)$ may or may not be permitted (implied by the presence of box-modality which can be satisfied by the presence or absence of box-modal action). In order to be able to terminate the recursive quotienting, the tag set keeps track of whether or not tt has already been quotiented against the state s . If yes, the recursive quotienting terminates with the result equal to the corresponding fixed-point variable Z_s .

Rule 2 states that the controlled plant state (s, q) satisfies ff if and only if the supervisor state q satisfies ff . This is a tautology. Rule 3 states that for the controlled plant state (s, q) to satisfy the atomic proposition p , p must be satisfied by s (otherwise no q state exists) and the supervisor state q must also satisfy p . In addition, the supervisor state should satisfy the control compatibility requirement $((\text{tt}/_T s))$. Rules 4 and 5 states that quotienting commutes with conjunction and disjunction.

Rule 6 corresponds to the modal formula $\langle a \rangle \varphi$. A controlled plant state (s, q) satisfies $\langle a \rangle \varphi$ if and only if there exists a successor state (s', q') on action “ a ” such that (s', q') satisfies φ ; or equivalently, there exists a successor (s', q') such that q' satisfies $(\varphi/_T s')$. Proceeding further, (s, q) satisfies $\langle a \rangle \varphi$ if and only if there exists a -successor s' and q satisfies the formula $\langle a \rangle \bigvee_{s' : s \xrightarrow{a} s'} (\varphi/_T s')$. Note that the quotiented formula includes the extra conjunct $(\text{tt}/_T s)$ to account for the control compatibility requirement. Rule 7 is a dual of Rule 6 and can be understood in a similar way.

Rules 8 and 9 are used for quotienting fixed point formula and fixed-point variable respectively. Due to (i) the multiplicity of the plant states, (ii) the multiplicity of the variables that a certain fixed-point formula embeds, and (iii) the fact that quotienting is performed by recursively descending the parse-tree of the sub-formulas, the quotienting of a fixed-point formula can occur in association with different states and multiple times with each state. The tag set keeps track for each fixed-point formula, and for each state, the number of times the fixed-point formula has been quotiented (with respect to the state). The count is incremented by one each time such a quotienting is performed. We argue that the count remains bounded. As a result the size of tag set itself remains bounded (see Theorem 1).

Rule 8 states that the quotient of a fixed point formula is the fixed point of the quotient formula, where the fixed point variable $X_{(s,j)}$ appearing in the quotiented formula is a function of (a) the variable X of the formula being quotiented, (b) the state s against which the quotient is being performed, and (c) the number of times, j , such quotient has been performed in past. Note that each repeated quotienting operation on the same fixed point formula against the same state increments the count of quotienting and the corresponding variable is stored in the tag set T .

Rule 9 states that the quotient of a fixed point variable is same as the quotient of the corresponding fixed point formula when that variable has not been quotiented in the past as indicated by the tag set; otherwise the result is the corresponding fixed point variable obtained from the tag

set. The latter case causes the termination of the recursive computation. Note that, if the variable being quotiented is a free variable, the quotienting operations generates a new free variable as the result.

Remark 1: The first quotienting rule is to accommodate the state-dependent controllability constraint. It is possible to modify this rule to accommodate any general constraint ψ_s that a controller state must satisfy when the plant is in state s by simply defining Rule 1 as $(\tau\tau/\tau s) = \psi_s$.

We have the following theorems establishing the termination of the quotienting of a fixed-point formula and the correctness of the reduction of the control problem to the satisfiability of the quotiented formula.

Theorem 1: Given $P = (S_P, A, \delta_P, AP, L_P, S_{0,P})$ and a control specification formula φ° , the maximum number of times a fixed point expression $\sigma Y.\varphi_y$, a sub-formula of φ° , is quotiented by any state $s \in S_P$ is $O(|S_P|^{nd^\circ})$.

Proof: For a formula φ with $nd(\varphi) = 1$, the above theorem can be proved immediately.

Assume that for φ with $nd(\varphi) = n$, the maximum number of times a fixed point expression is quotiented by a state is $f(n)$ (induction hypothesis). We add an outer fixed point formula $\sigma X.\varphi_x$ expression such that φ is a sub-formula in φ_x and $nd(\sigma X.\varphi_x) = n + 1$. If $\sigma X.\varphi_x$ is quotiented once, then fixed point expressions in its sub-formula φ with $nd(\varphi) = n$ can be quotiented $f(n)$ times by a state (from induction hypothesis). Since X is the outermost fixed point variable, it can be quotiented $|S_P|$ times. Proceeding further, fixed point expressions in its sub-formula φ can be quotiented $f(n) \times |S_P|$ times by a state, i.e., $f(n+1) = f(n) \times |S_P|$. Therefore, $\forall i \geq 1. f(i) = |S_P|^i$. ■

Theorem 2: Consider a plant $P = (S_P, A, \delta_P, AP, L_P, S_{0,P})$ and a control specification φ° . Then for any supervisor $C = (S_C, A, \delta_C, AP, L_C, S_{0,C})$, a controlled state (s, q) satisfies φ° iff the supervisor state q satisfies $(\varphi^\circ/\circ s)$.

Proof: Follows from the discussion of the quotienting rules and Theorem 1. ■

IV. SATISFIABILITY CHECKING AND MODEL DISCOVERY

Having reduced the problem of control to one of mu-calculus satisfiability, we present a technique for checking the satisfiability of a mu-calculus formula $\varphi^\circ \in \Phi[AP, \mathcal{X}, A]$ and identifying a model witnessing the satisfiability.

Preliminaries. To distinguish between greatest and least fixed point variables at various alternation depths, we assign an identifier, id to each variable:

$$id(X) := \begin{cases} 2 \times ad(\sigma X.\varphi_x) & \text{if } \sigma = \nu \\ 2 \times ad(\sigma X.\varphi_x) - 1 & \text{otherwise} \end{cases}$$

Note that the id of a fixed point variable is an odd number if and only if it is of the least fixed point nature, and is the largest for the outer most fixed point variable.

A. Tableau-based Approach

We use a set of rules to construct a tableau from which satisfiability of a mu-calculus formula can be established and an appropriate model can be identified. Each tableau

rule is of the form: $\frac{C_{\mathcal{H}^1}^1 M^1 \quad C_{\mathcal{H}^2}^2 M^2 \quad \dots \quad C_{\mathcal{H}^n}^n M^n}{C_{\mathcal{H}^0}^0 M^0}$ where each C is a set with elements of the form $(\varphi, \vec{X}, \vec{N})$. Here $\varphi \in \Phi[AP, \mathcal{X}, A]$ is a formula expression, $\vec{X} \in \mathcal{X}^*$ is a sequence of fixed point variables, and $\vec{N} \in \mathbb{N}^*$ is a sequence of integers. The history annotation \mathcal{H} is of the form $\{(C^j, s^j)\}$, where each s^j is a state-symbol.

For $i > 0$, each C^i, \mathcal{H}^i is determined as a function of C^0 and \mathcal{H}^0 , whereas M^0 is defined as a function of $M^1, M^2, \dots, M^n, C^0$ and \mathcal{H}^0 . Further each M^i is a model expression which satisfies the conjunction of the formula expressions in C^i . There are two constant model expressions: $M_{\tau\tau}$ is simply a single state (representing a true-model) while M_{ff} represents a model with no state (a false-model). Additionally M can take the form, $s * [\bigwedge_i (a_i : M_i)]$, meaning M is rooted at state s possessing for each i an a_i -transition to the root of model M_i . In case $M_i = M_{\text{ff}}$ for some i , then $s * [\bigwedge_i (a_i : M_i)]$ is equivalent to M_{ff} .

Each pair " $C_{\mathcal{H}^i}^i M^i$ " is referred to as a node of the tableau. We also say that the pair " $C_{\mathcal{H}^0}^0 M^0$ " is the numerator node of a tableau rule while " $C_{\mathcal{H}^i}^i M^i$ " ($1 \leq i \leq n$) pairs are referred to as the denominator nodes. Fig. 3 presents the tableau rules.

Discussion. Given a formula φ° whose satisfiability needs to be determined, a tableau-tree (or simply a tableau) rooted at the node " $\{(\varphi^\circ, \epsilon, \epsilon)\}_\emptyset M$ " is iteratively created by firing a tableau-rule whose numerator matches a node of the existing tableau-tree, and in which case successor tableau-tree nodes corresponding to the denominator are created. Note no successor is created for a tableau-rule whose denominator is empty (denoted as a " \bullet "). When the iteration creating the tableau-tree terminates the model expressions of the leaf nodes possess definite valuations, which are used to associate definite valuations with the model expressions of all other nodes, including the root. At this point φ° is satisfiable if and only if the model expression M of the root node is not M_{ff} (see Theorem 3).

Rule 1 states that M satisfies a conjunction of formula expressions one of which is $\tau\tau$ if and only if it satisfies the conjunction without the conjunct $\tau\tau$. On the other hand Rule 2 states that M satisfies a conjunction of formula expressions one of which is ff if and only if M equals M_{ff} . Rule 3 states that M satisfies a conjunction of formula expressions one of which is an atomic proposition p if and only if M is rooted at a state s labeled p ($p \in L(s)$) and also M satisfies the conjunction without the conjunct p . The fact that M is rooted at s has been represented as $M = s * B$, where B is of the form $\bigwedge_i (a_i : M_i)$.

Rule 4 states that the true-model (or equivalently any model) satisfies a conjunction of *empty* set of formula expressions. Rule 5 states that M satisfies a conjunctive formula if and only if it satisfies each of the conjuncts, whereas Rules 6 and 7 state that M satisfies a disjunctive formula if and only if it satisfies one of the disjuncts. Rule 8 states that M satisfies a fixed point formula $\sigma X.\varphi$ if and only if it satisfies φ .

Rule 9 states that M satisfies a conjunction of formula

$$\begin{array}{l}
1. \frac{\{(\text{tt}, \vec{X}, \vec{N})\} \cup C\}_{\mathcal{H}} M}{C_{\mathcal{H}} M} \quad 2. \frac{\{(\text{ff}, \vec{X}, \vec{N})\} \cup C\}_{\mathcal{H}} M = M_{\text{ff}}}{\bullet} \\
3. \frac{\{(p, \vec{X}, \vec{N})\} \cup C\}_{\mathcal{H}} M = s * B}{C_{\mathcal{H}} M = s * B} \text{ where } p \in L(s) \\
4. \frac{\{\gamma\}_{\mathcal{H}} M = M_{\text{tt}}}{\bullet} \quad 5. \frac{\{(\varphi \wedge \psi, \vec{X}, \vec{N})\} \cup C\}_{\mathcal{H}} M}{\{(\varphi, \vec{X}, \vec{N}), (\psi, \vec{X}, \vec{N})\} \cup C\}_{\mathcal{H}} M} \\
6. \frac{\{(\varphi \vee \psi, \vec{X}, \vec{N})\} \cup C\}_{\mathcal{H}} M}{\{(\varphi, \vec{X}, \vec{N})\} \cup C\}_{\mathcal{H}} M} \quad 7. \frac{\{(\varphi \vee \psi, \vec{X}, \vec{N})\} \cup C\}_{\mathcal{H}} M}{\{(\psi, \vec{X}, \vec{N})\} \cup C\}_{\mathcal{H}} M} \\
8. \frac{\{(\sigma X. \varphi, \vec{X}, \vec{N})\} \cup C\}_{\mathcal{H}} M}{\{\varphi, \vec{X}, \vec{N}\} \cup C\}_{\mathcal{H}} M} \\
9. \frac{\{(X, \vec{X}, \vec{N})\} \cup C\}_{\mathcal{H}} M}{C_{\mathcal{H}} M} \text{ where } X \in FV^{\circ} \\
10. \frac{\{(X, \vec{X}, \vec{N})\} \cup C\}_{\mathcal{H}} M}{\{(\sigma X. \varphi, (X. \vec{X}), \vec{N})\} \cup C\}_{\mathcal{H}} M} \quad \sigma X. \varphi \in \text{Sub}^{\circ} \\
11. \frac{\{([a]\varphi, \vec{X}, \vec{N}) \cup C\}_{\mathcal{H}} M}{\{([\varphi \wedge (a)\text{tt}], \vec{X}, \vec{N}) \cup C\}_{\mathcal{H}} M}} \quad 12. \frac{\{([a]\varphi, \vec{X}, \vec{N}) \cup C\}_{\mathcal{H}} M}{C_{\mathcal{H}} M} \\
\text{if } C = \{(\langle a_i \rangle \varphi_i, \vec{X}_i, \vec{N}_i)\}, \\
\mathcal{A}(C' = \{(\langle a_i \rangle \varphi_i, \vec{X}'_i, \vec{N}'_i)\}, s) \in \mathcal{H} \text{ and} \\
\mathcal{A}(\langle a \rangle \varphi_j, \vec{X}_j, \vec{N}_j) \in C \\
13. \frac{C_{\mathcal{H}} M}{C_{\mathcal{H}'}^{a,1} M^{a,1} \dots C_{\mathcal{H}'}^{a,n} M^{a,n}} \\
\text{if } C = \{(\langle a_i \rangle \varphi_i, \vec{X}_i, \vec{N}_i)\}, \mathcal{A}(C' = \{(\langle a_i \rangle \varphi_i, \vec{X}'_i, \vec{N}'_i)\}, s) \in \mathcal{H} \\
\text{and } \forall (\langle a \rangle \varphi_i, \vec{X}_i, \vec{N}_i) \in C. \exists (\langle a \rangle \varphi_j, \vec{X}_j, \vec{N}_j) \in C, \text{ in which case,} \\
M := s_C * \bigwedge_{a,j} a : M^{a,j} \\
C^{a,j} := \{(\varphi_i, \vec{X}_i, i. \vec{N}_i) \mid (\langle a \rangle \varphi_i, \vec{X}_i, \vec{N}_i) \in C\} \cup \{(\varphi_j, \vec{X}_j, j. \vec{N}_j)\}, \\
\text{such that } (\langle a \rangle \varphi_j, \vec{X}_j, \vec{N}_j) \in C \\
\mathcal{H}' := \mathcal{H} \cup \{(C^{\text{D}}, s_C)\}, \text{ where} \\
C^{\text{D}} := \{(\langle a \rangle \varphi_i, \vec{X}_i, i. \vec{N}_i) \mid (\langle a \rangle \varphi_i, \vec{X}_i, \vec{N}_i) \in C\} \\
14. \frac{C_{\mathcal{H}} M}{\bullet} \\
\text{if } C = \{(\langle a_i \rangle \varphi_i, \vec{X}_i, \vec{N}_i)\}, \exists (C' = \{(\langle a_i \rangle \varphi_i, \vec{X}'_i, \vec{N}'_i)\}, s) \in \mathcal{H} \\
\text{in which case,} \\
M := \begin{cases} M_{\text{ff}} & \text{if } \text{lfP}(C, C') \\ s & \text{otherwise} \end{cases}, \text{ where} \\
\text{lfP}(C, C') \text{ is a Boolean expression that holds iff} \\
\exists i_0, i_1, \dots, i_n = i_0 : (\forall j \in [0, n-1] : \vec{N}'_{i_j} \in \text{succ}(\vec{N}_{i_{j+1}}), \\
\max\{id(X) \mid X \in \vec{X}_{i_{j+1}} / \vec{X}'_{i_j}, j \in [0, n-1]\} \text{ is odd}
\end{array}$$

Fig. 3. Tableau for satisfiability and model identification for \wp

expressions one of which is a free variable $X \in FV^{\circ}$ if and only if it satisfies the conjunction without the conjunct X . (This is so because the satisfiability of a free variable can always be guaranteed by selecting an appropriate environment.) Rule 10 is similar but applies to a conjunction of formula expressions one of which is a fixed-point variable. In this case M satisfies the conjunction if and only if it satisfies the conjunction with the fixed-point variable conjunct replaced by its fixed point formula expression. At the same time the fixed-point variable X is prepended to the sequence \vec{X} of the corresponding tuple. This is to keep track of the sequence of fixed-point variables visited thus far.

Rules 11, 12, 13 and 14 apply when all the formula expressions in the numerator C are modal formula expressions. Rules 11 and 12 are applied when there exists a box-modal

formula on an action a with no diamond-modal formula on the same action. Together these rules state that $[a]\varphi$ can be satisfied if and only if either some a -transitions exist and all its successors satisfy φ (Rule 11) or there is no a -transition (Rule 12).

Rule 13 is applied where every box-modal action has a corresponding diamond modal obligation. It states that for any action a , a set of formula expression of the form $\{([a]\varphi)\}$ is satisfiable by a model state if and only if each diamond obligation is satisfied by some a -successor and each a -successor satisfies all of the box obligations. Accordingly for each action a , a denominator node aggregates all the box-obligations and one diamond obligation (see definition of $C^{a,j}$ in Rule 13 of Fig. 3). Finally the history is augmented to record (i) C , modified to include the ancestor node tag by prepending \vec{N}_i with i , and (ii) the model state s_C that satisfies the formula expressions in C . Such augmentation is performed to record the fact that C was visited. Note that a C containing only the modal formula expressions is recorded in the history set. This is because only for such a C , a transition in the model state occurs (on the associated modal actions).

Rule 14 applies when the modal formula expressions in the numerator C are also present in an element C' of the history set, implying that such formula expressions are being revisited owing to the expansion of certain fixed point variables (Rule 10). The set of fixed point variables expanded is given by $\vec{X}_{i_{j+1}} / \vec{X}'_{i_j}$, $j \in [0, n-1]$, where the notation “ \vec{X}_1 / \vec{X}_2 ” removes the suffix \vec{X}_2 from the sequence \vec{X}_1 . The predicate $\text{lfP}(C, C')$ holds if and only if the outermost fixed point variable (one having the largest id) expanded is of the least fixed point nature (i.e. its id is odd). If lfP evaluates to true, then the model M is set to M_{ff} ; otherwise it is set equal to the state s corresponding to the element C' in \mathcal{H} .

Theorem 3: A mu-calculus formula φ° is satisfiable iff there exists a tableau with root node “ $\{(\varphi^{\circ}, \epsilon, \epsilon)\}_{\emptyset} M$ ”, such that M is assigned to a non false-model.

Proof: Follows from the above discussion. \blacksquare

B. Complexity

We consider a nondeterministic plant with state set S_P , maximum branching degree d^a on any action a ($d^a=1$ for a deterministic P), maximum branching degree d over all actions, and a control specification φ° .

The length of the quotiented formula φ^{\dagger} can be estimated by first estimating its nesting depth and next estimating the number of boolean and modal operators appearing at each level of the nesting. The nesting depth of the quotiented formula is $O(|S_P|^{n_{d^{\circ}}})$ (from Theorem 1). Now to estimate the number of boolean and modal operators at any level of the nesting, we consider the “amplification factor” due to each quotienting rule (with respect to the existing number of boolean and modal operators in φ° , which is $|\varphi^{\circ}|$), and aggregate them to get the overall amplification. All but Rules 1, 6, and 7 have the unity amplification factor. The amplification factor of Rule 1 is $O(|S_P| \times d)$ since the number of boolean operators in each greatest fixed point formula is $O(d)$ and the number of greatest fixed point

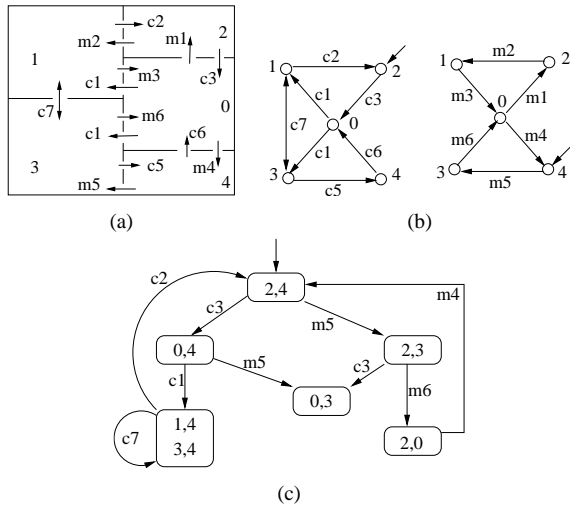


Fig. 4. (a) Maze for cat and mouse. (b) Models for cat and mouse. (c) Supervisor: Each state denotes cat and mouse position.

formula introduced at a nesting level is $O(|S_P|)$. Rules 6 and 7 have the amplification factor of $O(\max_a d^a) \leq O(d)$. So the overall amplification factor is $O(1 + (|S_P| \times d) + d)$. Multiplying this by the number of boolean and modal operators in φ° , i.e., $|\varphi^\circ|$, yields the second estimate as $O([1 + (|S_P| + 1) \times d] \times |\varphi^\circ|)$. So the length of the quotiented formula is $O([1 + (|S_P| + 1) \times d] \times |\varphi^\circ| \times |S_P|^{nd^\circ})$.

Note that when the controllability constraint is state-independent, Rule 1 can be simplified as:

$$(\text{tt} \not\sim_T s) = \begin{cases} \nu Z. (\bigwedge_{a \in A_u} (a)Z \wedge \bigwedge_{b \in A_c} [b]Z) & \text{if } s \in S_{0,P} \\ \text{tt} & \text{otherwise} \end{cases}$$

In this case, the length of the quotiented formula φ^\ddagger becomes $O([(1 + d) \times |\varphi^\circ| + |A|] \times |S_P|^{nd^\circ})$. A similar simplification is applicable for any other state-independent controllability constraint.

We next consider the complexity of satisfiability checking and model identification for the quotiented formula φ^\ddagger , which considers at each of its nesting level, all possible subsets of the subformulae of φ^\ddagger . At each nesting level the number of possible subsets of the subformulae φ^\ddagger examined is $O(2^{[1 + (|S_P| + 1) \times d] \times |\varphi^\circ|})$. So the overall complexity is given by $O(|S_P|^{nd^\circ} \times 2^{[1 + (|S_P| + 1) \times d] \times |\varphi^\circ|})$. In light of the discussion of the previous paragraph, the complexity simplifies to $O(|S_P|^{nd^\circ} \times 2^{(1+d) \times |\varphi^\circ| + |A|})$ when the controllability is state-independent. Note that this is polynomial in the number of plant states S_P .

V. CASE STUDY

Consider the maze of 5 rooms shown in Fig. 4(a) where a cat and a mouse move. The rooms are labeled 0 through 4 and are interconnected via a number of doorways. Each doorway is assigned a direction and has an exclusive accessibility either for the mouse or the cat: doorway accessible by the mouse is denoted by m_i while that accessible by the cat is denoted c_i . All doorways with the exception of c_7 are controllable. Initially the cat and the mouse are in the rooms 2 and 4 respectively. The behaviors of the cat and

the mouse are show in Fig. 4(b); their product generates the plant model.

The objective is to find a supervisor which does not allow the cat and the mouse to occupy the same room simultaneously. The specification of the controlled plant is represented by greatest fixed point formula $\nu X. (p \wedge [-]X)$ where p represents a proposition which is true only when the cat and the mouse are *not* in the same room. We use a short-hand notation $[-]$ to represent *any* action. The formula represents the states where p holds and this continues to remain true after any action.

The specification is quotiented against the plant model and a model (representing a candidate controller) for the quotient is generated. The supervisor model is presented in Fig. 4(c). In the figure, we denote states in the supervisor with the corresponding rooms in which the cat and the mouse are present. Note that, when the cat and the mouse are in rooms 0 and 4 respectively, the supervisor can allow the cat-move c_1 . As this is a non-deterministic transition for the cat, the successor controller state designates that the cat can be either in room 1 or 3. Prototype implementation of our technique is available at <http://www.cs.iastate.edu/~sbasu/control-quot>.

VI. CONCLUSION

We presented a technique for supervisory control of non-deterministic discrete event plants under complete observation of events subject to specification expressed in the propositional mu-calculus. Central to our method is a direct-quotienting of the mu-calculus specification against the plant model. A control-compatible supervisor exists if and only if the quotiented formula is satisfiable, and further a model witnessing the satisfiability can be used as a supervisor.

REFERENCES

- [1] H.R. Andersen. Partial model checking (extended abstract). In *Logic in Computer Science*, 1995.
- [2] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, pages 7–34, 2003.
- [3] Samik Basu and C. R. Ramakrishnan. Compositional analysis for verification of parameterized systems. *Theoretical Computer Science*, 354(2):211–229, 2006.
- [4] E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. *Siam Journal of Computing*, 29(1):132–158, 1999.
- [5] E. A. Emerson, C. S. Jutla, and A. P. Sistla. On model checking for the μ -calculus and its fragments. *Theoretical Computer Science*, 258(1-2):491–522, 2001.
- [6] D. Janin and I. Walukiewicz. Automata for the modal μ -calculus and related results. In *Proceedings of International Symposium on Mathematical Foundations of Computer Science*, 1995.
- [7] D. Kozen. Results on the propositional μ -calculus. *Theoretical Computer Science*, 1983.
- [8] Sophie Pinchinat and Stéphane Riedweg. A decidable class of problems for control under partial observation. *Information Processing Letters*, 95(4):454–465, August 2005.
- [9] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987.
- [10] S. Riedweg and S. Pinchinat. Quantified mu-calculus for control synthesis. In *Mathematical Foundations of Computer Science*, Bratislava, Slovak Republic, aug 2003.
- [11] Colin Stirling. Games and modal mu-calculus. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 298–312, 1996.