

# Second Order Quantification in Temporal Logics and their Applications to Control Theory

Sophie.Pinchinat@irisa.fr  
IRISA-Rennes, France

joint work with Stéphane Riedweg (PhD)

## Plan

- The Principle of the Approach
- Controller Synthesis as Model-Checking
- Examples of Specifications
- The Synthesis Procedure Principles
- Extensions and Discussion

## Processes for Systems

$$\mathcal{S} = \langle S, s^0, \rightarrow, L \rangle \text{ on } \Gamma \subseteq AP$$

where

- $\Sigma = \{a, b, \dots\}$  events
- $AP = \{p, p', c, c', \dots\}$  (atomic) propositions
- $S, s^0 \in S$  set of states, initial state,
- $\rightarrow: S \times \Sigma \rightarrow S$  transition (partial) function ( $s \xrightarrow{a} s'$ )
- $L: S \rightarrow 2^\Gamma$  labeling of states by propositions

## Synchronous Product

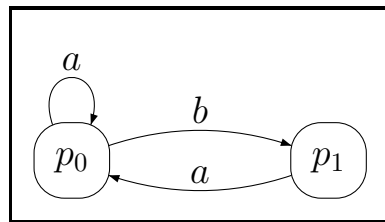
- $\mathcal{S}_1 = \langle S_1, s_1^0, \rightarrow_1, L_1 \rangle$  on  $\Gamma_1$  and  $\mathcal{S}_2 = \langle S_2, s_2^0, \rightarrow_2, L_2 \rangle$  on  $\Gamma_2$

$$\mathcal{S}_1 \times \mathcal{S}_2 = \langle S_1 \times S_2, (s_1^0, s_2^0), \rightarrow, L \rangle$$

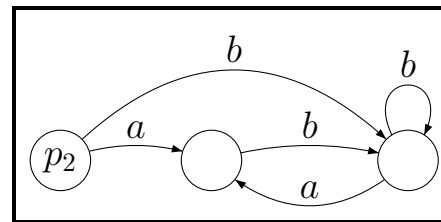
$(s_1, s_2) \xrightarrow{a} (s'_1, s'_2)$  whenever  $(s_1 \xrightarrow{a} s'_1)$  and  $(s_2 \xrightarrow{a} s'_2)$

$$L(s_1, s_2) = L_1(s_1) \cup L_2(s_2)$$

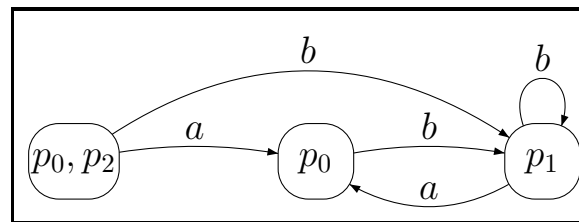
## Example of Synchronous Product



$\mathcal{S}_1$



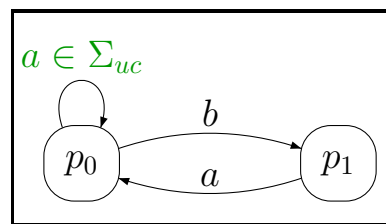
$\mathcal{S}_2$



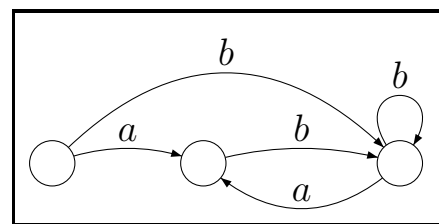
$\mathcal{S}_1 \times \mathcal{S}_2$

## Control Problems (with mu-calculus definable objectives)

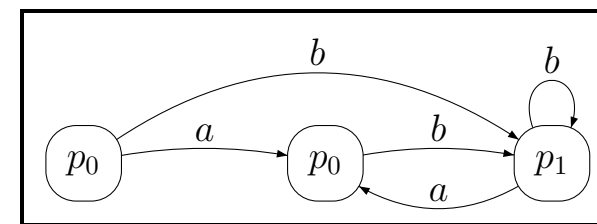
- $\Sigma = \Sigma_{uc} \uplus \Sigma_c$
- **Controllers** are processes (often with no propositions)
- **Admissible** Controllers do not disable events of  $\Sigma_{uc}$
- We want to synthesise a controller  $C$  s.t.
  - $C$  is admissible (or in some other  $L_\mu$  definable class)
  - $S \times C$  satisfies the formula  $\varphi \in L_\mu$
- More generally, we consider  $S \times C_1 \times \dots \times C_n$



$S$



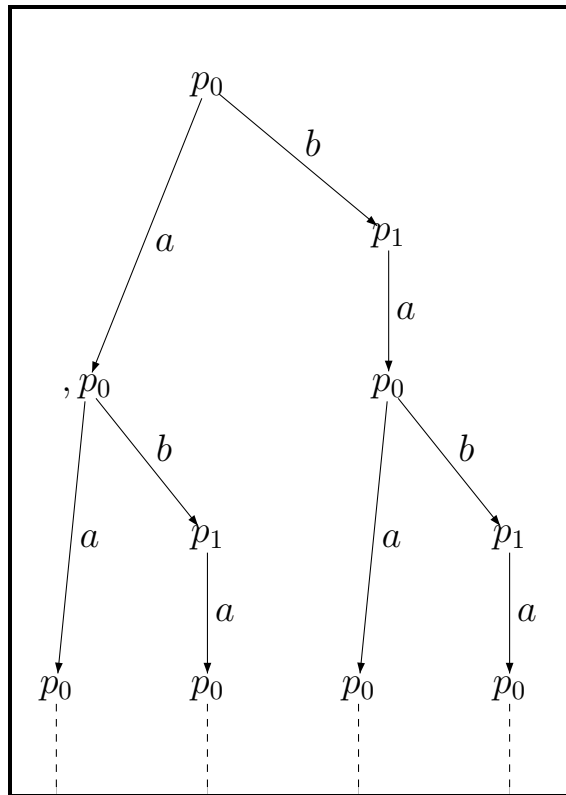
$C$



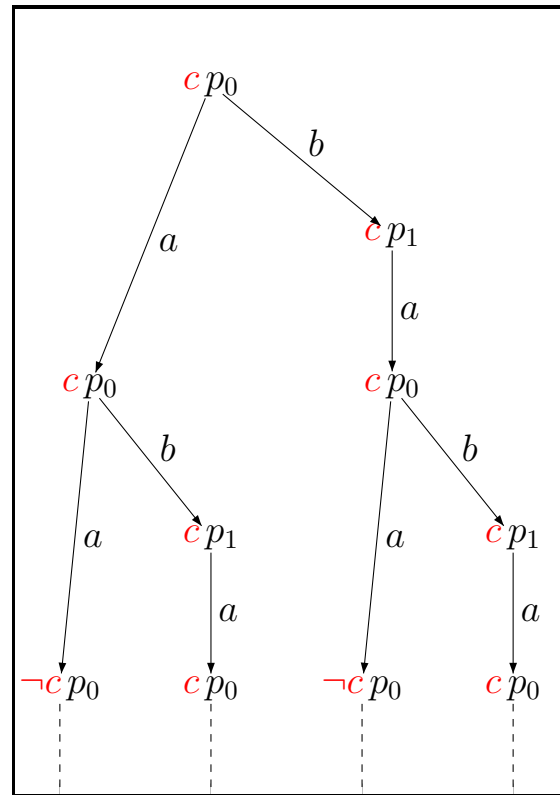
$S \times C$  satisfies  $\varphi$



## c-labeling of $\mathcal{S}$



computation tree of  $\mathcal{S}$



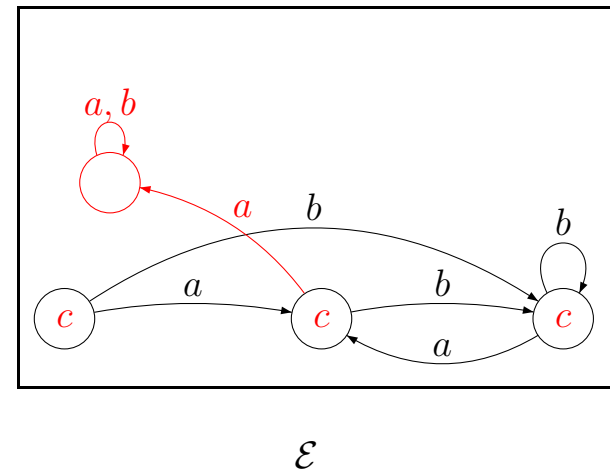
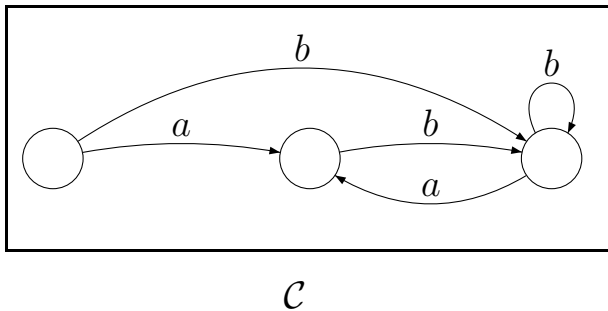
computation tree of  $\mathcal{S} \times \mathcal{E}$

## Labeling Processes are Controllers

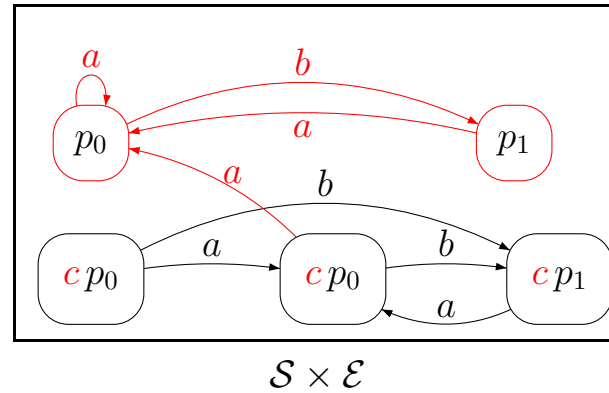
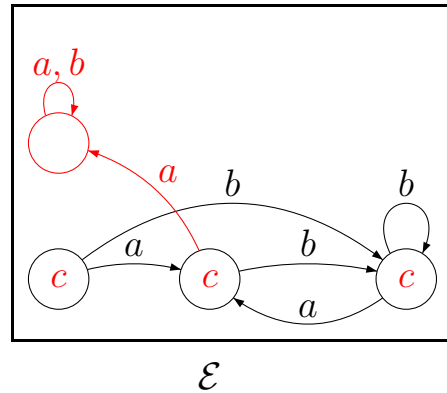
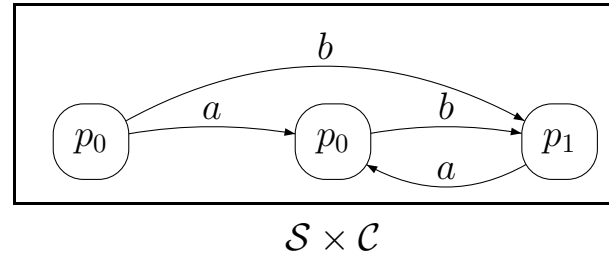
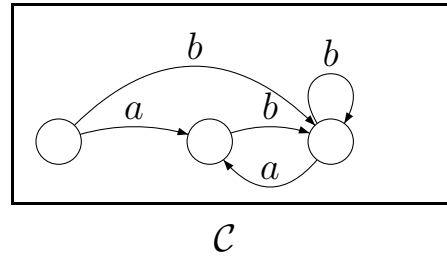
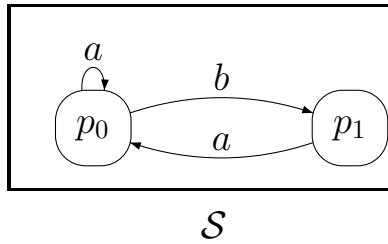
### ■ $c$ -labeling process

$\mathcal{E} = \langle E, e^0, \rightarrow', L' \rangle$  a process on  $\{c\}$

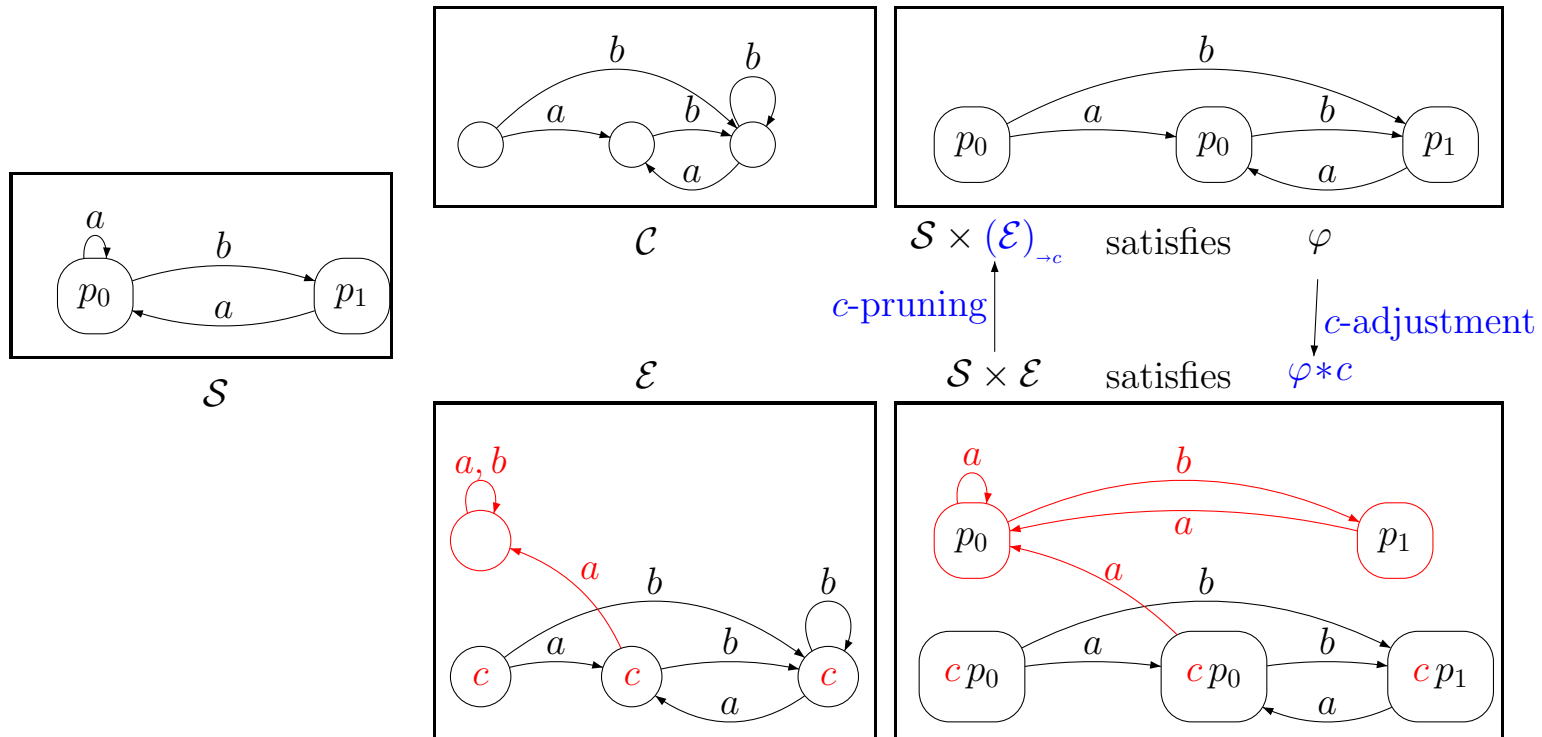
which is **complete**, i.e.  $e \xrightarrow{a'}$  is always defined.



## Labeling Processes and Controllers



# Pruning vs Adjustment



## c-Adjustment of formulas

- the  $c$ -adjustment of  $\varphi$  is  $\varphi * c$

It adjusts the formula to talk only about trajectories “inside”  $c$ :

essentially  $(\langle a \rangle \dots) * c \stackrel{\text{def}}{=} \langle a \rangle (c \wedge (\dots) * c)$

- CTL logic

$(\mathbf{EX} \dots) * c \stackrel{\text{def}}{=} \mathbf{EX}(c \wedge (\dots) * c)$

$(\mathbf{E} \dots \mathbf{U} \dots) * c \stackrel{\text{def}}{=} \mathbf{E}(c \wedge (\dots) * c \mathbf{U} c \wedge (\dots) * c)$

Hence  $(\mathbf{AX} \dots) * c \stackrel{\text{def}}{=} \mathbf{AX}(c \Rightarrow (\dots) * c) \dots$

Pruning

Adjustment

Theorem

$$\mathcal{S} \times \mathcal{E}_{\rightarrow c} \models \varphi \text{ iff } \mathcal{S} \times \mathcal{E} \models \varphi * c$$

where  $\mathcal{E}_{\rightarrow c}$  is  $c$ -pruning the of  $\mathcal{E}$  (keeps states “inside”  $c$ )

## The Quantified Mu-Calculus $QL_\mu$

- Syntaxe of  $QL_\mu$   $\exists c.\alpha \mid \neg\alpha \mid \alpha \vee \alpha' \mid \beta$

where  $c \in AP$  and  $\beta \in L_\mu$

- $\mathcal{S}, s \models \exists c.\alpha$  iff there exists  $\mathcal{E} = \langle E, \varepsilon^0, t', L' \rangle$  on  $\{c\}$  s.t.

$$\mathcal{S} \times \mathcal{E}, (s, \varepsilon^0) \models \alpha$$

- $\mathcal{S} \times \mathcal{E}$  is a  $c$ -labeling of  $\mathcal{S}$  that is (an unfolding of)  $\mathcal{S}$  with the new label  $c$  put somehow; the label  $c$  denotes a control policy
- Bisimulation invariant

## Specifying Controllers

For any  $\varphi \in L_\mu$ , there exists an **admissible controller**  $\mathcal{C}$  of  $\mathcal{S}$  for  $\varphi$  if and only if  $\mathcal{S} \models \exists c. \mathbf{AG} (\bigwedge_{u \in \Sigma_{uc}} [u]c) \wedge \varphi * c$

Use the pattern

$$\exists c. \boxed{\dots} \wedge \boxed{\dots} * c$$

- **Generalized Admissibility**  $\mathbf{AG} (\bigwedge_{u \in \Sigma} UnCont(u) \Rightarrow [u]c)$
- **Non-Blocking**
  - Task achievement  $\mathbf{AG} (\mathbf{EF} (m) * c)$
  - Livelock  $\mathbf{AG} (\bigvee_{a \in \Sigma} \langle a \rangle \top) * c$
- **Open Systems**  $\exists c. \forall e. [\beta(c) \wedge \beta(e) \Rightarrow (\varphi * e) * c]$  (can use  $\varphi * (e \wedge c)$ )

## My Favorite : the Theorem for Maximally Permissive Controllers

There exists a controller of  $\mathcal{S}$  for  $\varphi$  which is **maximally permissive**

if and only if

$$\mathcal{S} \models \exists c. \text{Solution}(c, \varphi) \wedge \forall c'. [c \sqsubseteq c' \Rightarrow \neg \text{Solution}(c', \varphi)]$$

- $c \sqsubseteq c'$ : “**labeled by  $c$  implied labeled by  $c'$** ” is  $L_\mu$  definable

RMK You can also specify **unicity**

## Model-Checking and Synthesis

- Mu-calculus formulas are equivalent to parity tree automata

[Emerson & Jutla 1991]  $\beta \rightsquigarrow \mathcal{A}_\beta$  and the parity game  $\mathcal{G}(\mathcal{S}, \mathcal{A}_\beta)$

- $QL_\mu$  formulas also

$\mathcal{A}_{\exists c.\alpha}$  is the  $c$ -projection of  $\mathcal{A}_\alpha$  [Rabin69]

$\Rightarrow$  **Small Model Property** (finite controllers)

Needs the Simulation Theorem [MullerSchupp95]

[RiedwegPinchinat03]  $\mathcal{A}_{\exists c.\alpha}$  to retrieve suitable values of  $c$  in  $\mathcal{G}(\mathcal{S}, \mathcal{A}_{\exists c.\alpha})$

## Model-Checking and Synthesis for a formula $\exists c.\beta$

- 1. build the **parity tree automaton**  $\mathcal{A}_{\exists c.\beta}$ ;
- 2. compute the parity game  $G(\mathcal{A}, \mathcal{S})$ ;
- 3. find a winning **memoryless** strategy (if any) (determined by [EJ91])
- 4. it tells how to  $c$ -label the comp. tree of  $\mathcal{S}$  in a **regular** manner
- 5. take the finite state corresponding  $\mathcal{E}$  to compute  $\mathcal{C} = \mathcal{E}_{\rightarrow c}$

Complexity  $EXPTIME(|\beta|)$  but Polynomial in the size of  $\mathcal{S}$

## A few Remarks

- **Max Perm Controllers** difficult problem in the full temporal logic setting [Thistle94]

$$\exists c. \text{Solution}(c, \beta) \wedge \forall c'. [c \sqsubseteq c' \Rightarrow \neg \text{Solution}(c', \beta)]$$

Independent of the plant

Complexity  $2EXPTIME(|\beta|)$  still Polynomial in the size of  $\mathcal{S}$

- As expected, the Model-checking of  $QL_\mu$  is non-elementary

## Extensions of the Work

- **Theorem** If  $\varphi = Q_1 c_1(O_1) \dots Q_n c_n(O_n) . \varphi$ , where  $\forall i, O_i \subseteq O_{i+1}$ , we can build  $\mathcal{A}_{(\varphi, \mathcal{S})}$  s.t.  $\mathcal{S} \models \varphi$  if and only if  $\mathcal{A}_{(\varphi, \mathcal{S})}$  has a model.

particular case of “well-synchronized hyp.” (Igor Walukiewicz)

- **Decentralized Controllers**  $\exists c_1(O_1) . \exists c_2(O_2) . \beta * (c_1 \wedge c_2) \wedge \dots$
- **Other Architectures** [PinchinatLeNay05]  $I(c_1, \dots, c_n)$  SOS spec.
- **Maximally Permissive Controllers !!**  
 $\exists c(O) . \text{Solution}(c, \alpha) \wedge \forall c'(O) . [c \sqsubseteq c' \Rightarrow \neg \text{Solution}(c', \alpha)]$
- **Non-deterministic Framework** [PinchinatRaclet05]

## Discussion

- Another proof of upper bounds Control of Open Systems [KMTV00]
- $QL_\mu$  is more expressive than ATL, ATL\*, AMC, Game Logic
- Other semantics for quantification: without unfolding [Kupferman9x]  
 $\exists L_\mu$  model-checking is NP-complete
- Heuristic to place  $c$  which avoids to apply the Simulation Theorem
- Other use of labelling process synthesis: Adding transitions, Labelling for Games to reason about strategies, maximal strategies; what about characterizing all the solutions (permissive games), distributed games?