Quantified Mu-Calculus for Control Synthesis

Stéphane Riedweg and Sophie Pinchinat

IRISA-INRIA, F-35042, Rennes, France
{sriedweg,pinchina}@irisa.fr
Fax: +33299847171

Abstract. We consider an extension of the mu-calculus as a general framework to describe and synthesize controllers. This extension is obtained by quantifying atomic propositions, we call the resulting logic *quantified mu-calculus*. We study its main theoretical properties and show its adequacy to control applications. The proposed framework is expressive : it offers a uniform way to describe as varied parameters as the kind of systems (closed or open), the control objective, the type of interaction between the controller and the system, the optimality criteria (fairness, maximally permissive), etc. To our knowledge, none of the former approaches can capture such a wide range of concepts.

1 Introduction

To generalize the control synthesis theory of Ramadge and Wohnam [1], lot of works use temporal logics as specification [2–4]. All those approaches suffer from substantial limitations: there is no way to impose properties on the interaction between the system and its controller, nor to require optimality of controllers. The motivation of our work is to fill these gaps. We put forward an extension of the mu-calculus well suited to describe general control objectives and to synthesize finite state controllers. The proposed framework is expressive : it offers a uniform way to describe as varied parameters as the kind of systems (closed or open), the control objective, the type of interaction between the controller and the system, the optimality criteria (fairness, maximally permissive), etc. To our knowledge, none of the former approaches can capture such a wide range of concepts.

As in [5–7], we extend a temporal logic (the mu-calculus) by quantifying atomic propositions. We call the resulting logic *quantified mu-calculus*. We study its main theoretical properties and show its adequacy to control applications. We start from alternating tree automata for mu-calculus [8, 9] and we extend their theory using the Simulation Theorem [10, 11, 8] and a projection of automata. The Simulation Theorem states that alternating automata and nondeterministic automata are equivalent. The projection is an adaption of the construction of [12]. The meanings of existential quantifier is defined projecting automata on sets of propositions. Decision procedures for model-checking and satisfaction can therefore be obtained. Both problems are non-elementary when we consider the full logic. We can however display interesting fragments with lower complexity, covering still a wide class of control problems.

B. Rovan and P. Vojtáš (Eds.): MFCS 2003, LNCS 2747, pp. 642–651, 2003.

The following explains the applications to control. We view supervision of systems as pruning the systems' computation trees. Consequently, a controller can be represented by a labeling c of the (uncontrolled) system's computation tree into $\{0,1\}$, such that the (downwards closed) 1-labeled subtree is the behavior of the controlled system. For any proposition c, we define a transformation $\alpha * c$ of mu-calculus formulas α such that some controller induced restriction of $\mathcal{S}$ satisfies α if and only if $\alpha * c$ holds of some c-labeling on the computation tree. Labeling allows to consider the forbidden part of the controlled system, and we derive controllers for large classes of specifications, using a constructive model-checking.

Beyond the capability to specify controllers which only cut controllable transitions, we can more interestingly specify (and synthesize) a *maximally permissive controller* for α; i.e. a controller c such that the c-controlled system satisfies α and no c'-controlled system such that $c \subsetneq c'$ satisfies α; where $c \subsetneq c'$ is the mu-calculus formula expressing that the 1-labeled subtree defined by c is a proper subtree of the 1-labeled subtree defined by c'. A maximally permissive controller enforcing α can therefore be specified by the quantified mu-calculus formula:

$$\exists c.\, \Big[\alpha * c \,\wedge\, \forall c'.\, \Big(c \subsetneq c' \;\Rightarrow\; \neg(\alpha * c')\Big)\Big].$$

Controllers and maximally permissive controllers for *open systems* [2] may also be specified and synthesized. Such controllers are required moreover to be robust against the environment's policy. Also, the implementation considerations of [13] and decentralized controllers may be formulated in quantified mu-calculus. Not surprisingly, the expressive power of the mu-calculus enables us to deal with fairness.

The rest of the paper is organized as follows : Section 2 presents the logic. Section 3 studies applications to control theory. Algorithms are developed in Section 4, based on the automata-theoretic semantics. Finally, control synthesis is illustrated in Section 5.

2 Quantified Mu-Calculus

We assume given a finite set of events A, a finite set of propositions AP, and an infinite set of variables $Var = \{X, Y, \ldots\}$.

Definition 1. (Syntax of $\mathrm{Q}L_\mu$) *The set of formulas of the quantified mu-calculus on $\Gamma \subseteq AP$, written $\mathrm{Q}L_\mu(\Gamma)$, is defined by the grammar:*

$$\exists \Lambda.\alpha \,|\, \neg\alpha_1 \,|\alpha_1 \vee \alpha_2 \,|\, \beta$$

where $\Lambda \subseteq AP$, $\alpha \in \mathrm{Q}L_\mu(\Gamma \cup \Lambda)$, α_1 and α_2 are formulas in $\mathrm{Q}L_\mu(\Gamma)$, and β is a formula of the pure mu-calculus on Γ. The set of formulas of the pure mu-calculus on $\Gamma \subseteq AP$, written $L_\mu(\Gamma)$, is defined by the grammar:

$$\top \,|\, p \,|\, X \,|\, \neg\beta \,|\; <a>\, \beta \,|\, \beta \vee \beta' \,|\, \mu X.\beta(X)$$

where $a \in A$, $p \in \Gamma$, $X \in Var$, and β and β' are in $L_\mu(\Gamma)$. To ensure meanings to fix-points formulas, X must occur under an even number of negation symbols $\neg$ in $\alpha(X)$, in each formula $\mu X.\alpha(X)$.
Extending the terminology of mu-calculus, we call sentences *all quantified mu-calculus formulas without free variables. We write $\bot$, $[a]\alpha$, $\alpha \wedge \beta$, $\nu X.\alpha(X)$, and $\forall \Lambda.\alpha$ respectively for negating $\top$, $<a>\neg\alpha$, $\neg\alpha \vee \neg\beta$, $\mu X.\neg\alpha(\neg X)$ and $\exists \Lambda.\neg\alpha$. We write also $\stackrel{a}{\rightarrow}$, $\stackrel{a}{\not\rightarrow}$, $\alpha \Rightarrow \beta$, and $\exists x.\alpha$ respectively for $<a>\top$, $[a]\bot$, $\neg\alpha \vee \beta$, and $\exists\{x\}.\alpha$.*

Since in general, fixed-point operators and quantifiers do not commute, we enforce no quantification inside fixed-point terms. The quantified mu-calculus $\mathrm{Q}L_\mu$, as a generalization of the mu-calculus, is also given an interpretation over deterministic transition structures called *processes* in [3] .

Definition 2. *A* process on $\Gamma \subseteq AP$ *is a tuple $\mathcal{S} =< \Gamma, S, s^0, t, L >$, where S is the set of states, $s^0 \in S$ is the initial state, $t : S \times A \to S$ is a partial function called the* transition function *and $L : S \to \mathcal{P}(\Gamma)$ maps states to subset of propositions. We say that $\mathcal{S}$ is* finite *if S is finite and that it is* complete *if for all $(a, s) \in A \times S$, $t(s, a)$ is defined.*

Compound processes can be built up by synchronous product.

Definition 3. *The* (synchronous) product *of two processes $\mathcal{S}_1 =< \Gamma_1, S_1, s_1^0, t_1, L_1 >$ and $\mathcal{S}_2 =< \Gamma_2, S_2, s_2^0, t_2, L_2 >$ on disjoint sets Γ_1 and Γ_2 is the process $\mathcal{S}_1 \times \mathcal{S}_2 =< \Gamma, S_1 \times S_2, (s_1^0, s_2^0), t, L >$ on $\Gamma = \Gamma_1 \cup \Gamma_2$ such that (1) $(s_1', s_2') \in t((s_1, s_2), a)$ whenever $s_1' \in t_1(s_1, a)$ and $s_2' \in t_2(s_2, a))$, and (2) $L(s_1, s_2) = L_1(s_1) \cup L_2(s_2)$.*

In the sequel, we shall in particular make the product of a process on Γ with another (complete) process on a disjoint set of propositions Λ in order to obtain a similar process on $\Gamma \cup \Lambda$. This is the way in which $\mathrm{Q}L_\mu$ will be applied to solve control problem (see Theorem 1 Section 3).

Definition 4. *A* labeling process on $\Lambda \subseteq AP$ *is simply a complete process $\mathcal{E}$ on Λ. Now, for any process $\mathcal{S} =< \Gamma, S, s^0, t, L >$ with Γ disjoint from Λ, $\mathcal{S} \times \mathcal{E}$ is called a* labeling of $\mathcal{S}$ (by $\mathcal{E}$) on Λ. *We let Lab_Λ denote the set of labeling processes on Λ.*

Definition 5. (Semantics of $\mathrm{Q}L_\mu$) *The interpretation of the $\mathrm{Q}L_\mu(\Gamma)$-formulas is relative to a process $\mathcal{S} =< \Gamma, S, s^0, t, L >$ and a valuation $val : Var \to \mathcal{P}(S)$. This interpretation $[\![\alpha]\!]_{\mathcal{S}}^{[val]} (\subseteq S)$ is defined by:*
$[\![\top]\!]_{\mathcal{S}}^{[val]} = S, \quad [\![p]\!]_{\mathcal{S}}^{[val]} = \{s \in S \,|\, p \in L(s)\}, \quad [\![X]\!]_{\mathcal{S}}^{[val]} = val(X),$
$[\![\neg\alpha]\!]_{\mathcal{S}}^{[val]} = S \setminus [\![\alpha]\!]_{\mathcal{S}}^{[val]}, \quad [\![\alpha \vee \beta]\!]_{\mathcal{S}}^{[val]} = [\![\alpha]\!]_{\mathcal{S}}^{[val]} \cup [\![\beta]\!]_{\mathcal{S}}^{[val]}$
$[\![<a> \alpha]\!]_{\mathcal{S}}^{[val]} = \{s \in S \,|\, \exists s' : t(s, a) = s' \, and \, s' \in [\![\alpha]\!]_{\mathcal{S}}^{[val]}\},$
$[\![\mu X.\alpha(X)]\!]_{\mathcal{S}}^{[val]} = \cap\{V \subseteq S \,|\, [\![\alpha]\!]_{\mathcal{S}}^{[val(V/X)]} \subseteq V\},$
$[\![\exists \Lambda.\alpha]\!]_{\mathcal{S}}^{[val]} = \{s \in S \,|\, \exists \mathcal{E} =< \Lambda, E, \varepsilon^0, t', L' >\in Lab_\Lambda, (s, \varepsilon^0) \in [\![\alpha]\!]_{\mathcal{S} \times \mathcal{E}}^{[val \times E]}\}$
where $(val \times E)(X) = val(X) \times E$ for any $X \in Var$.

Notice that the valuation *val* does not influence the semantics of a sentence $\alpha \in \mathrm{Q}L_\mu$; and we write $\mathcal{S} \models \alpha$ whenever the initial state of $\mathcal{S}$ belongs to $[\![\alpha]\!]_{\mathcal{S}}$.

Clearly, bisimilar processes satisfy the same $\mathrm{Q}L_\mu$ formulas.

3 Control Specifications

This section presents various examples of specifications for control objectives in $\mathrm{Q}L_\mu$. First, a transformation of formulas is defined, which used to link $\mathrm{Q}L_\mu$ model-checking with control problems, as shown by Theorem 1. Variants of the Theorem are then exploited to capture requirements, such as *maximal permissive controllers*, *controllers for open systems*, etc.

Definition 6. *For any sentence $\alpha \in \mathrm{Q}L_\mu(\Gamma)$ and for any $x \in AP$, the x-*lift of *α is the formula $\alpha * x \in \mathrm{Q}L_\mu(\Gamma \cup \{x\})$, inductively defined by (by convention, $*$ has highest priority) :*

$$\top * x = \top, \quad p * x = p, \quad X * x = X, \quad (\neg\alpha) * x = \neg\alpha * x,$$
$$(\alpha \vee \beta) * x = \alpha * x \vee \beta * x, \quad (<a>\ \alpha) * x = <a>\ (x \wedge \alpha * x),$$
$$(\mu X.\alpha) * x = \mu X.\alpha * x, \quad (\exists \Lambda.\alpha) * x = \exists \Lambda.\alpha * x.$$

Definition 7. *Given a process $\mathcal{S} =< \Gamma, S, s^0, t, L >$ and some $x \in \Gamma$, the x-*pruning of *$\mathcal{S}$ is the process $\mathcal{S}_{(\to^x)} =< \Gamma, S, s^0, t', L >$ such that, for all $s \in S$ and $a \in A$, $t'(s,a) = t(s,a)$ if $x \in L(t(s,a))$ or $t'(s,a)$ is undefined otherwise.*

Lemma 1. *For any process $\mathcal{S}$ on Γ, for any $x \in \Gamma$, and for any sentence $\alpha \in \mathrm{Q}L_\mu(\Gamma)$, we have: $[\![\alpha]\!]_{\mathcal{S}_{(\to^x)}} = [\![\alpha * x]\!]_{\mathcal{S}}$.*

Proof. Straightforward by induction on α. □

Control synthesis [1, 3, 14, 4] is a generic problem that consists to enforce a plant $\mathcal{S}$ to have some property α by composing it with another process $\mathcal{R}$, the *controller of $\mathcal{S}$ for α*. The goal is to synthesize $\mathcal{R}$ given α. We focus here on the joint specification α and constraints on $\mathcal{R}$: they reflect the way it exerts the control. This capability relies on the following theorem.

Theorem 1. *Given a sentence $\alpha \in \mathrm{Q}L_\mu(\Lambda \cup \Gamma)$, where Λ and Γ are disjoint, and a process $\mathcal{S}$ on Γ, the following assertions are equivalent :*

- *There exists a controller on Λ of $\mathcal{S}$ for α.*
- *$\mathcal{S} \models \exists c \exists \Lambda.\ \alpha * c$ where c is a fresh proposition.*

Proof. First, suppose that there exists a process $\mathcal{R} =< \Lambda, R, r^0, t, L >$ such that $\mathcal{S} \times \mathcal{R} \models \alpha$. Given $c \in AP \setminus \Lambda \cup \Gamma$, we can easily define $\mathcal{E} \in Lab_{\Lambda \cup \{c\}}$ s.t. $\mathcal{R}$ is $(\mathcal{E}))_{(\to^c)}$ without the label c. Now, $(\mathcal{S} \times \mathcal{E})_{(\to^c)}$ or equivalently $\mathcal{S} \times (\mathcal{E})_{(\to^c)}$ satisfies α, since $\mathcal{R}$ is $(\mathcal{E})_{(\to^c)}$ without c and c does not occur in α. Using Lemma 1, we conclude that $(\mathcal{S} \times \mathcal{E}) \models \alpha * c$. Suppose now that $\mathcal{S} \models \exists c \exists \Lambda.\alpha * c$. By Definition 5, there is a process $\mathcal{E} \in Lab_{\{c\} \cup \Lambda}$ such that $\mathcal{S} \times \mathcal{E} \models \alpha * c$. By Lemma 1, $(\mathcal{S} \times \mathcal{E})_{(\to^c)}$ satisfies α. Then take $\mathcal{R}$ as $(\mathcal{E})_{(\to^c)}$. □

We illustrate now the use of $\mathrm{Q}L_\mu$ to specify various control requirements. The formula $\exists c.\alpha * c$ of Theorem 1 is enriched to integrate control rules. In the sequel, we let $<A>= \bigvee_{a\in A} <a>$, $[A] = \bigwedge_{a\in A}[a]$, $Reach_c(\gamma) = (\mu Y.\ <A> Y \vee \gamma) * c$, and $Inv_c(\gamma) = (\nu Y.[A]Y \wedge \gamma) * c$. Also, the x-lift is canonically extended to conjunctions of propositions.

Maximally Permissive Admissible Controller for α. When a system $\mathcal{S}$ has uncontrollable transitions, denoted by the set of labels $A_{\mathtt{uc}}$, an *admissible controller* for α should not disable any of them. Its existence may be expressed by the formula (1). An admissible controller c for α is *maximally permissive* if no other admissible controller c' for α can cut strictly less transitions than c. Writing $c \subsetneq c'$ the mu-calculus formula $Inv_c(c') \wedge Reach_{c'}(\neg c)$; this requirement is expressed by the formula (2).

$$\mathcal{S} \models \exists c.\ Inv_c([A_{\mathtt{uc}}]c) \wedge \alpha * c \tag{1}$$

$$\mathcal{S} \models \exists c. Inv_c([A_{\mathtt{uc}}]c) \wedge \alpha * c \wedge \Big(\forall c'.\big(Inv_{c'}([A_{\mathtt{uc}}]c') \wedge (c \subsetneq c')\big) \ \Rightarrow\ \neg\alpha * c'\Big). \tag{2}$$

Maximally Permissive Open Controller for α. As studied in [2], an open system $\mathcal{S}$ takes the environment's policy into account : the alphabet A of transitions is a disjoint union of the alphabet $A_{\mathtt{co}}$ of controllable transitions and the alphabet $A_{\mathtt{uc}}$ of uncontrollable transitions, permitted or not by the environment. The *open controller* must ensure α for any possible choice of the environment. This requirement is expressed by the formula (3), where the proposition e represents the environment's policy. The ad-hoc solution of [2] cannot be easily extended to *maximally permissive open controller*. This requirement is expressed by the formula (4).

$$\mathcal{S} \models \exists c. Inv_c([A_{\mathtt{uc}}]c) \wedge \forall e.\big((Inv_e([A_{\mathtt{co}}]e)) \Rightarrow \alpha * (e \wedge c)\big). \tag{3}$$

$$\begin{aligned}\mathcal{S} \models \exists c. Inv_c([A_{\mathtt{uc}}]c) \wedge \big[&\forall e.\big((Inv_e([A_{\mathtt{co}}]e)) \Rightarrow \alpha * (e \wedge c)\big) \\ &\wedge \forall c'.\big((Inv_{c'}([A_{\mathtt{uc}}]c') \wedge (cut(c) \subsetneq cut(c'))) \Rightarrow \\ &\exists e'. Inv_e([A_{\mathtt{co}}]e) \wedge \neg\alpha * (e' \wedge c')\big)\big].\end{aligned} \tag{4}$$

Implementable Controller for "Non-blocking". Such a controller [13], is an admissible controller which, moreover, selects exactly one controllable transition at a time, and such that, in the resulting supervised system, a final state (given by the proposition P_f) is always reachable. Let $\mathtt{Nblock} = \nu Z.\big((\mu X.P_f \vee <A> X) \wedge [A]Z\big)$ and let $\mathtt{Impl} = \big((\bigvee_{a\in A_{\mathtt{co}}} \xrightarrow{a}) \Rightarrow \bigvee_{a\in A_{\mathtt{co}}}(< a > \ c \wedge [A_{\mathtt{co}} \setminus \{a\}]\neg c)\big)$; a non-blocking implementable controller of a system $\mathcal{S}$ may be expressed by the formula: $\mathcal{S} \models \exists c.\ c \wedge (\mathtt{Nblock}) * c \wedge Inv_c([A_{\mathtt{uc}}]c) \wedge Inv_c(\mathtt{Impl})$

Decentralized controllers for α. The existence of decentralized controllers $\mathcal{R}_1$ and $\mathcal{R}_2$ such that $\mathcal{S} \times \mathcal{R}_1 \times \mathcal{R}_2 \models \alpha$ may be expressed : $\mathcal{S} \models \exists c_1 \exists c_2.\alpha * (c_1 \wedge c_2)$.

4 Quantified Mu-Calculus and Automata

Automata-theoretic approaches provide the model theory of mu-calculus, and they offer decision algorithms for the satisfiability and the model-checking problems [15, 10, 16–18, 8]. Depending on the approach followed, different automata

have been considered, differing mainly in two orthogonal parameters : the more or less restricted kind of transitions, ranging from alternating automata to the subclass of non-deterministic automata, and the acceptance conditions e.g. Rabin, Streett, Motkowski/parity.

The class of tree automata for mu-calculus which we shall adapt to $\mathrm{Q}L_\mu$ is the class of *alternating parity automata*, or shortly *simple automata*, considered in [3]. This adaptation is stated by Theorem 2 below which constitutes the main result of this section; the remaining brings back the material needed for its proof.

Theorem 2. (Main result) *For any sentence $\alpha \in \mathrm{Q}L_\mu(\Gamma)$, there exists a simple automaton A_α on Γ such that, for any process $\mathcal{S}$ on Γ :*
$\mathcal{S} \models \alpha$ iff $\mathcal{S}$ is accepted by $\mathcal{A}_\alpha$

Definition 8. (Simple Automata on Processes) *A* simple automaton *on Γ is a tuple $\mathcal{A} =< \Gamma, Q, Q^\exists, Q^\forall, q^0, \delta : Q \times \mathcal{P}(\Gamma) \to \mathcal{P}(Moves(Q)), r >$ where Q is a finite set of states, partitioned into two subsets $Q^\exists$ and $Q^\forall$ of respectively* existential *and* universal *states, $q^0 \in Q$ is the initial state, $r : Q \to I\!N$ is the* parity condition*, and* the transition function *δ assigns to each state q and to each subset of Γ a set of possible* moves *included in $Moves(Q) = ((A \cup \{\epsilon\}) \times Q) \cup (A \times \{\to, \not\to\})$.*

Definition 9. (Nondeterministic Automata on Processes)
A simple automaton is nondeterministic *if for any set of labels $\Lambda \subseteq \Gamma$, $\delta(q, \Lambda) \subseteq \{\epsilon\} \times Q$ for any $q \in Q^\exists$, and $\delta(q, \Lambda) \subseteq Moves(Q) \setminus \{\epsilon\} \times Q$ for any $q \in Q^\forall$. Moreover, in case when $q \in Q^\forall$, it is required that $(a_1, q_1), (a_2, q_2) \in \delta(q, \Lambda) \cap (A \times Q)$ and $a_1 = a_2$ entail $q_1 = q_2$. Finally, the initial state should be an existential state. A nondeterministic automaton is* bipartite *if for any $\Lambda \subseteq \Gamma$, $\delta(q, \Lambda) \subseteq \{\epsilon\} \times Q^\forall$ for any $q \in Q^\exists$ and $\delta(q, \Lambda) \cap (A \times Q) \subseteq A \times Q^\exists$ for any $q \in Q^\forall$.*

Parity games provide automata semantics. A *parity game* is a graph with an initial vertex v^0, with a partition (V_I, V_{II}) of the vertices, and with a partial mapping r from the vertices to a given finite set of integers. A *play from some vertex v* proceeds as follows: if $v \in V_I$, then player I chooses a successor vertex v', else player II chooses a successor vertex v', and so on ad infinitum unless one player cannot make any move. The *play is winning for player I* if it is finite and ends in a vertex of V_{II}, or if it is infinite and the upper bound of the set of ranks $r(v)$ of vertices v that are encountered infinitely often is even. A *strategy for player I* is a function σ assigning a successor vertex to every sequence of vertices $\vec{v}$, ending in a vertex of V_I. A *strategy σ is memoryless* if $\sigma(\vec{v}) = \sigma(\vec{w})$ whenever the sequences $\vec{v}$ and $\vec{w}$ end in the same vertex. A *strategy for player I is winning* if all play following the strategy from the initial vertex are winning for player I. Winning strategies for player II are defined similarly. The fundamental result of parity games is the memoryless determinacy Theorem, established in [10, 8].

Theorem 3. (Memoryless determinacy) *For any parity game, one of the players has a (memoryless) winning strategy.*

Definition 10. *Given a simple automaton* $\mathcal{A} =< \Gamma, Q, Q^{\exists}, Q^{\forall}, q^0, \delta, r >$ *and a process* $\mathcal{S} =< \Gamma, S, s^0, t, L >$*, we define* the parity game $G(\mathcal{A}, \mathcal{S})$*; where the vertices of player I are in* $Q^{\exists} \times S \cup \{\top\}$ *and the vertices of player II are in* $Q^{\forall} \times S \cup \{\bot\}$*; the initial vertex* v^0 *is* (q^0, s^0)*, the other vertices and transitions are defined inductively as follows. Vertices* $\top$ *and* $\bot$ *have no successor.*

For any vertex (q, s) *and for all* $a \in A$*:*

- *there is an* ϵ*-edge to a successor vertex* (q', s) *if* $(\epsilon, q') \in \delta(q, L(s))$*,*
- *there is an* a*-edge to a successor vertex* (q', s') *if* $(a, q') \in \delta(q, L(s))$ *and* $t(s, a) = s'$*,*
- *there is an* a*-edge to a successor vertex* $\top$ *if* $(a, \rightarrow) \in \delta(q, L(s))$ *and* $t(s, a)$ *is defined, or* $(a, \not\rightarrow) \in \delta(q, L(s))$ *and* $t(s, a)$ *is not defined,*
- *there is an* a*-edge to a successor vertex* $\bot$ *if* $(a, \rightarrow) \in \delta(q, L(s))$ *and* $t(s, a)$ *is not defined, or* $(a, \not\rightarrow) \in \delta(q, L(s))$ *and* $t(s, a)$*is defined.*

The automaton $\mathcal{A}$ accepts $\mathcal{S}$ *(noted* $\mathcal{S} \models \mathcal{A}$*) if there is a winning strategy for player I in* $G(\mathcal{A}, \mathcal{S})$*.*

Like automata on infinite trees [10, 11], simple automata on processes are equivalent to bipartite non-deterministic automata. This fundamental result, due to [8, 3], is called the Simulation Theorem:

Theorem 4. (Simulation Theorem for processes) *Every simple automaton on processes is equivalent to a bipartite nondeterministic automaton.*

Since a constructive proof of Theorem 2 for $\alpha \in L_\mu$ may be found in [8, 18, 9], in order to extend it to $\mathrm{Q}L_\mu$, we consider projections of automata: it is the semantic counterpart of the existential quantification in $\mathrm{Q}L_\mu$. Projections presented here are similar to projections of nondeterministic tree automata presented in [12, 19] : projected automata are obtained by forgetting a subset of propositions in the condition of the transitions.

Definition 11. (Projection) *Let* $\Gamma \subseteq \Gamma'$ *and let* $\mathcal{A} =< \Gamma', Q, Q^{\exists}, Q^{\forall}, q^0, \delta, r >$ *be a bipartite nondeterministic automaton. The* projection of $\mathcal{A}$ on Γ *is the bipartite nondeterministic automaton* $\mathcal{A}\downarrow_\Gamma =< \Gamma, Q^{\exists} \cup Q^{\forall} \times \mathcal{P}(\Lambda), Q^{\exists}, Q^{\forall} \times \mathcal{P}(\Lambda), q^0, \delta\downarrow_\Gamma, r\downarrow_\Gamma >$*, where for all* $l \subseteq \Lambda$ *and for all* $l' \subseteq \Gamma$*:*

1. $\forall q \in Q^{\exists} : \delta\downarrow_\Gamma (q, l') = \{(\epsilon, (q', l)) \,|(\epsilon, q') \in \delta(q, l' \cup l)\}$,
2. $\forall q \in Q^{\forall} : \delta\downarrow_\Gamma ((q, l), l') = \delta(q, l' \cup l)$,
3. $\forall q \in Q^{\exists} : r\downarrow_\Gamma (q) = r(q)$ *and* $\forall q \in Q^{\forall} : r\downarrow_\Gamma ((q, l')) = r(q)$.

Theorem 5. (Projection) *Let* $\mathcal{A} =< \Gamma', Q, Q^{\exists}, Q^{\forall}, q^0, \delta, r >$ *be a bipartite nondeterministic automaton. For any process* $\mathcal{S} =< \Gamma, S, s^0, t, L >$ *on* $\Gamma \subseteq \Gamma'$*,* $\mathcal{S} \models \mathcal{A}\downarrow_\Gamma$ *if and only if there exists a labeling process* $\mathcal{E}$ *on* $\Lambda = \Gamma' \setminus \Gamma$ *such that* $\mathcal{S} \times \mathcal{E} \models \mathcal{A}$*.*

Proof. First, suppose $\mathcal{S} \models \mathcal{A}\downarrow_\Gamma$. Let σ be a winning memoryless strategy for player I in the game $G = G(\mathcal{A}\downarrow_\Gamma, \mathcal{S})$ (Theorem 3) and let $V_{II} \subseteq Q^{\forall} \times \mathcal{P}(\Lambda) \times S$ be the set of nodes from player II in G without $\bot$. Let $\mathcal{E} \in Lab_\Lambda$ be an arbitrary

completion of the process $< \Lambda, V_{II}, \sigma(q^0, s^0), t', L' >$, where for any (q, l, s) $in V_{II}$, $L'(q, l, s) = l$, and for all $a \in A$, $t'((q, l, s), a) = \sigma(s', q')$, for some (unique) (s', q') such that there is an a-arc from $((q, l), s)$ to (q', s') in G. Then, it can be show that we define a winning strategy σ' for player I in the game $G(\mathcal{A}, \mathcal{S} \times \mathcal{E})$ by $\sigma'((s, \sigma(s, q)), q) = \sigma(s, q)$. Reciprocally, suppose $\mathcal{S} \times \mathcal{E} \models \mathcal{A}$ for some $\mathcal{E} \in Lab_\Lambda$. It suffice to show that any memoryless winning strategy for player I in $G(\mathcal{A}, \mathcal{S} \times \mathcal{E})$ defines a memory winning strategy for player I in the game $G(\mathcal{A}{\downarrow_\Gamma}, \mathcal{S})$. □

We prove now Theorem 2, by induction on the structure of $\alpha \in \mathrm{Q}L_\mu(\Gamma)$. We only prove the case where α is quantified, since $\alpha \in L_\mu$ is dealt with following [8] and [9]. Without loss off generality, we can assume α of the form $\mathrm{Q}\Lambda\alpha'$, where $\mathrm{Q} \in \{\exists, \forall\}$. For $\mathrm{Q} = \exists$, let $\mathcal{A}$ be the bipartite nondeterministic automaton equivalent to $\mathcal{A}_{\alpha'}$ (Theorem 4). Now take $\mathcal{A}_\alpha = \mathcal{A}{\downarrow_\Gamma}$ and conclude by Theorem 5. The case where $\mathrm{Q} = \forall$ is obtained by complementation : since parity games are determined (Theorem 3), we complement $\mathcal{A}_{\exists\Lambda\neg\alpha}$ (as in [11]) to obtain $\mathcal{A}_\alpha$
Theorem 2 gives an effective construction of finite controllers on finite processes: given a finite process $\mathcal{S}$ and a sentence $\exists c.\alpha \in \mathrm{Q}L_\mu$ expressing a control problem, we construct the automaton $\mathcal{A}_{(\exists c.\alpha)}$. If we find a memoryless winning strategy in the finite game $G(\mathcal{A}_\alpha, \mathcal{S})$, Theorem 5 gives a finite controller. Otherwise, there is no solution. We can show that the complexity of such problem is $(k+1)EXPTIME - complete$ where k is the number of alternations of existential and universal quantifiers in α. The result of [2] is retrieved: synthesizing controllers for open systems is $2EXPTIME - complete$ for mu-calculus control objectives.

5 Controller Synthesis

This section illustrates the constructions on a simple example. The plant $\mathcal{S}$ (to be controlled) is drawn next page. Both states s^0 and s^1 are labeled with the empty set of propositions, thus $\mathcal{S}$ is a process on $\Gamma = \emptyset$. The control objective is the formula $\alpha = \nu Y.\ <b>\ Y \wedge\ <a>\ (\mu X.[a]X)$. There is a controller of $\mathcal{S}$ for α iff $\mathcal{S} \models \exists c.\alpha * c$ but also iff $\mathcal{S} \models \exists c.c \wedge \alpha * c$. Let ϕ be the formula $c \wedge \alpha * c \equiv c \wedge \nu Y.\ <b>\ (c \wedge Y) \wedge\ <a>\ (c \wedge \mu X.[a](c \Rightarrow X))$. The bipartite nondeterministic automaton $\mathcal{A}_\phi$ is shown in Figure 1, where the following graphical conventions are used: circled states are existential states, while states enclosed in squares are universal states; the transitions between states are represented by edges in $\{a, b, \epsilon\} \times \mathcal{P}(\{c\})$; the other transitions are represented by labeled edges from states to the special boxe containing the symbol $\rightarrow$. The rank function maps q^0 to 2 and q_2 to 1. The projected automaton $\mathcal{A}_\phi{\downarrow_\emptyset}$ is shown in Figure 2, using similar conventions. Note that all transitions are labeled in $\{a, b, \epsilon\} \times \{\emptyset\}$, since $\mathcal{A}_\phi{\downarrow_\emptyset}$ is an automaton on $\Gamma = \emptyset$, but all universal states are now labeled in $Q \times \mathcal{P}(\{c\})$, as a result of the projection. Now, $\mathcal{S} \models \exists c.\phi$ iff $\mathcal{A}_\phi{\downarrow_\emptyset}$ accepts $\mathcal{S}$ and this condition is equivalent to the existence of a winning strategy for player I in the finite parity game $G(\mathcal{A}_\phi{\downarrow_\emptyset}, \mathcal{S})$ of Figure 3. Clearly, player I has an unique memoryless wining strategy σ, that maps the vertex (q_2, s^0) to $(q_2', \emptyset, s^0)$. The labeling process $\mathcal{E}$ on $\{c\}$ derived from σ is shown in Figure 4. Four states and

transitions between them are first computed, yielding an incomplete process on $\{c\}$. A last state c is then added so as to obtain a complete process. The dashed transitions (and all dead transitions) are finally suppressed to yield the synthesized controller.

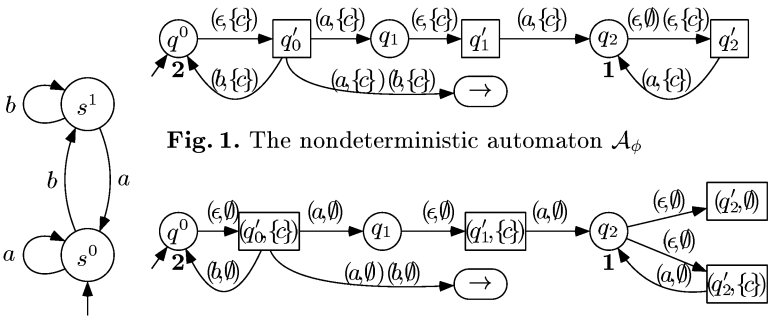

Fig. 1. The nondeterministic automaton $\mathcal{A}_\phi$

Fig. 2. The nondeterministic automaton $\mathcal{A}_{\exists c.\phi}$

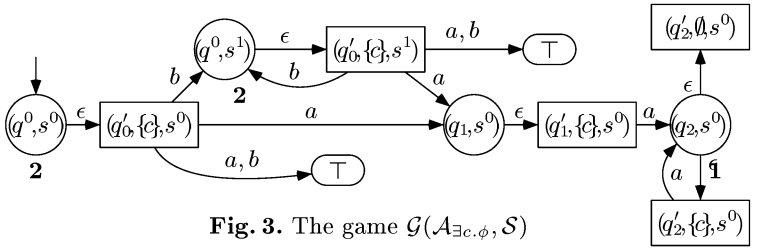

Fig. 3. The game $\mathcal{G}(\mathcal{A}_{\exists c.\phi}, \mathcal{S})$

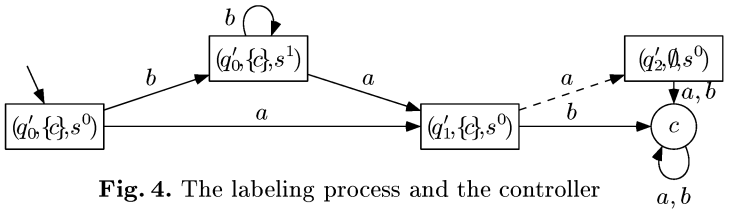

Fig. 4. The labeling process and the controller

6 Conclusion

The logical formalism we have developed allows to synthesize controllers for a large class of control objectives. All the constraints, as maximally permissive controllers or admissible ones for open systems, are formulated as objectives. As it is, the class of controllers is left free and we cannot, for example, deal with partial observation. The recent work of [3] offers two constructions that we can use to interpret the quantified mu-calculus relatively to some fixed classes of labeling processes. The first construction, the quotient of automata, forces the labeling processes to be in some mu-calculus (definable) class. It can be seen as a generalization of the automata projection, and used instead. The quantified mu-calculus could hence be extended by constraining each quantifier to range

over some mu-calculus class. Nevertheless, the class of controllers under partial observation being undefinable in the mu-calculus, we need to consider the second construction: the quotient of automata over a process exhibits (when it exists) a controller under partial observation inside some mu-calculus class. The outermost quantification of a sentence is then made relative to some class of partial observation. Therefore, we can seek a controller under partial observation for open systems, but we cannot synthesize a maximally permissive controller among the controllers under partial observation.

References

1. Ramadge, P.J., Wonham, W.M.: The control of discrete event systems. Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems **77** (1989)
2. Kupferman, O., Madhusudan, P., Thiagarajan, P., Vardi, M.: Open systems in reactive environments: Control and synthesis. CONCUR 2000, LNCS 1877.
3. Arnold, A., Vincent, A., Walukiewicz, I.: Games for synthesis of controllers with partial observation. To appear in TCS (2003)
4. Vincent, A.: Synthèse de contrôleurs et stratégies gagnantes dans les jeux de parité. MSR 2001
5. Sistla, A., Vardi, M., Wolper, P.: The complementation problem for Buchï automata with applications to temporal logic. TCS**49** (1987)
6. Kupferman, O.: Augmenting branching temporal logics with existential quantification over atomic propositions. Journal of Logic and Computation **9** (1999)
7. Patthak, A.C., Bhattacharya, I., Dasgupta, A., Dasgupta, P., Chakrabart, P.P.: Quantified computation tree logic. IPL **82** (2002)
8. Arnold, A., Niwinski, D.: Rudiments of mu-calculus. North-Holland (2001)
9. Walukiewicz, I.: Automata and logic. In: Notes from EEF Summer School'01. (2001)
10. Emerson, E.A., Jutla, C.S.: Tree automata, mu-calculus and determinacy. FOCS 1991. IEEE Computer Society Press (1991)
11. Muller, D.E., Schupp, P.E.: Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. TCS **141** (1995)
12. Rabin, M.O.: Decidability of second-order theories and automata on infinite trees. Trans. Amer. Math. Soc. **141** (1969)
13. Dietrich, P., Malik, R., Wonham, W., Brandin, B.: Implementation considerations in supervisory control. In: Synthesis and Control of Discrete Event Systems. Kluwer Academic Publishers (2002)
14. Bergeron, A.: A unified approach to control problems in discrete event processes. Theoretical Informatics and Applications **27** (1993)
15. Emerson, E.A., Sistla, A.P.: Deciding full branching time logic. Information and Control **61** (1984)
16. Emerson, E.A., Jutla, C.S., Sistla, A.P.: On model-checking for fragments of mu-calculus. CAV 1993, LNCS 697.
17. Streett, R.S., Emerson, E.A.: The propositional mu-calculus is elementary. ICALP 1984, LNCS 172.
18. Kupferman, O., Vardi, M.Y., Wolper, P.: An automata-theoretic approach to branching-time model checking. Journal of the ACM **47** (2000)
19. Thomas, W.: Automata on infinite objects. In Leeuwen, J.v., ed.: Handbook of TCS, vol. B. Elsevier Science Publishers (1990)