Minimising Makespan of Discrete Controllers: A Qualitative Approach

Ezequiel Castellano[1], Victor Braberman[2], Nicolás D'Ippolito[2], Sebastián Uchitel[2,3] and Kenji Tei[4]

Abstract— **Qualitative controller synthesis techniques produce controllers that guarantee to achieve a given goal in the presence of an adversarial environment. However, qualitative synthesis only produces one controller out of many possible solutions and typically does not provide support for expressing preferences over other alternatives. In this paper, we thus present a formal approach to reason about preferences qualitatively, restricting attention to makespan of discrete event-based controllers for reachability goals. Time is reasoned upon symbolically, which relieves the user from providing concrete quantitative measures. In particular, we study the scenario in which durations of individual activities are not known up-front. We first show how controllers can be symbolically and fairly compared by fixing the contingencies. Then, we present an algorithm to produce controllers that are makespan-minimising.**

I. Introduction

The problem of automatically synthesising event-based solutions from environment models and qualitative goal specifications has been widely studied [Ramadge and Wonham, 1984, Pnueli and Rosner, 1989, Cimatti et al., 2003, Piterman et al., 2006]. In these problems, the environment and the goals are specified by using a formal language. The environment is typically modelled as a state machine whose actions are partitioned into controllable and uncontrollable actions. The controller synthesis problem is to automatically produce a solution, i.e., a controller, that by only disabling controllable actions guarantees the satisfaction of the goals. In particular, we focus on reachability and safety goals which are of interest to supervisory control theory [Ramadge and Wonham, 1984], conformant [Goldman and Boddy, 1996] and contingent [Pryor and Collins, 1996] planning.

Qualitative control problems are boolean in the sense that a controller satisfies a set of goals, or it does not. When a qualitative control problem has a solution, we say it is realisable. Realisable control problems may allow for several possible solutions. Different solutions may differ in the strategy which they apply to satisfy the goals. Typically, based on the arrival of monitored actions, the strategies implemented by controllers decide which and when to start activities. For instance, regarding end-to-end makespan, a controller that starts several activities concurrently instead of executing them sequentially can be, intuitively, considered as a better strategy, no matter which the durations of the activities are. Unfortunately, qualitative synthesis procedures are, so far, oblivious to such considerations. The controller produced is one of the many alternatives and users cannot specify their preferences; e.g., lower-makespan controller. Thus, it is desired to have the ability to express preferences and automatically compute a solution from a set of possible solutions to a control problem accordingly.

Synthesis and planning techniques that allow expressing preferences exist, such as those regarding performance or reliability. Such quality attributes are modelled by introducing a quantitative aspect to the system specification, which imposes a preference order on the controllers that satisfy the qualitative part of the specification [Bloem et al., 2009, Chatterjee et al., 2005, Thrun et al., 2005]. However, from a practical perspective, these approaches require modelling quality attributes quantitatively, whereas in many cases, such detailed representation is not available, possible, or desired.

In this paper, we define lower-makespan as our preference and introduce a formal approach to qualitatively reason about makespan of discrete event-based controllers for safety and reachability goals. To reason qualitatively about makespan of controllers, we introduce a symbolic time metric derived from Parametric Timed Automata (PTA) [Alur et al., 1993] semantics. This metric requires modelling sub-tasks of the problem which take time as activities. However, no quantitative information about the duration of the activities is required. Then, we define a mechanism to compare makespan of controllers under unknown durations of activities and event contingencies produced by uncontrollable behaviour. Such a comparison is made through exhaustive analysis by using a symbolic computation over the parameters of a PTA [Henzinger, 2000] and Satisfiability Modulo Theories (SMT) solving [Barrett and Tinelli, 2018]). The parameters of the PTA represent the uncertain duration of the activities. Then, we define makespan-minimising controllers by using the symbolic comparison and we introduce an algorithm that produces a makespan-minimising controller qualitatively.

The paper is structured as follows. Section II introduces the background. Section III presents the control problem with activities. Section IV shows an example that is used throughout the rest of the paper. Section V defines how to compare the makespan of a pair of discrete controllers without using quantitative information. Section VI defines makespan-minimising controllers and presents an algorithm that produces a makespan-minimising controller. Section VII introduces the related work. Finally, Section VIII presents the conclusions and directions for our future work.

[1]SOKENDAI & National Institute of Informatics, Japan `ecastellano@nii.ac.jp`
[2]University of Buenos Aires & CONICET, Argentina `{vbraber, ndippolito, suchitel}@uba.dc.ar`
[3]Imperial College London, United Kingdom
[4]Waseda University, Japan `ktei@aoni.waseda.jp`

II. Background

We use a standard definition of LTS to model behavioural models and parallel composition [Baier and Katoen, 2008] to model the interaction between LTSs. Parallel composition is defined as an LTS that models the asynchronous execution of composed models, interleaving non-shared actions while forcing synchronisation on shared actions. LTS models are deterministic, which means that given a state and an action there is at most one successor. We assume that the actions Σ is partitioned into controllable Σ_c and uncontrollable Σ_u actions ($\Sigma = \Sigma_c \cup \Sigma_u$).

Definition 1 (LTS): A deterministic *LTS* is a tuple (Q, Σ, Δ, q_0), where Q is a finite set of states, $\Sigma \subseteq Act$ is its alphabet, Act is the set of observable actions, $\Delta \subseteq (Q \times \Sigma \times Q)$ is a transition relation, $q_0 \in Q$ is the initial state, and deterministic means $\forall q \in Q\ \forall \ell \in \Sigma\ \{(q, \ell, q') \mid (q, \ell, q') \in \Delta\} \leq 1$.

Throughout the paper we use the following notations.

- *Step*: $q \xrightarrow{\ell} q'$ is equivalent to $(q, \ell, q') \in \Delta$.
- *Enabled Actions*: $\Delta(q) = \{\ell \mid q \xrightarrow{\ell} q'\}$.
- *Successors*: $\Delta(q, \ell) = \{q' \mid q \xrightarrow{\ell} q'\}$.
- *Enabled Controllable Actions*: $\Delta^c(q) = \Delta(q) \cap \Sigma_c$.
- *Enabled Uncontrollable Actions*: $\Delta^u(q) = \Delta(q) \cap \Sigma_u$.

Besides, when the members of a tuple are not explicitly described we assume them to be indexed by the name of the tuple. For instance, given an LTS M we refer to its members as $M = (Q_M, \Sigma_M, \Delta_M, q_{M_0})$ with q_M a state of Q_M.

Definition 2 (Parallel Composition): The *parallel composition* ($\|$) of two LTSs M and N, is a symmetric operator such that $M\|N = (Q_M \times Q_N, \Sigma_M \cup \Sigma_N, \Delta_{M\|N}, (q_{M_0}, q_{N_0}))$, where $\Delta_{M\|N}$ is the smallest relation that satisfies the following rules:

$$\frac{q_M \xrightarrow{\ell}_M q_M{'}}{(q_M, q_N) \xrightarrow{\ell}_{M\|N} (q_M{'}, q_N)}\ \ell \in \Sigma_M \setminus \Sigma_N \qquad \frac{q_N \xrightarrow{\ell}_N q_N{'}}{(q_M, q_N) \xrightarrow{\ell}_{M\|N} (q_M, q_N{'})}\ \ell \in \Sigma_N \setminus \Sigma_M$$

$$\frac{q_M \xrightarrow{\ell}_M q_M{'},\ q_N, \xrightarrow{\ell}_N, q_N{'}}{(q_M, q_N) \xrightarrow{\ell}_{M\|N} (q_M{'}, q_N{'})}\ \ell \in \Sigma_M \cap \Sigma_N$$

We use Fluent Linear Temporal Logic (FLTL) [Giannakopoulou and Magee, 2003] as the language for describing properties. FLTL is a linear-time temporal logic for reasoning about fluents instead of state-based propositions. A *fluent* *Fl* is defined by a pair of sets and a Boolean value: $Fl = \langle I_{Fl}, T_{Fl}, Init_{Fl} \rangle$, where $I_{Fl} \subseteq Act$ is the set of initiating actions, $T_{Fl} \subseteq Act$ is the set of terminating actions and $I_{Fl} \cap T_{Fl} = \emptyset$. A fluent may be initially *true* or *false* as indicated by $Init_{Fl}$. Every action $\ell \in Act$ induces a fluent, namely $\ell = \langle \ell, Act \setminus \{\ell\}, false \rangle$. The logic has the same expressiveness as standard LTL [Baier and Katoen, 2008]. However, as fluents can be used to overlay state-based propositions on an event-based model, FLTL allows for a more compact representation of properties.

Let $\mathcal{F}$ be the set of all possible fluents over the observable actions Act. An FLTL formula is defined inductively by using the standard Boolean connectives and temporal operators $\mathbf{X}$ (next), $\mathbf{U}$ (strong until) as follows:

$$\varphi ::= Fl \mid \neg\varphi \mid \varphi \vee \psi \mid \mathbf{X}\varphi \mid \varphi \mathbf{U} \psi$$

where $Fl \in \mathcal{F}$. As well, we use the standard operators $\wedge$, $\Diamond$ (eventually), $\Box$ (always), and $\mathbf{W}$ (weak until). FLTL formula satisfaction is computed over traces and fluents, it is standard and omitted. We denote that a possibly infinite trace π satisfies an FLTL formula φ as $\pi \models \varphi$.

III. Control Problems with Activities

The problem of controller synthesis can be expressed as follows. Given an *LTS* model E of the environment, a goal G expressed in *FLTL*, and a set of controllable actions Σ_c, the aim is to build an LTS C that $i)$ when the controller C is concurrently executed with E (i.e., $E\|C$), it does not block any uncontrollable action in the environment, and $ii)$ G is satisfied in every *trace* of $E\|C$. The notion of a controller that does not block uncontrollable actions is built on the concept of the *legal environment* for Interface Automata [de Alfaro and Henzinger, 2001]. Intuitively, C is a legal environment for E, when in every state (q_E, q_C) of $E\|C$, an uncontrollable action is enabled in (q_E, q_C) iff it is also enabled in q_E. In this paper, we restrict attention to goals of the form $G = \Box S \wedge \Diamond P$, where S (safety goals) and P (reachability goals) are propositional FLTL formulas.

In LTS models, time only passes on states, and transitions change states instantaneously. A priori, the amount of time that passes between two different actions is unknown. The standard way of modelling the passing of time in LTS is to incorporate actions that represent the start and the end of activities which take time. This is the approach we adopt in this paper. We assume that *all* activities that have duration will be modelled with start and end actions. We also assume that, if getting a chance, the controller is fast enough to be ready to take enabled transitions. Thus, states in which there is at least one outgoing controllable action ($\Delta^c(q) \neq \emptyset$) are considered to the be ones in which time does not pass. Such states are called *transient states* [Letier and Heaven, 2013].

Thereupon, we require LTSs to be annotated with a set of activity definitions $\mathcal{AD} = (\mathcal{A}, Start, Ends)$. $\mathcal{A}$ is a set of symbols that represent each activity α uniquely. The function $Start : \mathcal{A} \to \Sigma_c$ defines the controllable action that starts an activity. The function $Ends : \mathcal{A} \to 2^{\Sigma_u}$ defines the uncontrollable actions that determine the end of an activity. These two functions must be defined for each activity $\alpha \in \mathcal{A}$. An action can be related to at most one activity. Given a path $q_0 \xrightarrow{\ell_1} \ldots \xrightarrow{\ell_k} \ldots \xrightarrow{\ell_i} \ldots \xrightarrow{\ell_m} q_m$, we say that activity $\alpha \in \mathcal{A}$ is *running* in the state q_m, if there exists a $k \leq m$ such that $\ell_k = Start(\alpha)$ and $\ell_i \notin Ends(\alpha)$ for every $k < i \leq m$. That is to say, the activity has started but not ended at the given state. We also require, w.l.o.g., the set of *running activities* $\zeta(q)$ to be the same for all paths leading to a given state q. We assume that an activity cannot be started again while running and that no activities are running in the goal states of a controller. The *goal states* Q_G are those in $Q_{E\|C}$ that can be reached through a path from the initial state that does not contain another state satisfying the reachability property. In other words, it exists a path $q_0 \xrightarrow{\ell_1} \ldots \xrightarrow{\ell_m} q_m \xrightarrow{\ell_G} q_G$ s.t

$\ell_1 \dots \ell_m \not\models G \wedge \ell_1 \dots \ell_m, \ell_G \models G$. Note that every path of $E \| C$ satisfies the safety requirements.

Definition 3: (Control Problem with Activities) Given an LTS $E = (Q_E, \Sigma, \Delta_E, q_{E_0})$, a set of activity definitions $\mathcal{AD} = (\mathcal{A}, Start, Ends)$, a safety and reachability goal $G = \Box S \wedge \Diamond P$ expressed in FLTL, and a set of controllable actions $\Sigma_c \subseteq \Sigma$, the solution to the control problem $\mathcal{E} = \langle E, \mathcal{AD}, G, \Sigma_c \rangle$ is to find a deterministic LTS $C = (Q_C, \Sigma, \Delta_C, q_{C_0})$ such that $i)$ C is a legal environment for E, $ii)$ $E \| C$ is deadlock free, and $iii)$ every infinite trace π in $E \| C$ satisfies G ($\pi \models G$).

In controller synthesis problems with reachability goals, algorithms that produce memoryless solutions have linear complexity in the size of the problem. These controllers are a sub-graph of the LTS defined by the parallel composition of the environment with the LTSs that represent the fluents used to describe the goals ($E \| M_{fl_1} \| \dots \| M_{fl_k}$). From a game-theoretical perspective, this composition represents the arena of the game between the environment and the controller. Furthermore, universal controllers [Thomas, 1995] are those that subsume any other memoryless controller. If we restrict attention to the sub-graph resulting from removing all outgoing transitions from goal states, universal controllers have the form of directed acyclic graphs (DAG). This observation is important to the rest of the paper.

IV. Industrial Automation Example

Assume an industrial automation setting, in which different product types are to be produced through a variety of processing activities according to given constraints. The specification of the control problem is given as a set of LTSs describing the behaviour of the environment (Figure 1) and a set of FLTL formulas describing production constraints (Figure 2). For simplicity, we assume that there are two product types (T_1 and T_2). Type T_1 products require activity A_3, while T_2 type products require activity A_1 and A_2. After finishing the required activities, products must undergo a quality check (R1). When the quality check fails, it is required to do a repair activity (R2). Once the product has passed the quality requirements, it is signalled with the action *done* (R3). Additionally, for safety reasons, A_2 cannot be started while A_1 is ongoing (R4). The goal is to produce one product of the required type ($\Diamond done$) while always satisfying the production constraints ($\Box(R_1 \wedge R_2 \wedge R_3 \wedge R_4)$).

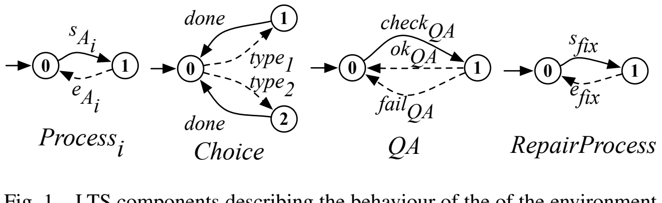


Fig. 1. LTS components describing the behaviour of the of the environment $E = Process_1 \| Process_2 \| Process_3 \| Choice \| QA \| RepairProcess$.[1]

R1) $\dot{checkQA} \implies ((TC_1 \wedge AC_3) \vee (TC_2 \wedge AC_1 \wedge AC_2))$
R2) $\dot{s_{fix}} \implies QAFailed$
R3) $\dot{done} \implies QAChecked$
R4) $\dot{s_{A_1}} \implies \neg OngoingA_1$

Fig. 2. Production constraints described in FLTL.

Note that the formulas described in Figure 2 are combinations of the fluents described in Figure 3 and fluents $\dot{\ell}$ induced by an action ℓ. Fluents TC_i and AC_i are described with the index i for brevity. TC_i denotes the product type choice while AC_i denotes that the activity A_i has finished. $QAChecked$ holds, when a product satisfies the quality requirements. $QAFailed$ holds when the quality check fails but the product is not repaired yet. $OngoingA_1$ is a fluent that holds when A_1 has started (s_{A_1}) but not ended yet (e_{A_1}).

a) $TC_i = \langle \{type_i\}, \{done\}, false \rangle$
b) $AC_i = \langle \{e_{A_i}\}, \{done\}, false \rangle$
c) $QAChecked = \langle \{okQA, e_{fix}\}, \{type_1, type_2\}, false \rangle$
d) $QAFailed = \langle \{failQA\}, \{e_{fix}\}, false \rangle$
e) $OngoingA_1 = \langle \{s_{A_1}\}, \{e_{A_1}\}, false \rangle$

Fig. 3. Fluent definitions used to describe the formulas of Figure 2.

The synthesis problem is to build a controller that satisfies production requirements by controlling processing activities, while the choice of product type and the result of the quality check are unknown in advance. Various solutions to this problem exist, and current synthesis techniques are likely to yield the controller C_2 of Figure 4; whereas the controller C_1 in the same figure is another valid solution.

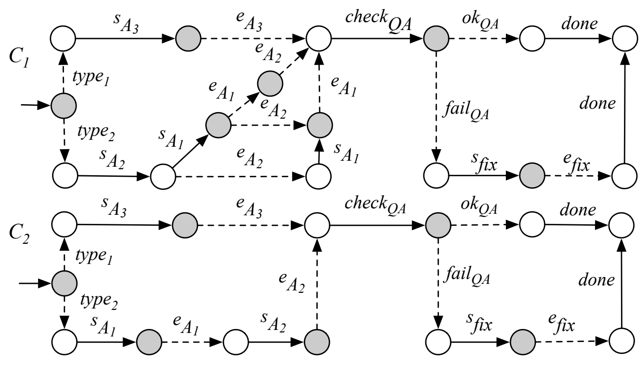


Fig. 4. Two possible controllers for the same problem.[1,2]

Regarding makespan, C_1 seems to be better than C_2. It is because, for products of type T_2, C_1 will always run the two activities concurrently, while C_2 will perform them sequentially. Intuitively, C_1 will perform as well as or better than C_2 for any fixed choice of the environment (the type of product to be built and the result of the quality check).

V. Symbolic Comparison of Controllers

To compare the makespan of controllers, we compute a symbolic expression that describes the time a controller takes to reach its goal. The symbolic expression here is composed

[1] $\dashrightarrow$ denotes uncontrollable actions, and $\rightarrow$ denotes controllable actions.

[2] Grey states denote non-transient states.

of parameters which represent the time each activity takes. A particular strategy for the environment must be fixed to allow fine-grained comparison between controllers' strategies. That means how uncontrolled contingencies pan out. In this Section, we first formalise how to deal with contingencies by using schedulers (Section V-A). Then, we give a timed semantics to LTSs based on PTA (Section V-B) and show how to obtain a symbolic expression that represents the makespan of a controller using that PTA (Section V-C). The parameters of the PTA stand for activity duration as well as a parameter representing the end-to-end makespan. By using parameters, we model that time durations are unknown a priori. Finally, we define how to compare a pair of controllers by comparing their symbolic expressions (Section V-D).

A. Comparing under the Same Contingencies

When comparing controllers, we should pay attention to how uncontrollable behaviour may occur. For instance, let us consider the following unfair comparison between the two controllers mentioned in Section IV. On the one hand, controller C_1 requires the final repair while constructing a product of type T_1. On the other hand, controller C_2 does not require the repair while building a product of type T_2. This comparison could suggest that controller C_2 has a lower makespan, but the conclusion is obviously flawed because the contingencies of which type of product is built and whether a repair is required or not do not depend on the controller itself and should not affect the comparison. Therefore, contingency scenarios should be consistent when a comparison is performed.

We use schedulers [Bellman, 1957] to formalise how to resolve contingencies at a given state of the environment. We assume that schedulers are Markovian [Bellman, 1957] and yield a decision only based on the current environment state regardless of the history of states traversed. Thus, we define schedulers as a function $\sigma : Q_E \times 2^{\Sigma} \rightarrow 2^{\Sigma}$ that picks a subset of enabled actions A (K1), which stand for the actions that might be enabled by an arbitrary controller (Definition 4). The conditions defined aim to model schedulers that, we believe, ensure a fair comparison. This definition distinguishes actions into three categories: controllable, ending, and other uncontrollable actions.

Definition 4 (Scheduler): Given an environment E and a set of activity definitions $\mathcal{AD}$, a *scheduler* is a function $\sigma : Q_E \times 2^{\Sigma} \rightarrow 2^{\Sigma}$ that satisfies the following conditions:

K1) $\sigma(q_E, A) \subseteq A \cap \Delta_E(q_E)$

K2) $(|\sigma(q_E, A)| = 1 \wedge \sigma(q_E, A) \subseteq \Sigma_c \cup (\Sigma_u \setminus \Sigma_e)) \vee (\sigma(q_E, A) \cap \Sigma_e = \sigma(q_E, A))$

K3) $(\sigma(q_E, A) \subseteq \Sigma_u) \Rightarrow \forall A' (\sigma(q_E, A') = \sigma(q_E, A))$

K4) $\sigma(q_E, A) \subseteq \Sigma_c \Rightarrow \forall A' \subseteq \Delta_E(q_E)$
$((\sigma(q_E, A) \subseteq A' \Rightarrow \sigma(q_E, A') = \sigma(q_E, A)) \wedge$
$(\sigma(q_E, A) \nsubseteq A' \wedge A' \cap \Sigma_c \neq \emptyset \Rightarrow \sigma(q_E, A') \subseteq \Sigma_c))$
$(\sigma(q_E, A) \nsubseteq A' \wedge A' \cap \Sigma_c = \emptyset \Rightarrow \sigma(q_E, A') \subseteq \Sigma_u))$

where $A \subseteq \Delta_E(q_E)$, $q_E \in Q_E$, and $\Sigma_e = \{\ell \mid \ell \in \Sigma_u \wedge \exists \alpha \in \mathcal{A}\ \ell \in Ends(\alpha)\}$.

Above conditions require schedulers to either pick all ending actions or exactly one of the other actions (K2). Since the duration of the activities is unknown, we do not want schedulers to choose among the ending actions. Besides, schedulers have to be consistent, regarding the choices of uncontrollable (K3) and controllable actions (K4). That is, if possible, they pick the same actions or category.

Schedulers are applied to a controller when executing in parallel with the environment $E\|_{\sigma}C$ (Definition 5). This produces an LTS $E\|_{\sigma}C$ that has a subset of the transitions defined by the standard parallel composition $E\|C$.

Definition 5 (Scheduled Composition): The *scheduled composition* $E\|_{\sigma}C$ is an asymmetric operator defined as the parallel composition ($\|$) changing the shared rule of $\Delta_{E\|_{\sigma}C}$ as follows:

$$\frac{q_E \xrightarrow{\ell}_E q'_E,\ q_C \xrightarrow{\ell}_C q'_C,\ \ell \in \sigma(q_{E_1}, \Delta_C(q_{C_1}))}{(q_E, q_C) \xrightarrow{\ell}_{M\|_{\sigma}N} (q'_E, q'_C)}\ \ell \in \Sigma_E \cap \Sigma_C$$

B. Timed Semantics of LTS

Our interpretation of time in a discrete controller is based on PTA [Alur et al., 1993]. PTA is an extension of Timed Automata (TA) [Baier and Katoen, 2008] that incorporates final states, clocks and parameters to LTSs. Clocks are real-valued variables that increase linearly, and parameters are unknown constants. Clocks can be compared with constant values or parameters in guards and invariants, which are sets of linear inequalities that must be satisfied to take a transition or to stay in a state. These clocks can be reset when taking a transition. In this work, we use PTA to interpret how time passes on a controller, when the controller is concurrently executing with the environment under a given scheduler σ. Definition 6 presents the *Timed Semantics* of a controller by using a PTA. This PTA focuses on the paths between the initial state and goal states, because makespan is measured between these states.

Definition 6 (Timed Semantics): Given a control problem $\mathcal{E} = \langle E, \mathcal{AD}, G, \Sigma_c \rangle$, a controller C that is solution for $\mathcal{E}$, a scheduler σ for the environment E and goal states Q_G in $E\|_{\sigma}C$, we define the *timed semantics* of $E\|_{\sigma}C$ as a parametric timed automaton $PTA(E\|_{\sigma}C) = (\Sigma', Q', Q'_0, X, P, Q_f, I, \Theta)$ as follows:

- a set of actions $\Sigma' = \Sigma \cup \{s_f, e_f\}$
- a set of states $Q' = Q_{E\|_{\sigma}C} \cup \{{q_0}', q'_f\}$
- a set of initial states $Q'_0 = \{{q_0}'\}$
- a set of clocks $X = \{x_\alpha \mid \alpha \in \mathcal{A}\} \cup \{x_{q_E} \mid q_E \in Q_E\} \cup \{x_u, x_f\}$
- a set of parameters $P = \{p_\alpha \mid \alpha \in \mathcal{A}\} \cup \{p_{q_E} \mid q_E \in Q_E\} \cup \{p_f\}$
- a set of final states $Q_f = \{q'_f\}$
- a state invariant I defined as

$$I(q) = \begin{cases} x_u \leq 0 & \text{if } q_C \text{ is transient} \\ \bigwedge_{\alpha \in \zeta(q)} x_\alpha \leq p_\alpha & \text{if } q_C \text{ is not transient} \\ & \text{and } \zeta(q) \neq \emptyset \\ x_{q_E} \leq p_{q_E} & \text{otherwise} \end{cases}$$

  for every state $q = (q_E, q_C) \in Q_{E\|_{\sigma}C}$ and $I({q_0}') = I(q'_f) = x_u \leq 0$
- a set of edges Θ s.t. $(q_1, \ell, q_2, \lambda, \mu) \in \Theta$ iff either
  - $(q_1 = (q_{E_1}, q_{C_1}), \ell, q_2 = (q_{E_2}, q_{C_2})) \in \Delta_{E\|_{\sigma}C}$ and

((q_{C_1} is transient and has guard

$$\mu = \begin{cases} x_u = 0 \wedge x_\alpha = p_\alpha & \text{if } \exists \alpha \in \mathcal{A}\ \ell \in Ends(\alpha) \\ x_u = 0 & \text{otherwise} \end{cases}$$

) or

(q_{C_1} is not transient and has guard

$$\mu = \begin{cases} x_\alpha = p_\alpha & \text{if } \exists \alpha \in \mathcal{A}\ \ell \in Ends(\alpha) \\ x_{q_{E_1}} = p_{q_{E_1}} & \text{if } \zeta(s_1) = \emptyset \end{cases}$$

)) and reset clocks

$$\lambda = \{x_u\} \cup \{x_{q_{E_2}}\} \cup \{x_\alpha | \exists \alpha \in \mathcal{A}\ \ell = Start(\alpha)\}$$

- $q_1 \in Q_G$ and q_2= q_f' and $\ell = e_f$ and has guard $\mu = (x_f = p_f)$ and reset clocks $\lambda = \{x_u\}$
- $q_1 = {q_0}'$ and $q_2 = q_{(E\|C)^{\downarrow}_{\sigma 0}}$ and $\ell = s_f$ and has guard $\mu = (x_u = 0)$ and reset clocks $\lambda = \{x_u, x_f\}$

The defined PTA features a clock x_α and parameter p_α for each activity α. Each clock x_α measures the time elapsed from the start of the activity α to the end of it. The transitions that are start of activity reset the clock, and the transitions that are end of activity have a guard. This guard denotes that the transition can be taken, when the time elapsed by the clock of the activity is equal to its corresponding parameter ($p_\alpha = x_\alpha$). When the activity is being processed, there are invariants that restrict the clock of the activity to be greater than its parameter ($x_\alpha \leq p_\alpha$). Similarly, there is a clock x_{q_E} and parameter p_{q_E} for each state (q_E, q_C), in which all the outgoing transitions are uncontrollable actions and no activities are running. The clock x_{q_E} measures the time spent at (q_E, q_C), and p_{q_E} stands for the total sojourn time. Finally, another clock x_f and parameter p_f are added to measure the end-to-end makespan. This clock works as an envelope that subsumes all the other clocks. That is to say, all the other parameters that appear in at least one path of the PTA are bounded by p_f. Besides, transient states, i.e., states that have controllable actions enabled, are forced to be abandoned in zero time through an invariant ($x_u < 0$). Note that "q_C is transient" refers to the enabled actions in C without applying the scheduler ($\Delta^c_C(q_C) \neq \emptyset$). The final states are the goal states that appear in the scheduled composition.

In Figure 5, we show the PTA of the controller C_1 and C_2 (in Figure 4) composed in parallel with the environment E under a scheduler σ that, amongst other things, determines the product type to be built and that repair is not needed. Note that the clocks x_{q_1} and parameters p_{q_1} refer to a state of the environment $q_1 \in Q_E$, which is the initial state of the composition E of the LTSs from Figure 1. The other state q_4 corresponds to the state of the composition in which the LTS QA is in the state 1, LTS $Choice$ is in the state 2 and the other LTSs are in the initial state 0. We use the environment state as id to represent that two controllers have the same waiting time in the same state of the environment.

C. Consistent Parameters as a Makespan Metric

The parameters in PTA can be instantiated into values. A parameter valuation for P is an assignment of values in T (domain of time values) to the parameters in P. Given

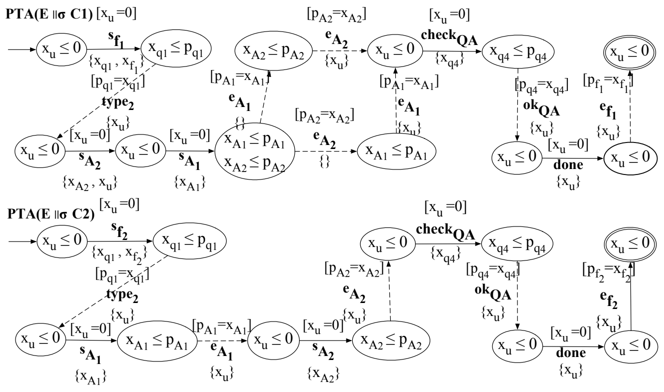


Fig. 5. PTAs of the controllers and environment defined in Section IV under a scheduler. Doubly circled states denote final states, invariants are defined within the states, and transitions are composed by an action ℓ in bold, a guard μ=[condition], and reset clocks $\lambda = \{x_1, \ldots, x_k\}$.

a parameter valuation and a PTA, we can obtain a Timed Automaton (TA) with domain T by replacing the parameters with the constant values. We use $\mathbb{R}_{\geq 0}$ as our time domain T. However, some parameter valuations might fix the conditions and invariants with values that do not allow any path to reach a final state from the initial state in the obtained TA. A parameter valuation is consistent with a PTA, if there exists at least one path that reaches a final state in the TA obtained from replacing the parameters with the constant values (i.e., the TA has a non-empty language). The set of parameter valuations that is consistent with a PTA H is known in the literature as $\Gamma(H)$ [Alur et al., 1993]. Γ is a symbolic expression that defines the conditions which a parameter valuation has to satisfy to be consistent with a PTA. We denote a parameter valuation, i.e., an assignment of real values $(r_1, \ldots, r_n) \in \mathbb{R}^{|P|}_{\geq 0}$ to the parameters $(p_1, \ldots, p_n) \in P$, that is consistent with a PTA H as $(r_1, \ldots, r_n) \Vdash H$.

In general, computing Γ is undecidable, if there are more than three clocks in PTAs with cycles [Alur et al., 1993]. However, by using a symbolic procedure [Henzinger, 2000], Γ can be computed for the fragment of PTAs related to reachability controllers. This procedure works symbolically through computing a fixed point of a precondition operator. Given that the PTAs generated are acyclic, the procedure terminates, and the fixpoint computation boils down into a one-pass backwards propagation of expressions. These expressions are changed at each edge by the precondition operator. Moreover, for the particular case of the PTAs from Definition 6, Algorithm 1 shows how to calculate Γ by using the precondition operator shown in Definition 7. The algorithm first initialises the formula ψ_{q_f} of the final state with $True$ (line 3), and it traverses the other states in the inverse topological order while initialising the value of the formula in each state (line 4 - 5). The result of $\Gamma(H)$ is the formula of the initial state ψ_{q_0} after resetting the clocks, i.e. replacing all the occurrences by 0. The precondition operator uses two sub-operators: transition-step $\triangleright$ and time-step $\Rightarrow$. The transition-step $\triangleright$ adds the guard μ to the formula and resets the clocks $x \in \lambda$ in ψ. The time-step $\Rightarrow$ adds the state invariant to the formula, but in the non-transient case it uses a new variable δ and an existential quantifier to represent the possible passage of time. The passage of time is done by replacing all the appearances of the clocks $x \in X$ in the formula ψ and state invariant $I(q)$ with $x + \delta$. Note that the new variable δ will not appear in the formula produced by pre, because the algorithm applies quantifier elimination in each step to remove the existential quantifier.

Notation 1: We denote the edges from state q as $\Theta(q) = \{(q_1, \ell, q_2, \lambda, \mu) \mid (q_1, \ell, q_2, \lambda, \mu) \in \Theta \wedge q = q_1\}$.

Algorithm 1 Obtaining Γ of PTA from Definition 6.

```
1: procedure Γ($H$)
2:     $[q_0, \dots, q_n, q_f] = $ TOPOLOGICAL_SORT($H, Q_f$)
3:     $\psi_{q_f} = True$[3]
4:     for $q \in [q_n, \dots, q_0]$ do
5:         $\psi_q = \bigvee_{\varepsilon \in \Theta(q)} pre_\varepsilon(\psi_{q'})$
       return $\psi_{q_0}[\forall x \in X (x := 0)]$[3]
```

Definition 7 (Precondition Operator): Given an edge $\varepsilon = (q_1, \ell, q_2, \lambda, \mu) \in \Theta$ and the linear formula ψ corresponding to the propagated conditions of the state q_2, the *precondition operator pre_ε* is defined as follows:

$$pre_\varepsilon(\psi) = (\Rightarrow (q_1, \rhd(\varepsilon, \Rightarrow (q_2, \psi))))$$
$$\rhd(\varepsilon, \psi) = (\mu \wedge \psi[\forall x \in \lambda \cdot x := 0])$$
$$\Rightarrow (q, \psi) = \begin{cases} \psi \wedge I(q) & \text{if } q \text{ is transient} \\ \exists \delta \geq 0 \ / & \text{otherwise} \\ (\psi \wedge I(q))[\forall x \in X (x := x + \delta)] & \end{cases}$$

For the PTAs in Definition 6, Γ of the built PTA is a time constraint of the end-to-end makespan parameter as well as the parameters of the activities and environment states. Γ is the relation among parameters that defines the end-to-end makespan as a linear expression of time durations by using the parameters. For instance, for the PTAs in Figure 5, $\Gamma(PTA(E\|_\sigma C_1)) = ((p_{A_2} \geq p_{A_1} \wedge p_{f_1} = p_{A_2} + p_{q_1} + p_{q_4}) \vee (p_{A_1} \geq p_{A_2} \wedge p_{f_1} = p_{A_1} + p_{q_1} + p_{q_4}))$ whereas $\Gamma(PTA(E\|_\sigma C_2)) = (p_{f_2} = p_{A_1} + p_{A_2} + p_{q_1} + p_{q_4})$ are simplified versions of the symbolic expression that represent their makespan. The expression $\Gamma(PTA(E\|_\sigma C_1))$ states that, when the duration of activity α_1 is greater than that of α_2, the makespan (p_{f_1}) is the sum of durations of activity α_1 plus the time spent in the states q_1 and q_4; otherwise, the makespan is the duration of activity α_2 plus the time spent in the states q_1 and q_4. In contrast, $\Gamma(PTA(E\|_\sigma C_1))$ is the sum of the duration of the two activities plus the time spent in the states. Thus, no matter which values are assigned to the parameters, p_{f_1} cannot be greater than p_{f_2}. Even though here it is easy to see, checking the existence of these parameters will require the use of an SMT-solver.

D. Comparing Symbolic Expressions

Here we proceed to discuss how to compare controllers by using Γ. Controller C_1 has a behaviour of higher makespan than C_2 (noted $C_2 \lhd C_1$), if there exists a parameter valuation and a scheduler such that the makespan of C_1 is strictly higher than that of C_2. That is to say, C_1 has a behaviour of higher makespan than C_2 iff $\exists \sigma \ \exists (r_1..r_n, r_{f1}, r_{f2}) \in \mathbb{R}^{n+2}_{\geq 0} \ ((r_1..r_n, r_{f1}) \Vdash \Gamma(PTA(E\|_\sigma C_1)) \wedge (r_1..r_n, r_{f2}) \Vdash \Gamma(PTA(E\|_\sigma C_2)) \wedge r_{f1} > r_{f2})$. It means that there exists at least one case, in which for the same scheduler σ and assignment $(r_1, \dots, r_n)$ to the parameters $(p_1 \dots p_n)$, C_1 performs worse than C_2 ($r_{f1} > r_{f2}$). These parameters stand for activity durations and environment states that are common to the two controllers. The values r_{f1} and r_{f2} are assigned to the end-to-end makespan parameters p_{f1} and p_{f2} of the controllers C_1 and C_2 respectively. These two values are conditioned by the values $(r_1, \dots, r_n)$. $C_2 \lhd C_1$ can be resolved by using SMT-solvers [Barrett and Tinelli, 2018].

Besides, in order to compare two controllers, we perform the $\lhd$ comparison in both ways, $C_1 \lhd C_2$ and $C_2 \lhd C_1$. Table I shows the possible results. Two controllers are: i) *Incomparable* when there are circumstances in which both controllers have a behaviour of higher makespan than the other, ii) *Equivalent* when there is no scheduler that allows one controller to have a behaviour of higher makespan than the other, or iii) one *dominates* the other when one controller does not show a behaviour of higher makespan than the other, while the other shows a behaviour of higher makespan. Note that this comparison takes into account all possible schedulers of an environment.

$C_2 \lhd C_1$	$C_1 \lhd C_2$	Conclusion
Sat	Sat	Incomparable
Unsat	Unsat	Equivalent
Sat	Unsat	C_2 dominates C_1
Unsat	Sat	C_1 dominates C_2

TABLE I

POSSIBLE RESULTS OF COMPARING TWO CONTROLLERS

VI. MAKESPAN-MINIMISING CONTROLLERS

In this work, we define a *makespan-minimising controller* based on the *dominance* relationship defined in Section V-B. A controller C is makespan-minimising if it is a *non-dominated* controller (Definition 8), i.e., there is no other controller C' that *dominates* C.

Definition 8 (Non-dominated): Given a control problem $\mathcal{E} = \langle E, \mathcal{AD}, G, \Sigma_c \rangle$ and the set of solutions $\mathcal{C}$ for $\mathcal{E}$, we say that $C \in \mathcal{C}$ is non-dominated iff $\nexists C' \in \mathcal{C}$ (C' dominates C).

Note that when a control problem is realisable, there is at least one non-dominated controller. W.l.o.g., we restrict to the subset of memoryless solutions [Thomas, 1995], because solutions that have additional memory would also have additional makespan. Since this set is finite, it is not possible for all of them to be dominated by another controller. This is because C *dominates* C' is a transitive and asymmetric relation. Also note that there can be more than one non-dominated controller.

There are algorithms for controller synthesis problems that produce memoryless controllers [Thomas, 1995]. Such algorithms [Maler et al., 1995] have linear complexity in the size of the problem, and they build controllers in the form of directed acyclic graphs (DAG), which are subgraphs of the given problem. These controllers are universal in the sense

[3]These PTAs have only one initial and one final state.

that any other memoryless controller is a subgraph of the universal controller.

Algorithm 2 Non-dominated controller algorithm.

```
1: procedure NON_DOMINATED($E, \mathcal{AD}, G, \Sigma_c$)
2:     $\mathcal{U}$ = UNIVERSAL_SYNTHESIS($E$, $\Sigma_c$, $G$)
3:     $K = E \| \mathcal{U}$
4:     $[q_0, \ldots, q_n]$ = TOPOLOGICAL_SORT($K, Q_G$)
5:     for $q \in [q_n, \ldots, q_0]$ do
6:         $A = \{\{c\} \mid c \in \Delta^c_K(q)\}$
7:         if $\Delta^u_K(q) \neq \emptyset$ then A= A$\cup\{\emptyset\}$
8:         if $|Alt| > 1$ then
9:             $\mathcal{D} = \emptyset$
10:            for $A \in Alt \wedge A' \in Alt \setminus \{A\}$ do
11:                if $K_{\langle q,A' \rangle}$ dominates $K_{\langle q,A \rangle}$ then $\mathcal{D} = \mathcal{D} \cup A$
12:            $\Delta_K = \Delta_K \setminus \{(q, c, q') \mid (q, c, q') \in \Delta_K \wedge c \in \mathcal{D}\}$
    return $K$
```

Algorithm 2 is an algorithm that produces a non-dominated controller. This algorithm first obtains a universal controller. Then, it traverses the states of the universal controller in the inverse topological order. In the states that have more than one controllable action enabled, the algorithm uses the dominance comparison to disable some of the enabled controllable actions. The dominance comparison initially only compares two controllers from the initial state. However, by defining a sub-LTS (Definition 9), the comparison can be made from any states, not limiting to the initial state. It then generates the set of alternatives Alt of a state. An alternative can be one of the controllable actions $c \in \Delta^c(q)$ or $\emptyset$ that represents waiting for uncontrollable events to happen. It compares alternative $A \in Alt$ against all the other alternatives $A'_1 \ldots A'_n \in Alt$, by comparing sub-LTS $M_{\langle q,A \rangle}$ against sub-LTSs $M_{\langle q,A'_1 \rangle} \ldots M_{\langle q,A'_n \rangle}$. If any of the other alternatives in $M_{\langle q,A'_1 \rangle} \ldots M_{\langle q,A'_n \rangle}$ dominates $M_{\langle q,A \rangle}$, it removes the controllable transition of alternative A from Δ_K.

Definition 9 (sub-LTS): Given an LTS $M = (Q, \Sigma, \Delta, q_0)$, a state $q \in Q$ and $A \subseteq \Delta^c(q)$, we define $M_{\langle q,A \rangle} = (Q, \Sigma, \Delta', q)$ as a *sub-LTS* of M, whose initial state is q, and the transition function as $\Delta' = \Delta \setminus \{(q, \ell', q') \mid (q, \ell', q') \in \Delta^c(q) \wedge \ell' \notin A\}$.

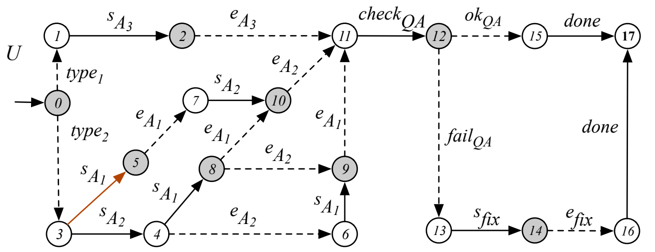


Fig. 6. Universal controller for the example of Section IV. The red transition is the transition removed by the Algorithm 2.

Figure 6 shows the universal controller of the example presented in Section IV. Note that $K = E \| \mathcal{U}$ is structurally equivalent to $\mathcal{U}$. In this example, the algorithm traverses the states in the order specified in the states of the figure. Here, the states with more than one alternative are only two. One is the state 4 which has alternatives $\emptyset$ and $\{s_{A_1}\}$. Since the alternative of waiting for e_{A_2} does not dominate the one that enables the action s_{A_1} (i.e. $K_{\langle q_3, \{s_{A_1}\} \rangle}$ *dominates* $K_{\langle q_3, \emptyset \rangle}$), the algorithm does not remove any controllable transition here. The other one is the state 3. Here, the alternative $\{s_{A_1}\}$ is dominated by the alternative $\{s_{A_2}\}$ ($K_{\langle q_4, \{s_{A_2}\} \rangle}$ *dominates* $K_{\langle q_4, \{s_{A_1}\} \rangle}$). Then, the algorithm removes the transition (q_3, s_{A_1}, q_5) from the universal controller. Finally, the resulting controller would be equivalent to C_1 of Figure 4, if we consider only the states that are reachable from the initial state.

The following theorem shows that the Algorithm 2 produces a controller K that is non-dominated.

Theorem 1 (K is non-dominated): Given an activity control problem $\mathcal{E} = \langle E, \mathcal{AD}, G, \Sigma_c \rangle$, a K = NON_DOMINATED($E, \mathcal{AD}, G, \Sigma_c$), and the set of solutions $\mathcal{C}$ for $\mathcal{E}$, K is a controller and $\nexists C \in \mathcal{C}$ (C dominates K).

Proof Sketch: i) The algorithm produces a controller. This comes from the fact that it is not possible to have a circular relationship of dominance. Thus, the algorithm can never remove all the transitions of a state. ii) The non-dominance can be proved by induction on the topological order. The inductive step requires reasoning on properties which relate valuations satisfying Γ in the sub-LTS of a state to valuations satisfying Γ of successor sub-LTSs. Universality of $\mathcal{U}$ is also required to prove that all memoryless controllers are considered. Also note that all non-dominated memoryless controllers are included in the result yielded by the algorithm. ■

VII. Related Work

Quantitative planning and control to minimise or maximise payoff functions (that can include makespan) have been studied extensively, mostly based on Markov Decision Processes and similar formalisms [Thrun et al., 2005, Bloem et al., 2009, Chatterjee et al., 2005]. Two different controllers can exhibit very different behaviours, depending on how the environment interacts with them. The problem of comparing two controllers (or contingent plans) can be solved by minimising the expected value of the payoff function, whereas computing an expected value requires a probability distribution for different alternative paths. Optimal control for quantitative discrete-event systems has also been studied in the supervisory control community [Brave and Heymann, 1993, Passino and Antsaklis, 1989]. In those works, the performance of the controller is measured by introducing cost functions, and the goal is to reach a set of desired states while optimising the cost. Some works focus on synthesising an optimal supervisor that minimises the makespan [Su et al., 2012]. Typically, to compare solutions, quantitative techniques must aggregate makespan of different runs by using worst or average case. In contrast, our approach does not require probabilistic modelling or quantitative information about the environment. Instead, we establish a way to compare controllers qualitatively by using a symbolic comparison that fixes the uncontrolled behaviour of the environment. This form of comparison, we believe, is a contribution.

Time is typically considered as a quality measure of a solution that involves concurrency. In fact, the main goal of temporal planning problems is to try to minimise the makespan in reaching a deadline [Cushing et al., 2007] while satisfying a set of quantitative temporal constraints about the order in which activities have to be executed. Recently, there has been strong interest in the uncertainty of durations in this community, i.e., achieving the goal for any possible value of uncontrollable durations [Cimatti et al., 2015]. In the work of Cimatti et al., the uncertain durations of the activities are bounded between concrete values $[\delta_{min}, \delta_{max}]$. They also use SMT to model time constraints and solve a temporal plan, but the problems involved have no contingencies. In contrast, we solve a reactive problem against an adversarial environment, and the symbolic durations are used to express preferences among solutions.

Parametric timed models have been studied since the seminal work of [Alur et al., 1993]. Parameters are the basis to study robust schedulability conditions [Cimatti et al., 2008, Saksena, 1994]. The effect of parameters is studied in generalisations of shortest and critical paths [Karp and Orlin, 1981]. In our work, parametric systems are the basis of comparing different controllers.

VIII. Conclusions

The main contribution of this work is building a framework to symbolically compare makespan of controllers for safety and reachability goals. In the comparison, controllers are compared under the same contingencies. Moreover, we define a makespan-minimising controller as a controller that cannot be dominated by any other controller. We also present a sound algorithm to produce a makespan-minimising controller.

Furthermore, even though we currently make no assumption on possible relations among activity durations, this technique can be easily extended to support symbolic constraints of activity durations (e.g., $p_{\alpha_1} \leq p_{\alpha_2} + p_{\alpha_3}$). These constraints can be added into the comparison framework, which would also be reflected in the algorithm. Although in this paper we focus on reachability goals and makespan analysis, we plan to extend the work to support more general goals, such as GR(1) [Piterman et al., 2006]. However, this has a key challenge which is to deal with cycles, which are product of uncontrollable behaviour that can be executed an unbounded number of times.

References

[Alur et al., 1993] Alur, R., Henzinger, T. A., and Vardi, M. Y. (1993). Parametric real-time reasoning. In *Proc. of the ACM Symp. on Theory of Computing*, STOC '93.

[Baier and Katoen, 2008] Baier, C. and Katoen, J.-P. (2008). *Principles of model checking*. MIT press.

[Barrett and Tinelli, 2018] Barrett, C. and Tinelli, C. (2018). Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer.

[Bellman, 1957] Bellman, R. (1957). A Markovian decision process. *Journal of Mathematics and Mechanics.*, 6:679–684.

[Bloem et al., 2009] Bloem, R., Chatterjee, K., Henzinger, T. A., and Jobstmann, B. (2009). Better quality in synthesis through quantitative objectives. In *Computer Aided Verification*, pages 140–156. Springer.

[Brave and Heymann, 1993] Brave, Y. and Heymann, M. (1993). On optimal attraction in discrete-event processes. *Information sciences*, 67(3):245–276.

[Chatterjee et al., 2005] Chatterjee, K., Henzinger, T. A., and Jurdzinski, M. (2005). Mean-payoff parity games. In *LICS*, pages 178–187. IEEE Computer Society.

[Cimatti et al., 2008] Cimatti, A., , Palopoli, L., and Ramadian, Y. (2008). Symbolic computation of schedulability regions using parametric timed automata. In *Real-Time Systems Symposium*, RTSS'08, pages 80–89. IEEE Press.

[Cimatti et al., 2015] Cimatti, A., Micheli, A., and Roveri, M. (2015). Strong temporal planning with uncontrollable durations: A state-space approach. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 3254–3260.

[Cimatti et al., 2003] Cimatti, A., Pistore, M., Roveri, M., and Traverso, P. (2003). Weak, strong, and strong cyclic planning via symbolic model checking. *Artificial Intelligence*, 147(1-2):35–84.

[Cushing et al., 2007] Cushing, W., Kambhampati, S., Mausam, and Weld, D. S. (2007). When is temporal planning really temporal? In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 1852–1859.

[de Alfaro and Henzinger, 2001] de Alfaro, L. and Henzinger, T. A. (2001). Interface automata. In *ESEC / SIGSOFT FSE*, pages 109–120. ACM.

[Giannakopoulou and Magee, 2003] Giannakopoulou, D. and Magee, J. (2003). Fluent Model Checking for Event-Based Systems. In *ESEC/FSE*, pages 257–266, Helsinki, Finland.

[Goldman and Boddy, 1996] Goldman, R. P. and Boddy, M. S. (1996). Expressive planning and explicit knowledge. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems, Edinburgh, Scotland, May 29-31, 1996*, pages 110–117.

[Henzinger, 2000] Henzinger, T. A. (2000). *The theory of hybrid automata*. Springer.

[Karp and Orlin, 1981] Karp, R. M. and Orlin, J. B. (1981). Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Applied Mathematics*, 3(1):37–45.

[Letier and Heaven, 2013] Letier, E. and Heaven, W. (2013). Requirements modelling by synthesis of deontic input-output automata. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 592–601. IEEE Press.

[Maler et al., 1995] Maler, O., Pnueli, A., and Sifakis, J. (1995). On the synthesis of discrete controllers for timed systems (an extended abstract). In *STACS*, pages 229–242.

[Passino and Antsaklis, 1989] Passino, K. and Antsaklis, P. (1989). On the optimal control of discrete event systems. In *Proceedings of the 28th IEEE Conference on Decision and Control,*, pages 2713–2718. IEEE.

[Piterman et al., 2006] Piterman, N., Pnueli, A., and Saar, Y. (2006). Synthesis of reactive (1) designs. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer.

[Pnueli and Rosner, 1989] Pnueli, A. and Rosner, R. (1989). On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–190. ACM.

[Pryor and Collins, 1996] Pryor, L. and Collins, G. (1996). Planning for contingencies: A decision-based approach. *J. Artif. Intell. Res. (JAIR)*, 4:287–339.

[Ramadge and Wonham, 1984] Ramadge, P. and Wonham, W. (1984). Supervisory control of a class of discrete event processes. In *Analysis and Optimization of Systems*, pages 475–498. Springer.

[Saksena, 1994] Saksena, M. C. (1994). *Parametric Scheduling for Hard Real-time Systems*. PhD thesis, College Park, MD, USA. UMI Order No. GAX95-14577.

[Su et al., 2012] Su, R., Van Schuppen, J. H., and Rooda, J. E. (2012). The synthesis of time optimal supervisors by using heaps-of-pieces. *IEEE Transactions on Automatic Control*, 57(1):105–118.

[Thomas, 1995] Thomas, W. (1995). On the synthesis of strategies in infinite games. In *STACS*, pages 1–13.

[Thrun et al., 2005] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

Appendix

Lemma 1.1 (Stuttering: redundancy of the step operator in the precondition operator): Given a two pair of edges of a PTA like the ones shown in Figure 8, $pre_{\varepsilon_1}(pre_{\varepsilon_2}(\Psi)) \implies (q_1, \rhd(\varepsilon_1, pre_{\varepsilon_2}(\Psi)))$.

Proof Sketch: The intuition is that the internal transition step of the state q_2 was already applied when applying the precondition operator for the first time $pre_{\varepsilon_2}(\Psi))$. This can be extended to many edges in the state q_2 as in the Figure 7

$pre_{\varepsilon_1}(pre_{\varepsilon_2}(\Psi)) \implies (q_1, \rhd(\varepsilon_2, \underline{\Rightarrow (q_2, \Rightarrow (q_2}, \rhd(\varepsilon_2, \Rightarrow (q_3, \Psi))))))$ By using Definition 7, we can show that is redundant to apply the $\Rightarrow$ twice consecutively. ■

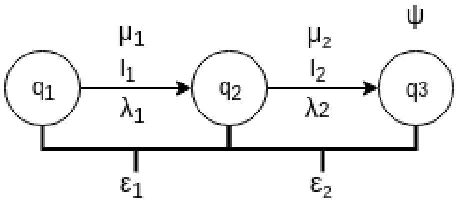


Fig. 7. Two consecutive edges of a PTA. Ψ is the partial value of the state q_3 when applying the precondition operator.

NOTE: We can always use Lemma 1.1 when comparing PTAs because the last transition is to the dummy final state.

Lemma 1.2 (Precondition of controllable edges are Γ of its successor): Given a state q_1 with a single outgoing edge $\varepsilon = (q_1, \ell, q_2, \lambda, \mu)$ of a PTA of Definition 6 such that $q_2 \notin Q_f$ and $\ell \in \Sigma_c$, $\Gamma(H_{\langle q_1 \rangle}) = \Gamma(H_{\langle q_2 \rangle}) \wedge \mu'[\forall x \in X\ x := 0]$.

Proof Sketch: Because q_1 is transient, the precondition operator can be expressed as follows: $pre_{\varepsilon_1}(\Psi) = I(q_1) \wedge (\mu_1 \wedge \Psi[\forall x \in \lambda_1 x := 0])$ (Lemma 1.1 and Definition 7).

In particular, transient states of PTAs of Definition 6 have an urgent invariant: $pre_{\varepsilon_1}(\Psi) = (x_u \leq 0) \wedge (\mu_1 \wedge \Psi[\forall x \in \lambda_1 x := 0])$

Then, $\Gamma(H_{\langle q_1 \rangle}) = pre_{\varepsilon_1}(\Psi)[\forall x \in X\ x := 0] = ((x_u \leq 0) \wedge (\mu_1 \wedge \Psi[\forall x \in \lambda_1 x := 0]))[\forall x \in X\ x := 0] = (\mu_1 \wedge \Psi)[\forall x \in X\ x := 0] = \Gamma(H_{\langle q_1 \rangle}) \wedge \mu_1[\forall x \in X\ x := 0]$.

In particular, when $\ell_1 \in \Sigma_c$, $\Gamma(H_{\langle q_1 \rangle}) = \Gamma(H_{\langle q_2 \rangle})$ because $\mu_1[\forall x \in X\ x := 0] = (x_u == 0)[\forall x \in X\ x := 0] = True$ ■

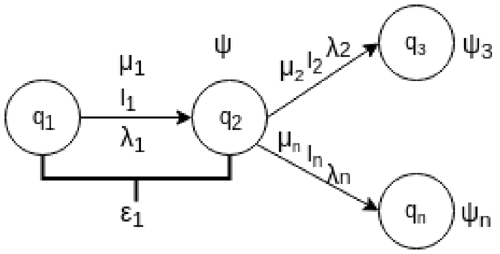


Fig. 8. Two consecutive edges of a PTA. Ψ is the partial value of the state q_3 when applying the precondition operator.

Lemma 1.3 (Transitivity): Given controllers C_1, C_2, C_3 s.t. C_1 *dominates* C_2 and C_2 *dominates* C_3 or C_2 *as good as* C_3, then C_1 *dominates* C_3.

Proof Sketch: It can be prove directly using $\forall\sigma\forall\gamma \ldots p_1 \leq p_2 \wedge p_2 = p_3$ and $\exists\sigma\exists\gamma \ldots p_1 > p_2 \wedge p_2 = p_3$ ■

TODO: Theorem 2 should actually first prove that there is no point in comparing against controllers with memory. That is the reason why we focus on memoryless controllers that are subset of a universal controller.

Theorem 2 (Sub-controllers of Algorithm 2 are non-dominated): Given a controller K produced by Algorithm 2, $\forall q \in Q\ \nexists C \in \mathcal{C}\ s.t.\ C_{\langle q \rangle}\ dominates\ K_{\langle q \rangle}$, where Q are states of the universal controller and $\mathcal{C}$ is the set of memoryless solutions.

Proof Sketch: We will prove it by using induction on the topological order of the states of the universal controller.

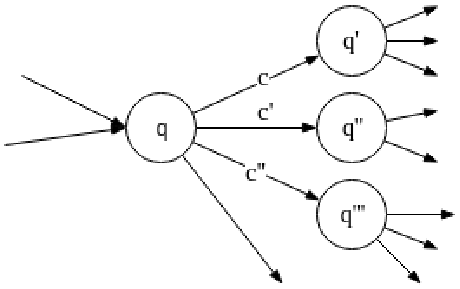
Basis case: Final States) There are no transitions available. No other sub-controllers to compare with.

Inductive Hypothesis) $\forall q \in Q_{[i+1..n]}\ \nexists C \in \mathcal{C}\ s.t.\ C_{\langle q \rangle}\ dominates\ K_{\langle q \rangle}$

Inductive Step) We will prove the induction step using contradiction.

Lets assume that there is a controller C *s.t.* $C_{\langle q\rangle}$ *dominates* $K_{\langle q\rangle}$.
First, we divide between the transient and non-transient case.

- Non-transient: There is no controllable choice. The scheduler always enables the same transitions in the state q. To dominate at the state q it should also dominate in its successors. But this contradicts the inductive hypothesis.
- Transient states:
  The following figure is a local view to help to understand the naming conventions.



  $\exists\sigma\exists\gamma$ s.t. $\gamma \models \Gamma(K_{\langle q\rangle},\sigma) \wedge \gamma \models \Gamma(C_{\langle q\rangle},\sigma) \wedge p_K > p_C$
  By using Lemma 1.2, Γ can be expressed as $\Gamma(K_{\langle q\rangle}) = \bigvee_{\varepsilon'=(q,\ell,q',\lambda',\mu')\in\Theta_K(q)}(\Gamma(K_{\langle q'\rangle}) \wedge \mu'[\forall x \in X\ x := 0])$
  Then, $\exists\sigma\exists\gamma$ s.t.
  $\gamma \models (\bigvee_{\varepsilon'\in\Theta_K(q)}(\Gamma(K_{\langle q'\rangle},\sigma) \wedge \mu'[\forall x \in X\ x := 0])\ \wedge\gamma \models (\bigvee_{\varepsilon''\in\Theta_C(q)}(\Gamma(C_{\langle q''\rangle},\sigma) \wedge \mu''[\forall x \in X\ x := 0])\ \wedge p_K > p_C$.
  Also, $\forall\sigma\forall\gamma$ s.t.
  $\gamma \models (\bigvee_{\varepsilon'\in\Theta_K(q)}(\Gamma(K_{\langle q'\rangle},\sigma) \wedge \mu'[\forall x \in X\ x := 0])\ \wedge\gamma \models (\bigvee_{\varepsilon''\in\Theta_C(q)}(\Gamma(C_{\langle q''\rangle},\sigma) \wedge \mu''[\forall x \in X\ x := 0])\ \wedge p_K \geq p_C$.
  Lets consider σ be a scheduler that shows that $p_K > p_C$.
  - If $\sigma(q,\Delta_K(q)) = \sigma(q,\Delta_C(q) = Z$,
    $\exists\sigma\exists\gamma$ s.t. $\gamma \models (\bigvee_{\varepsilon'\in Z}(\Gamma(K_{\langle q'\rangle},\sigma) \wedge \mu'[\forall x \in X\ x := 0])\ \wedge\gamma \models (\bigvee_{\varepsilon'\in Z}(\Gamma(C_{\langle q'\rangle},\sigma) \wedge \mu'[\forall x \in X\ x := 0])\ \wedge p_K > p_C$.
    Then, $\exists\sigma\exists\gamma$ s.t. $\gamma \models (\bigvee_{\varepsilon'\in Z}(\Gamma(K_{\langle q'\rangle},\sigma) \wedge \Gamma(C_{\langle q'\rangle},\sigma) \wedge \mu'[\forall x \in X\ x := 0])\ \wedge p_K > p_C$.
    Analogously, $\forall\sigma\forall\gamma$ s.t. $\gamma \models (\bigvee_{\varepsilon'\in Z}(\Gamma(K_{\langle q'\rangle},\sigma) \wedge \Gamma(C_{\langle q'\rangle},\sigma) \wedge \mu'[\forall x \in X\ x := 0])\ \wedge p_K \geq p_C$.
    This implies that $C_{\langle q'\rangle}$ dominates $K_{\langle q'\rangle}$, which contradicts the inductive hypothesis.
  - If $\sigma(q,\Delta_K(q)) \neq \sigma(q,\Delta_C(q)) \implies \sigma(q,\Delta_K(q)) = c \wedge \sigma(q,\Delta_C(q)) = c'$
    By using Lemma 1.2, Γ can be expressed as $\Gamma(K_{\langle q\rangle}) = \Gamma(K_{\langle q'\rangle})$ and $\Gamma(C_{\langle q\rangle}) = \Gamma(C_{\langle q''\rangle})$
    * Lets consider the case $c' \notin \Delta_K$, which implies that the algorithm removed the transition.
      Then, there must be a $K_{\langle q'''\rangle}$ s.t. $K_{\langle q'''\rangle}$ *dominates* $K_{\langle q''\rangle}$ because the transition (q,c',q'') was removed.
      Also, from the definition of scheduler, $C_{\langle q\rangle}$ *dominates* $K_{\langle q\rangle}$ implies that $C_{\langle q''\rangle}$ was compared against all the controllable alternatives in $K_{\langle q\rangle}$, in other words, $C_{\langle q''\rangle}$ vs $K_{\langle q'\rangle}, K_{\langle q'''\rangle}\ldots K_{\langle q_n\rangle}$. Moreover, $C_{\langle q''\rangle}$ must dominate or be equally good against all of them. Then, $C_{\langle q''\rangle}$ *dominates* $K_{\langle q'''\rangle}$ or $K_{\langle q'''\rangle}$ *as good as* $C_{\langle q''\rangle}$. By using transitivity (Lemma 1.3) and $K_{\langle q'''\rangle}$ *dominates* $K_{\langle q''\rangle}$, we can show that $C_{\langle q''\rangle}$ *dominates* $K_{\langle q''\rangle}$, which contradicts the inductive hypothesis.

$$\begin{tikzcd}
C_{<q''>} \arrow[r, "\text{dominates}"] \arrow[d, "\text{dominates or eq. good}"'] & K_{<q'>} \arrow[d, dashed, dash, "?"] \\
K_{<q'''>} \arrow[r, "\text{dominates}"] & K_{<q''>}
\end{tikzcd}$$

    * Lets consider the case $c' \in \Delta_K$.
      Similarly, from the definition of scheduler, $C_{\langle q\rangle}$ *dominates* $K_{\langle q\rangle}$ implies that $C_{\langle q''\rangle}$ was compared against all the controllable alternatives in $K_{\langle q\rangle}$. In particular, $C_{\langle q''\rangle}$ *dominates* $K_{\langle q''\rangle}$ or $K_{\langle q''\rangle}$ *as good as* $C_{\langle q''\rangle}$ has to hold. $C_{\langle q''\rangle}$ *dominates* $K_{\langle q''\rangle}$ contradicts the inductive hypothesis. $K_{\langle q''\rangle}$ *as good as* $C_{\langle q''\rangle}$ and $C_{\langle q''\rangle}$ *dominates* $K_{\langle q'\rangle}$ implies that $K_{\langle q''\rangle}$ *dominates* $K_{\langle q'\rangle}$ (Lemma 1.3) but the transition (q,c,q') was not removed by the algorithm. This is also a contradiction.

∎