Event-Triggered and Time-Triggered Duration Calculus for Model-Free Reinforcement Learning

Kalyani Dole*, Ashutosh Gupta*, John Komp†, Shankaranarayanan Krishna*, Ashutosh Trivedi†
* Department of Computer Science and Engineering, Indian Institute of Technology Bombay, Mumbai, India
Email: {kalyanid,akg,krishnas}@cse.iitb.ac.in
† Department of Computer Science, University of Colorado Boulder, Boulder (CO), USA
Email: {john.komp,ashutosh.trivedi}@colorado.edu

2021 IEEE Real-Time Systems Symposium (RTSS) | 978-1-6654-2802-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/RTSS52674.2021.00031

***Abstract*—Reinforcement Learning (RL) is a sampling based approach to optimization, where learning agents rely on scalar reward signals to discover optimal solutions. The specification of learning objectives as scalar rewards is tedious and error prone, and more so for real-time systems with complex time-critical requirements. This paper advocates the use of Duration Calculus (DC)—a highly expressive real-time logic with duration and length modalities—in expressing the learning objectives in model-free RL for stochastic real-time systems. On the other hand, to model stochastic real-time environments, we consider *probabilistic timed automata* (PTA)—Markov decision processes extended with clock variables—that provide an expressive yet computationally decidable formalism to capture real-time constraints over nondeterministic and probabilistic behaviors.**

The key hurdle in developing a convergent RL algorithm for DC specifications is the undecidability of the synthesis problem for PTA against general DC specifications. Inspired by the dichotomy between event-triggered and time-triggered approaches to the design of real-time systems, we present two variants of DC logic—that we dub *event-triggered duration calculus* (EDC) and *time-triggered duration calculus* (TDC)—and identify their subclasses with appealing theoretical properties. We study the decidability (and exact complexity) of the satisfiability of these calculi as well as the controller synthesis against PTA models. Based on these results, we propose a reward scheme for RL agents in such a way that guarantees that any RL algorithm maximizing rewards is guaranteed to maximize the probability of satisfaction for the given DC specification. The effectiveness of the proposed approach is demonstrated via grid-world benchmarks and a proof-of-concept case study for synthesizing control for simple cardiac pacemaker directly from a set of DC specifications.

***Index Terms*—Duration Calculus, Probabilistic Timed Automata, Model-free Reinforcement Learning, Event-Triggered and Time-Triggered Real-Time Systems**

I. Introduction

Reinforcement learning (RL) [32] is an approach to controller synthesis where agents rely on reward signals to choose actions aimed at achieving prescribed objectives. This paper investigates the use of RL to search for controllers satisfying complex real-time specifications on probabilistic real-time systems where the environment may not be completely known (but can be learned via sampling). Towards this goal, a key objective of this paper is a model-free reduction of the real-time specifications (Duration Calculus) to scalar reward signals used in model-free RL that is both formally correct (reward-optimal policy is specification-optimal) and effective (RL algorithms consistently converge toward optimal value). *The key contribution is an identification of precise conditions on two natural subclasses of DC that enjoy these properties.*

The controller synthesis for real-time and stochastic systems against formal requirements is an important and challenging problem. The workhorse temporal logics of LTL and CTL [4] can reason about discrete ordering of events, but are incapable of expressing dense-time relationship between various events of interest; for instance, specifying a region of time where events should or should not occur or specifying the number of events within a given time interval. Similarly, while Markov decision processes (MDPs) [27] is an expressive formalism to model systems with stochastic behavior, it cannot express real-time constraints over system events. Duration Calculus (DC) [9] is a highly expressive and succinct logic, which can capture specifications involving durations. On the other hand, probabilistic timed automata (PTAs) [21] generalize MDPs with time-measuring variables (called clocks) to enable modeling of systems with real-time and stochastic behavior. *This paper develops RL-based controller synthesis for systems modeled as PTAs against DC specifications.*

Duration Calculus. DC formulae are interpreted over the signals and a given interval. DC extends propositional logic with the "chop" operator ($\frown$), a special symbol ℓ for the interval length, and operators $\int P$ and $\sum P$ measuring the duration of time when the property P was true and the number of times proposition P was true, respectively. Furthermore, the modalities $\lceil P \rceil$ and $\lceil P \rceil^{\bullet}$ express the fact that a propositional formula P was true over an interval and point interval, respectively. The chop operator fuses the behavior of systems over time intervals, enabling composition. For example, the property "*each of the propositions p, q, r are true exactly once*" can be expressed as:

$$\bigwedge_{x \in \{p,q,r\}} \big[(\mathsf{true} \frown \lceil x \rceil^{\bullet} \frown \mathsf{true}) \wedge \neg(\mathsf{true} \frown \lceil \neg x \rceil \frown \lceil x \rceil^{\bullet} \frown \lceil \neg x \rceil \frown \lceil x \rceil^{\bullet} \frown \mathsf{true})\big],$$

To express the same in LTL (and its real-time generalizations such as MTL or STL) we have to capture all the permutations in which p, q, r may be true in a behaviour, along with the

This material is based upon work supported by the National Science Foundation under grant no. 2009022.

978-1-6654-2802-6/21/$31.00 ©2021 IEEE
DOI 10.1109/RTSS52674.2021.00031

condition that they do not hold true more than once. This example demonstrates that DC can potentially be exponentially more succinct than LTL and its variants. The classical temporal modalities $\Box P$ (P holds forever in the interval) and $\Diamond P$ (P holds sometimes within the interval) can be derived in DC using the chop modality (See, Section II-C).

Example 1.1: Consider a safety property for a closed-loop pacemaker system expressing that the minimum and maximum heart rate: "*within every minute, the heart beats must be within the range* 60 *and* 100". This requirement can be expressed as:

$$\Box\big[(\ell = 60) \rightarrow (60 \leq \textstyle\sum h_b \leq 100)\big],$$

where h_b is a proposition which is true every time instant with a heart beat.

Event-Triggered and Time-Triggered DC. The high expressiveness of DC comes with an undecidable satisfiability [9], making it difficult to have automated tools to verify DC properties. One way to recover decidability is to work on discrete time, thereby stripping away the real time duration measurements offered by $\ell \bowtie c$ and $\int P \bowtie c$. The discrete duration calculus [25] is decidable, and the tool DCVALID [10] checks satisfiability of such formulae by translating it into deterministic finite automata (DFA). Another approach [3] to a decidable verification for timed automata against dense-time DC requirements is to consider the bounded-time fragment of DC against strongly non-Zeno timed automata (akin to the acyclicity assumption in the bounded-time setting).

We consider two subclasses of DC to regain decidability, by restricting the usage of the measurement constructs. The first subclass restricts all measurements to begin from some observed event; we call this subclass *event-triggered* DC (EDC). The second subclass, that we dub *time-triggered* DC (TDC), restricts the measurements to start from a globally integral time point. When restricted to only the $\ell \bowtie c$ measurement (i.e., disallowing the $\int P$ modality), the decidability of these logic subclasses (EDC$[\ell]$ and TDC$[\ell]$) is obtained by translating them to event clock automata [2] and integer-reset timed automata [31] respectively.

Probabilistic Timed Automata. Probabilistic timed automata (PTA) [21] enable modeling of systems with both stochastic and nondeterministic behavior. The popular formal verification tool PRISM [20] permits modeling systems as PTAs and provides automatic verification and controller synthesis for a subclass of PCTL properties. In this work, we are interested in controller synthesis for PTAs against DC specifications. In particular, we characterize subclasses of EDC and TDC calculi for which controller synthesis problem for PTA models remain decidable. We will then propose and solve the unknown model setting where the probabilistic transition structure of the PTA is not explicitly known but can be sampled. We will propose a model-free reinforcement learning (RL) scheme to synthesize optimal strategies for DC specifications. We provide a convergent RL algorithm for PTAs when the specifications are from the decidable fragment of EDC and TDC calculi.

Logic and Reinforcement Learning. RL algorithms have been extended [28], [12], [14], [15], [23], [8] to work with formal specification given as ω-regular objectives instead of scalar reward signals. These approaches are model-free, i.e. RL agent does not explicitly learn a model of the underlying system, but instead directly computes the optimal strategy and hence can be adapted to work with highly scalable artificial neural networks based RL (deep RL) algorithms [29], [32].

The key idea behind these approaches is that they translate the formal objective into a monitor automata (finite automata [23], Büchi automata [12], [15], [8], Rabin automata [28], or parity automata [14]) and device an interpreter that observes the traces of the system for satisfaction using the monitor automata and provide scalar rewards to the learning agent in such a way that the learning agent maximizing an appropriate expected reward objective [13] results in a guaranteed convergence to an optimal policy maximizing the probability of satisfaction of the formal specification. Our RL algorithm is based on a similar model-free RL scheme where we reduce the time-triggered and event-triggered specifications into various variants of timed/stopwatch automata and design an interpreter that executes these automata over the traces resulting from the agent's choices to scalar rewards.

Key Challenges. There are two key challenges in designing a convergent model-free RL algorithm for synthesis problem for PTA against DC: 1) the state space of PTA is uncountably infinite; hence convergent RL approaches are not directly applicable, and 2) the controller synthesis for the general time- and event-triggered DC specifications reduce to nondeterministic stopwatch automata and due to non-determinism are not convenient for providing deterministic rewards.

We overcome these issues by considering subclasses of EDC and ADC without the $\int P$ modality. As discussed earlier, EDC$[\ell]$ and TDC$[\ell]$ can be compiled into event-clock and integer-reset timed automata, respectively; both of these subclasses belong to determinizable subclasses of timed automata. It is a well-known result [30], [16] for probabilistic timed automata that search for the optimal strategy in PTA against determinizable timed automata specifications can be reduced to a digital-clock abstraction [16] where the variables are restricted to take only integer values (closely related to the region graph [1] and corner-point abstraction [7]). *We emphasize that the proposed approach does not assume discrete-time semantics. Instead, it characterizes DC subclasses and conditions in such a way that the search for an optimal controller can be limited to controllers with integral delays.*

When the model is unknown, it is not feasible to construct a digital-clock model explicitly, however, we implicitly construct this abstraction by restricting RL-agent to choose timed actions with integral time delays. In addition, the RL agent abstracts the observed state to the nearest integral value of the clocks and updates the corresponding Q-value. The finiteness of the MDP corresponding to the digital clock abstraction of the PTA and the specification automata guarantees the convergence of the Q-learning.

Contributions. The key contributions of this paper are summarized below.

1) It introduces event-triggered (EDC) and time-triggered (TDC) subclasses of DC and presents a translation of these calculi into subclasses of stopwatch automata.
2) It further investigates subclasses EDC$[\ell]$ and TDC$[\ell]$ of EDC and TDC, respectively, that disallow the $\int P$ modality. For these subclasses, the paper presents a translation to determinizable subclasses event-clock (timed) automata and integer-rest timed automata, respectively. This translation provides us the decidability of the satisfiability as well as controller synthesis for logics EDC$[\ell]$ and TDC$[\ell]$ against PTA models.
3) The paper details the first RL-based algorithm to learn optimal strategy against EDC$[\ell]$ and TDC$[\ell]$ specifications for unknown PTA.
4) Finally, the paper presents an implementation of the proposed RL algorithm as part of the extensible formal RL tool MUNGOJERRIE and demonstrate the effectiveness of the proposed approach over some grid-world benchmarks and an artificial pacemaker based case study.

Organization. The paper is organized as follows. In Section II we formalize the problem by introducing necessary definitions and stating some known results. We introduce probabilistic stopwatch automata (PSA) as a general automatic structure and define various subclasses of interest such as PTAs, stopwatch automata, and timed automata. We also formally introduce the DC logic, and state the key synthesis problem. In Section III and IV, we propose two variants of duration calculus, *time-triggered DC* (TDC) and *event-triggered DC* (EDC) and study their theoretical properties. In section V we present the details of our RL algorithms to solve the synthesis problem for unknown PTAs. In section VI, we present the results of our implementation using some benchmarks.

II. Problem Definition

We write $\mathbb{N}$ and $\mathbb{N}_{\geq 0}$ for the set of nonnegative and positive integers. Similarly, we write $\mathbb{R}$ and $\mathbb{R}_{\geq 0}$ for the set of reals and nonnegative reals, respectively. Let $[\![k]\!]_{\mathbb{R}}$ denote the set of reals in $[0, k]$, while $[\![k]\!]_{\mathbb{N}}$ denotes the set of naturals $\{0, 1, \ldots, k\}$. We write $\bowtie$ for any of the operator in $\{>, \geq, =, \leq, <\}$.

In this paper, we model real-time systems as probabilistic timed automata with unknown probability distributions and express their properties using subclass of duration calculus (DC) formulae. The DC formulae will be compiled into stopwatch, and timed automata, and these automatic structures will then be used in model-free reinforcement learning algorithms to design appropriate reward schemes. We begin this section by first formally defining the semantics of Markov decision processes (Section II-A), followed by the definitions of various subclasses of stochastic timed and stopwatch automata (Section II-B). We then introduce our specification formalism Duration calculus (Section II-C).

A. Markov Decision Processes

A *discrete probability distribution* over a (possibly countable) set S is a function $d : S \to [0, 1]$ such that $\sum_{s \in S} d(s) = 1$ and $supp(d) = \{s \in S \mid d(s) > 0\}$ is at most countable. We say that $d : S \to [0, 1]$ is a point distribution if $d(s) = 1$ for some $s \in S$. Let $\mathcal{D}(S)$ be the set of distributions over S.

A *Markov decision process* $\mathcal{M}$ is a tuple (S, s_0, A, T) where: S is a (potentially uncountable) set of *states*, $s_0 \in S$ is the *initial state*, A is a (potentially uncountable) set of *actions*, and $T : S \times A \to \mathcal{D}(S)$ is the *probabilistic transition function*. We say that the MDP $\mathcal{M}$ is *finite* if both S and A are finite. Abusing notation, we call an MDP a finite automaton if $T(s, a)$ is a point distribution for all $s \in S$ and $a \in A$.

For any state $s \in S$, we let $A(s)$ denote the set of actions that can be selected in state s. For states $s, s' \in S$ and $a \in A(s)$, we write $p(s'|s, a)$ for $T(s, a)(s')$. A *run* of $\mathcal{M}$ is a finite sequence $\langle s_0, a_1, s_1, \ldots, s_n \rangle \in S \times (A \times S)^*$ such that $p(s_{i+1}|s_i, a_{i+1}) > 0$ for all $0 \leq i < n$. We write $Runs^{\mathcal{M}}$ for the set of runs of the MDP $\mathcal{M}$ and $Runs^{\mathcal{M}}(s)$ for the set of runs starting from state s. We write $last(r)$ for the last state of a finite run r and $len(r)$ for the number of transitions in r. A strategy in $\mathcal{M}$ is a function $\pi : FRuns \to \mathcal{D}(A)$ such that $supp(\pi(r)) \subseteq A(last(r))$. Let $Runs^{\mathcal{M}}_{\pi}(s)$ denote the subset of runs $Runs^{\mathcal{M}}(s)$ that correspond to strategy π with initial state s. Let $\Pi_{\mathcal{M}}$ be the set of all strategies. A strategy π is *deterministic* if $\pi(r)$ is a point distribution for all runs $r \in FRuns^{\mathcal{M}}$ and we say that π is *stationary* if $last(r) = last(r')$ implies $\pi(r) = \pi(r')$ for all runs $r, r' \in FRuns^{\mathcal{M}}$. A strategy is *positional* if it is both pure and stationary.

The behavior of an MDP $\mathcal{M}$ under a strategy π is defined on a probability space $(Runs^{\mathcal{M}}_{\pi}(s), \mathcal{F}_{Runs^{\mathcal{M}}_{\pi}(s)}, \Pr^{\mathcal{M}}_{\pi}(s))$ over the set of runs of π with starting state s. Given a real-valued random variable over the set of infinite runs $f : Runs^{\mathcal{M}} \to \mathbb{R}$, we denote by $\mathbb{E}^{\mathcal{M}}_{\pi}(s)\{f\}$ the expectation of f over the runs of $\mathcal{M}$ originating at s following π.

A *rewardful* MDP is a pair $(\mathcal{M}, \rho)$, where $\mathcal{M}$ is an MDP and $\rho : S \times A \to \mathbb{R}$ is a reward function assigning utility to state-action pairs. A rewardful MDP $(\mathcal{M}, \rho)$ under a strategy π determines a sequence of random rewards $\rho(X_{i-1}, Y_i)_{i \geq 1}$, where X_i and Y_i are the random variables denoting the i-th state and action, respectively. Given a target set $S_t \subseteq S$, the *reachability probability* $Reach(S_t)^{\mathcal{M}}_{\pi}(s)$ is defined as

$$Reach(S_t)^{\mathcal{M}}_{\pi}(s) \left\{ \langle s, a_1, s_1, \ldots \rangle \in Runs^{\mathcal{M}}_{\pi}(s) : \exists i . s_i \in S_t \right\}.$$

The optimal reachability probability $Reach(S_t)^{\mathcal{M}}_{*}(s)$ from $s \in S$ is defined as $Reach(S_t)^{\mathcal{M}}_{*}(s) = \sup_{\pi \in \Pi_{\mathcal{M}}} Reach(S_t)^{\mathcal{M}}_{\pi}(s)$. We say that a strategy $\pi \in \Pi_{\mathcal{M}}$ is optimal if the $Reach(S_t)^{\mathcal{M}}_{\pi}(s) = Reach(S_t)^{\mathcal{M}}_{*}(s)$ for all $s \in S$.

B. Modeling Stochastic Real-Time Systems

We fix a finite set X of (clock or stopwatch) *variables*. A *valuation* is a function $\nu : X \to \mathbb{R}_{\geq 0}$. We write $\mathbf{0}$ for the valuation $x \in X \mapsto 0$. An integer valuation is a function $\nu : X \to \mathbb{N}$. For $k \in \mathbb{N}$, we say that an integer valuation ν is k-bounded if $\nu(x) \leq k$ for all $x \in X$. We write V, $V_{\mathbb{N}}$,

and $V_{\mathbb{N}}^k$ for the set of all valuations, integer valuations, and k-bounded integer valuations, respectively.

Let $\gamma : X \to \{0,1\}$ be a *rate-configuration* characterizing paused ($\{x{\in}X : \gamma(x){=}0\}$) and ticking ($\{x{\in}X : \gamma(x){=}1\}$) variables. For $\nu \in V$, rate-configuration γ, and time step $t \in \mathbb{R}_{\geq 0}$, we write $\nu \oplus_\gamma t$ for the valuation after time t, i.e.

$$(\nu \oplus_\gamma t)(x) = \nu(x) + \gamma \cdot t,$$

for all $x \in X$. We write $\nu + t$ for $\nu \oplus_\gamma t$ when γ equals $x \mapsto 1$. For $X \subseteq X$, we write $\nu[X{:=}0]$ for the valuation after resetting variables in $X \subseteq X$, i.e.

$$\nu[X{:=}0](x) = \begin{cases} 0 & \text{if } x \in X \\ \nu(x) & \text{otherwise.} \end{cases}$$

The set of *constraints* over X is the set of conjunctions of constraints of the form $x \bowtie i$ or $x{-}x' \bowtie i$, where $x, x' \in X, i \in \mathbb{N}$. For every $\nu \in V$, let $\mathrm{SCC}(\nu)$ be the set of constraints that hold in ν. A *region* is a maximal set $\eta \subseteq V$, such that $\mathrm{SCC}(\nu){=}\mathrm{SCC}(\nu')$ for all $\nu, \nu' \in \eta$. Every region is an equivalence class of the indistinguishability-by-constraints relation, and vice versa. A *zone* is a convex set of valuations, that is a union of a set of regions. We write $\mathcal{Z}$ for the set of zones. For any zone W and valuation ν, we use the notation $\nu \in W$ to denote that $[\nu] \subseteq W$. A set of valuations is a zone if and only if it is definable by a constraint.

Definition 2.1: A probabilistic stopwatch automaton (PSA) is a tuple $\mathcal{T} = (Q, q_0, \Sigma, X, \gamma, E, \delta, F)$ where

- Q is the finite set of *locations*,
- $q_0 \in Q$ is the initial *location*,
- Σ is a finite alphabet of *actions*,
- X is the finite set of (clock or stopwatch) *variables*,
- $\gamma : Q \times \Sigma \to (X \to \{0,1\})$ is the *rate* function
- $E : Q \times \Sigma \to \mathcal{Z}$ is the *action guard*,
- $\delta : Q \times \Sigma \to \mathcal{D}(2^X \times Q)$ is the *transition function*
- and $F \subseteq Q$ is the set of final locations.

The following are some important subclasses of PSA:

1) We say that a PSA is a *probabilistic timed automaton* (PTA) if for all $q \in Q$, $a \in \Sigma$, and $x \in X$ we have that $\gamma(q,a)(x) = 1$. For a PTA, we omit the description of the rate configuration function and represent it as a tuple $(Q, I, \Sigma, X, E, \delta, F)$ and refer to its variables as *clocks*.
2) We say that a PSA is a (non-probabilistic) *stopwatch automaton* (SA) if for all $q \in Q$, $a \in \Sigma$, and $\varphi \in \mathcal{Z}$, we have that $\delta(q,a)$ is a point distribution.
3) We say that a PSA is a *timed automaton* (TA) if it satisfies both of the previous conditions, i.e. all of the variables are clocks and transitions are non-probabilistic.

A *configuration* of a PSA $\mathcal{T}$ is a pair (q, ν), where $q \in Q$ is a location and $\nu \in V$ is a valuation. Informally, the behavior of a PSA is as follows: In configuration (q, ν) time passes before an available action from Σ is enabled, after which a discrete probabilistic transition occurs. An action $a \in \Sigma$ can be chosen after time t elapses only if the action a is enabled in the configuration reached after t time, i.e. $\nu{\oplus}_{\gamma(q,a)}t \in E(q,a)$. Both the time and the action chosen are nondeterministic. If an action a is chosen, then the probability of moving to a location q' and resetting all of the variables in $X \subseteq X$ to 0 is given by $\delta(q,a)(X,q')$. Formally, the semantics of a PSA is an MDP with uncountable states and actions.

The semantics of PSA $\mathcal{T} = (Q, q_0, I, \Sigma, X, \gamma, E, \delta, F)$ is given as an MDP $[\![\mathcal{T}]\!] = (S, s_0, A, T)$ where: $S \subseteq Q{\times}V$ is the set of states; $s_0 = (q_0, \mathbf{0}) \subseteq Q{\times}V$ is the initial state; $A = \mathbb{R}_{\geq 0}{\times}Act$ is the set of (timed) *actions*; $T : S \times A \to \mathcal{D}(S)$ is such that for $(q,\nu) \in S$ and $(t,a) \in A$, we have $T((q,\nu),(t,a)) = d$ if and only if $\nu \oplus_{\gamma(q,a)} t \in E(q,a)$ and

$$d((q',\nu'))= \sum_{\substack{X \subseteq X \\ (\nu \oplus_{\gamma(q,a)} t)[X:=0]=\nu'}} \delta(q,a)(X,q')$$

for all $(q',\nu') \in S$. From this semantics, a run of a PSA is defined as a sequence of configurations and timed actions, i.e. $\langle (q_0,\nu_0),(t_1,a_1),(q_1,\nu_1),\ldots,(q_n,\nu_n)\rangle$ where $\nu_0 = \mathbf{0}$. The concept of strategy and optimal reachability probability are naturally defined based on its MDP semantics.

C. Duration Calculus

Let Var be a finite set of atomic propositions. We interpret DC formulae over timed traces generated from runs of systems modeled as a probabilistic stopwatch automaton $\mathcal{T} = (Q, q_0, I, \Sigma, X, \gamma, E, \delta, F)$, $\Sigma = 2^{Var}$. For every timed run $r = \langle (q_0,\nu_0),(t_1,a_1),(q_1,\nu_1),\ldots,(q_n,\nu_n)\rangle$ of $\mathcal{T}$ we associate its timed trace $\sigma = \langle (s_0,\tau_0),(s_1,\tau_1),\ldots,(s_n,\tau_n)\rangle$ where $\tau_0 = 0$ and for every $0 < i \leq n$ we have $\tau_i = \tau_{i-1} + t_i$ and $s_i = a_i$ for all $0 \leq i \leq n$.

Definition 2.2 (Duration Calculus: Syntax): Given the set Var, we define the syntax of DC formulae as follows:

$$\begin{aligned} P &::= \mathsf{false} \mid \mathsf{true} \mid x \in Var \mid P \wedge P \mid \neg P \\ D &::= \lceil P \rceil \mid \lceil P \rceil^\bullet \mid D \wedge D \mid \neg D \mid D^\frown D \mid M \\ M &::= \ell \bowtie c \mid \int P \bowtie c \mid \sum P \bowtie c \end{aligned}$$

DC formulae are evaluated over timed traces σ and a reference interval $I = [b,e]$ where $b \leq e$ and $b, e \in \mathbb{N}$ range over the indices $0,\ldots,n$ of the timed trace $\sigma = (s_0,\tau_0)\ldots(s_n,\tau_n)$. The satisfaction of a DC formula ψ evaluated on timed trace $\sigma = \langle (s_0,\tau_0),(s_1,\tau_1),\ldots\rangle$ with respect to an interval $[b,e]$ and is denoted as $(\sigma,[b,e]) \models \psi$.

For a timed trace $\sigma = \langle (s_0,\tau_0)(s_1,\tau_1)\ldots(s_n,\tau_n)\rangle$ and propositional formula P, we say $(\sigma,i) \models P$ iff $s_i \models P$. The satisfaction of other DC formulae is defined inductively:

1) $(\sigma,[b,e]) \models \lceil P \rceil$ iff $b{<}e$, and $(\sigma,t) \models P$ for all $b{<}t{<}e$;
2) $(\sigma,[b,e]) \models \lceil P \rceil^\bullet$ iff $b{=}e$ and $(\sigma,b) \models P$;
3) $(\sigma,[b,e]) \models D_1{\wedge}D_2$ iff $(\sigma,[b,e]) \models D_1$,$(\sigma,[b,e]) \models D_2$;
4) $(\sigma,[b,e]) \models \neg D$ iff $(\sigma,[b,e]) \not\models D$;
5) $(\sigma,[b,e]) \models D_1^\frown D_2$ iff there exists a point $b \leq z \leq e$ s.t. $(\sigma,[b,z]) \models D_1$ and $(\sigma,[z,e]) \models D_2$;
6) $(\sigma,[b,e]) \models \ell \bowtie c$ iff $(\tau_e - \tau_b) \bowtie c$ holds;
7) $(\sigma,[b,e]) \models \int P \bowtie c$ iff $\sum\{\tau_{i+1}{-}\tau_i \,:\, (\sigma,i) \models P\} \bowtie c$;
8) $(\sigma,[b,e]) \models \sum P \bowtie c$ iff $|\{i \,:\, (\sigma,i) \models P\}| \bowtie c$.

We often refer to the measurement operators $(\sum P, \int P, \ell)$ collectively as mt. The subclass of DC, which does not use $\int P$

and ℓ operators, is known as *discrete duration calculus* (DDC). The tool QDDC [25] works with this discrete subclass of DC, devoid of real time measurements.

Using the chop modality $\frown$, we can derive the following syntactic sugar: eventually modality $\Diamond D \stackrel{\text{def}}{=} \mathsf{true} \frown D \frown \mathsf{true}$ and globally modality $\Box D \stackrel{\text{def}}{=} \neg\Diamond\neg D$. We also define the integral duration modality $\ell \in \mathbb{N}$ to denote an integral interval, that is, $(\sigma, [b, e]) \models \ell \in \mathbb{N}$ iff $\tau_e - \tau_b \in \mathbb{N}$. Given a DC formulae ϕ, the DC satisfiability problem is to decide whether there exists a timed trace σ and interval I such that $(\sigma, I) \models \phi$.

Theorem 2.3: [Chaochen, Hansen, and Sestoft [9]] The DC satisfiability problem is undecidable.

The undecidability can easily be shown by reducing the halting problem of the two-counter (Minsky) machines to the satisfiability of DC. One can use the measurement construct $\ell \bowtie c$ to encode configurations of a two counter machine, along with the chop operator as well as $\lceil p \rceil$ for propositional variables p. The fact that measurement operators can be used in an unrestricted manner forms the core of the undecidability. This motivates the need for a class of DC (other than DDC, over discrete time [25]) which has a decidable satisfiability.

D. Synthesis for Unknown PTA against DC Specifications

Given a PTA $\mathcal{T} = (Q, I, \Sigma, X, E, \delta, F)$, set of atomic propositions Var, labeling function $\mathcal{L} : Q \to Var$, and DC specification ϕ, the controller synthesis problem is to compute a strategy $\pi \in \Pi_{[\![\mathcal{T}]\!]}$ that maximizes the probability of satisfaction of ϕ. We are also interested in the unknown PTA setting where the probability distribution $\delta(q, a)$ is not known but can be learned from repeated sampling of the environment. The following is an easy corollary of Theorem 2.3.

Theorem 2.4: The controller synthesis problem for PTA against DC specifications in undecidable.

In view of this negative result, we present two subclasses of DC—that we dub event-triggered and time-triggered DC. We present a reduction of these calculi to stopwatch automata and present further restrictions to recover decidability for the satisfiability as well as the controller synthesis problem for PTA. In Section V, these results will form the basis of our convergent RL algorithm for controller synthesis of these subclasses against unknown PTA.

III. Event-Triggered and Time-Triggered DC

One of the ways to recover decidability is to work with bounded time. This means that we work with formulae of the form $D \wedge (\ell \leq K)$ for some time bound K. The formula D can be an arbitrary DC formula. However, the conjunct $\ell \leq K$ ensures that we only observe behaviours over time intervals of length at most K. We call this variant *bounded* DC. The decidability of model checking DC formulae in [3] is in this spirit. In contrast, this work presents an alternative way to recover decidability without imposing time-boundedness by designing practical, expressive yet decidable subclasses of DC.

A. Event-Triggered Duration Calculus (EDC)

Oftentimes the start of various tasks or processes is prompted by the arrival of an external event (e.g., an interrupt) from the environment or by the progress (termination) of the preceding task. We refer to systems where all tasks are triggered by such events as an *event-triggered* system. Our key observation is that for event-triggered systems, restricted use of the measurement operators suffices to express properties. We argue that time since the arrival of an event is an important requirement for such systems. Consider, for instance, the case of online transactions where in the event of a purchase, an OTP (one time password) gets generated. This OTP remains valid only for a specific time from the purchase. Measuring the time since the purchase happened, therefore, is important for checking the validity of the transaction via OTP. Yet another example is in accessing secure information like logging into one's account: once the user name/account number has been provided (this is the event), the password/PIN must be correctly entered within some time duration. The number of trials allowed in such high security applications is also limited, after which the account gets blocked either temporarily or permanently. These examples motivate using $\ell \bowtie c$ and $\int P \bowtie c$ in a guarded fashion by attaching them to the arrival of some fixed event. We express such modality by $mt \bowtie^{\bullet}_{B} c$ to indicate that we evaluate $mt \in \{\ell, \int P\}$ since the last time some proposition B was true. The semantics of $mt \bowtie^{\bullet}_{B} c$ is defined as follows. Given a timed word σ, and a point interval $[e, e]$, we say that $(\sigma, [e, e]) \models mt \bowtie^{\bullet}_{B} c$ iff

$$\exists\ b{<}e \text{ s.t. } (\sigma, b) \models B \text{ and } (\sigma, [b, e]) \models (\lceil \neg B \rceil \wedge mt \bowtie c).$$

This restriction breaks the undecidability result for DC as the Theorem 2.3 requires measuring durations arbitrarily and not since the last occurrence of some proposition.

Definition 3.1 (Event-Triggered Duration Calculus): Given the set of variables Var, we define the syntax of event triggered DC (EDC) as follows:

$$\begin{aligned} P &::= \mathsf{false} \mid \mathsf{true} \mid x \in Var \mid P \wedge P \mid \neg P \\ D &::= \lceil P \rceil \mid \lceil P \rceil^{\bullet} \mid D \wedge D \mid \neg D \mid D \frown D \mid M_{\texttt{EDC}} \\ M_{\texttt{EDC}} &::= \ell \bowtie^{\bullet}_{B} c \mid \int P \bowtie^{\bullet}_{B} c \mid \sum P \bowtie c \end{aligned}$$

Let EDC[ℓ] denote the subclass of EDC which uses only measurement constructs $\ell \bowtie^{\bullet}_{B} c$ and $\sum P \bowtie c$.

The semantics of the event-triggered DC can be given in a similar fashion as that for DC.

B. Time Triggered DC (TDC)

In time-triggered systems, the start of the task is triggered by the progression of a global notion of time, i.e., when the global time reaches a specified value. The notion of time-triggered systems finds a natural motivation in the time-triggered system architectures such as [18] and [19]. Kopetz [18] defines time-triggered systems as *distributed systems where a system-wide global time base of* known precision *is provided at every node to trigger the execution of significant computational and*

communication tasks. So while time is dense and global, all tasks work with global but sparse time base where dense time is broken into granular intervals of time of fixed precision.

This notion motivates the globally available set of reference time points to all nodes and motivates the subclass of DC where we restrict measurements $M = mt\,op\,c$ only from such global reference time points. We consider these reference points as globally integral points of time. This is written as $mt \bowtie^{\bullet}_{\mathtt{tic}} c$ to indicate that we evaluate $mt \in \{\ell, \int P\}$ since a globally integral time. The semantics of $mt \bowtie^{\bullet}_{\mathtt{tic}} c$ is defined as follows. Given a timed word σ, and a point interval $[e, e]$, we say that $\sigma, [e, e] \models mt \bowtie^{\bullet}_{\mathtt{tic}} c$ iff

$$\exists\ b \leq e,\ \sigma, [0, b] \models \ell \in \mathbb{N},\ (\sigma, [b, e]) \models mt \bowtie c$$

Once again, it can be seen that the undecidability result of Theorem 2.3 needs to use measurement constructs not just from globally integral time points.

Definition 3.2 (Time-Triggered Duration Calculus): Given the set of variables Var, we define the syntax of time-triggered DC (TDC) as follows:

$$\begin{aligned} P &::= \mathsf{false} \mid \mathsf{true} \mid x \in Var \mid P \wedge P \mid \neg P \\ D &::= \lceil P \rceil \mid \lceil P \rceil^{\bullet} \mid D \wedge D \mid \neg D \mid D^{\frown} D \mid M_{\mathtt{TDC}} \\ M_{\mathtt{TDC}} &::= \ell \bowtie^{\bullet}_{\mathtt{tic}} c \mid \int P \bowtie^{\bullet}_{\mathtt{tic}} c \mid \sum P \bowtie c \end{aligned}$$

Let TDC[ℓ] denote the subclass of TDC which uses only measurement constructs $\ell \bowtie^{\bullet}_{\mathtt{tic}} c$ and $\sum P \bowtie c$.

Again, the semantics of the time-triggered DC can be given in a similar fashion as that for DC.

IV. Theoretical Properties of EDC and TDC

A formula $D \in$ EDC (TDC) having measurement constructs $\varphi_1, \ldots, \varphi_n$ with $\varphi_i = mt \bowtie^{\bullet}_{B_i} c_i$ ($mt \bowtie^{\bullet}_{\mathtt{tic}} c$), $mt \in \{\ell, \int P\}$, will be denoted by $D(\varphi_1, \ldots, \varphi_n)$ in our algorithms. The key result of this section are algorithms to check for the satisfiability of EDC[ℓ] and TDC[ℓ].

Given a set of propositional variables $Var = \{p_1, \ldots, p_n\}$, and a proposition P (a Boolean combination of variables from Var), let Var_P denote subsets s of Var which satisfy P, i.e.

$$Var_P = \{s \subseteq Var \mid s \models P\}.$$

A. Automata based characterization

Definition 4.1 (Event-Driven Stopwatch Automata (ESA)): We use alphabet $\Sigma = 2^{Var}$ where Var is a finite set of propositional variables. We unify the E, γ, δ in Definition 2.1 and denote transitions as $(q, a, Y_a, \varphi, \gamma_a, q') \in \delta$, where $a \in \Sigma, \varphi \in E(q, a)$, γ_a is the rate mapping, and $Y_a \subseteq X$ is the set of variables reset. An event-driven stopwatch automaton is a stopwatch automaton $\mathcal{T} = (Q, q_0, \Sigma, X, \gamma, E, \delta, F)$ with following properties:

1) The variables X in the ESA are tied to Var; that is, for each $p \in Var$, we allow clock, stopwatch variables c_p, s_p such that $X = \bigcup_{p \in Var}(c_p \uplus s_p)$ where c_p, s_p respectively are the clock, stopwatch variables associated to p.
2) Further, events (symbols from Σ) drive the transitions. We succinctly write a transition as the tuple $(q, a, Y_a, \varphi, \gamma_a, q')$ where $E(q, a) = \varphi$ and $\delta(q, a) = (Y_a, q')$ and $\gamma(q, a) = \gamma_a$. Since transitions are driven by events, an ESA is *event recording*. On any transition $(q, a, Y_a, \varphi, \gamma_a, q') \in \delta$,
   - the reset set Y_a is determined by $a \in \Sigma$. There is a mapping $reset : \Sigma \mapsto 2^X$ which assigns to each $a \in \Sigma$, the set of variables $Y_a \subseteq X$ which are reset each time a transition on a happens. Thus, the resets are determined by the symbol from Σ on which the transition is taken.
   - the rate mapping γ_a is determined by $a \in \Sigma$. Thus, for the set of stopwatch variables $S = \bigcup_{p \in Var} S_p$, $\gamma_a = Y_a \subseteq S$ determines the subset Y_a of stopwatch variables which have rate 1, whenever a transition on a is taken. For instance, if $\gamma_a = \{p, q\}$, then all transitions on a will assign rate 1 to stopwatches s_p, s_q and rate 0 to other stopwatches. The clocks have rate 1 on all transitions.

Clearly, ESAs extend Event Clock Automata (ECA) [2] which are a well behaved subclass of timed automata. ECAs are known to be determinizable, complementable and have a decidable universality and inclusion, unlike general timed automata [1]. When the ESA has only clock variables (no stopwatches), then it is an ECA. ESAs also demonstrate some of the nice properties of ECAs. As expected,

Lemma 1: ESA are determinizable.

However, surprisingly,

Theorem 4.2: The reachability for ESAs is undecidable.

Definition 4.3 (Integer Reset Timed Automata (IRTA)): An IRTA is a timed automaton $\mathcal{T} = (Q, q_0, \Sigma, X, E, \delta, F)$ where clock resets happen only on transitions where the global time elapsed so far is integral. The main characteristics of IRTA are defined by the transitions, which have the form (q, a, Y, φ, q'); note that we drop γ since all clocks have rate 1. On any transition (q, a, Y, φ, q') where $Y \neq \emptyset$, the constraint φ must be of the form $(x = c) \wedge \psi$ where $c \in \mathbb{N}$ and ψ is some constraint. ψ can also be true (given by $y \geq 0$ for $y \in X$). IRTAs [31] are well behaved class of timed automata as they are determinizable and closed under all Boolean operations.

Recall that any run begins at time 0 when all clocks have value 0. The condition on resets ensures that when a reset happens, the time elapsed so far is integral. Consider the *first* time a reset happens in a run. Clearly, after the reset, some clocks have value 0 (since they were reset), and all others have the same value c (for some $c \in \mathbb{N}$). This is the c that has been used in the constraint $x = c$ of the resetting transition for some $x \in X$. Notice that all clocks are integral when the reset happens. Since resets happen only when a constraint of the form $x = c$ is fulfilled, clearly, resets happen only after integral time elapses. This ensures that all clocks have aligned fractional parts always since they always keep ticking.

Theorem 4.4: [31] IRTA are a determinizable class of timed automata and have a decidable reachability.

Algorithm 1 ElimMeasure(D) : Eliminate Measurement Constructs from D, $D \in$ EDC

Input: EDC formula $D(\varphi_1, \ldots, \varphi_n)$ over Var, with measurement constructs $\varphi_1, \ldots, \varphi_n$, each $\varphi_i = B_i \curvearrowleft M_i$.
Output: DDC+Past formula D' with no measurement constructs, over Var'=$Var \uplus \{E_1, \ldots, E_n\}$

1: **for** each φ_i **do**
2: Introduce a new propositional variable E_i
3: Replace φ_i by $W_i = \lceil E_i \rceil^\bullet \wedge (\lceil B_i \rceil^\bullet \wedge \lceil \neg B_i \rceil)$
4: **end for**
5: $D' = D[\varphi_i \leftarrow W_i]$

B. Reduction from EDC[ℓ] to ECA

We show that we can compile an EDC[ℓ] into an ECA. The decidability of reachability for ECA gives the decidability of the satisfiability for EDC[ℓ]. The translation is given by Algorithms 1, 2 and 3. Note that we can encode the whole logic EDC into ESA; however, since ESA does not have a decidable reachability, we discuss only the EDC[ℓ] to ECA translation here. The full translation is available in the extended version of the paper.

Below, we explain the algorithms and illustrate them with examples. We begin with an EDC[ℓ] formula D over Var consisting of measurement constructs $\varphi_1, \ldots, \varphi_n$, each φ_i of the form $(\ell \bowtie^\bullet_{B_i} c_i)$. We use the shorthand $\mathsf{Past}(\lceil B_i \rceil^\bullet \wedge \lceil \neg B_i \rceil)$ to assert at a point e that, there is a point $b < e$ such that B_i is true at b, and $\lceil \neg B \rceil$ is true in $[b, e]$. That is, $(\sigma, [e, e]) \models \mathsf{Past}(\lceil B_i \rceil^\bullet \wedge \lceil \neg B_i \rceil)$ iff

$$\exists b < e, \quad \sigma, b \models B_i, \quad \sigma, [b, e] \models \lceil \neg B_i \rceil.$$

Algorithm 1 does the following : for each measurement construct φ_i in D, it introduces a fresh propositional variable E_i, and replaces φ_i with $\lceil E_i \rceil^\bullet \wedge \mathsf{Past}(\lceil B_i \rceil^\bullet \wedge \lceil \neg B_i \rceil)$. This helps in obtaining a simpler formula devoid of measurement constructs, over a larger set of variables $Var' = Var \uplus \{E_1, \ldots, E_n\}$. Note that the resultant formula is a DDC formula enhanced with the Past macro, that is, it is a DDC+Past formula.

Example 4.1 (Illustration of Algorithm 1): Consider the EDC formula

$$D(\varphi_1) = (\lceil \neg p \rceil^\bullet \frown \lceil p \rceil \rightarrow true \frown (\ell \leq^\bullet_{\neg p} 60))$$

which says that p cannot be true for longer than 60 time units. Recall the semantics of $\ell \leq^\bullet_{\neg p} 60$, which when asserted at a point $[e, e]$, says that there is a point $b < e$ where $\lceil \neg p \rceil^\bullet$ is true, and $\lceil p \rceil$ holds in $[b, e]$, while ensuring that real time duration of $[b, e]$ is ≤ 60. It has the measurement construct φ_1=$(\ell \leq^\bullet_{\neg p} 60)$ over Var=$\{p\}$. Using the fresh variable E_1 we obtain the formula

$$D'=(\lceil \neg p \rceil^\bullet \frown \lceil p \rceil \rightarrow true \frown [\lceil E_1 \rceil^\bullet \wedge \mathsf{Past}(\lceil \neg p \rceil^\bullet \wedge \lceil p \rceil))$$

over $Var' = \{p, E_1\}$.

Let us proceed to Algorithm 2. This converts the DDC+Past formula D' into deterministic finite automata (DFA). The

Algorithm 2 Formula2Aut(D'): Convert DDC+Past formula D' without measurements to automata $A(D')$

Input: DDC+Past Formula D' over Var' with no measurement constructs.
Output: Automaton $A(D')$ s.t. $L(A(D'))$=$Untime(L(D'))$.

1: **if** $D' = \lceil P \rceil^0$ **then**
2: $A(D')$= 
3: Determinize $A(D')$
4: **end if**
5: **if** $D' = \lceil P \rceil$ **then**
6: $A(D')$= 
7: Determinize $A(D')$
8: **end if**
9: **if** D'=$\mathsf{Past}(\lceil B \rceil^0 \wedge \lceil \neg B \rceil)$ **then**
10: $A(D')$= 
11: Determinize $A(D')$
12: **end if**
13: **if** $D' = D_1 \frown D_2$ **then**
14: $A(D')$= 
15: Determinize $A(D')$
16: **end if**
17: **if** $D' = D_1 \wedge D_2$ **then** $A(D) = A(D_1) \cap A(D_2)$
18: **end if**
19: **if** $D' = \neg D_1$ **then** $A(D) = \overline{A(D_1)}$
20: **end if**

translation is done inductively starting from the atomic formulae $\lceil p \rceil^\bullet, \lceil p \rceil$ and then proceeding to handle chop as well as Boolean operations. The macro Past is handled separately as well. Once this is done, we know that we can always obtain a DFA. The DFA $A(D')$ constructed corresponding to the DDC+Past formula D' has the following property : for any $w \in L(D')$, $Untime(w) \in L(A(D'))$. $Untime(w)$ removes the time stamps from w. This is the first step before obtaining the ECA corresponding to D'.

Example 4.2 (Illustration of Algorithm 2): Let us continue from where we left off in Example 4.1, from the formula D'. The automaton corresponding to D' is in the left of Figure 1.

Let us now proceed to Algorithm 3. The deterministic finite automaton A (referred to as $A(D')$ in Example 4.1) obtained from Algorithm 2 is converted into a ECA. This is done by Algorithm 3. We introduce clocks x_i into A to check the measurement constructs $\varphi_i = \ell \bowtie^\bullet_{B_i} c_i$ (line 2, Algorithm 3). φ_i must hold good at the point where its witness variable E_i is true in A. We must check the time elapsed since the last B_i, say t is such that $t \bowtie c$. In the automaton A, we reset x_i on each transition where B_i is true (line 7, Algorithm 3). Then, on each transition of A where E_i is true, we check the constraint $x_i \bowtie c$ (lines 10-14, Algorithm 3). The clock x_i measures the time elapsed since the last B_i; we check if this value is $\bowtie c$ when we reach the transition with E_i. Since E_i is the witness for φ_i, satisfaction of $x_i \bowtie c$ on the

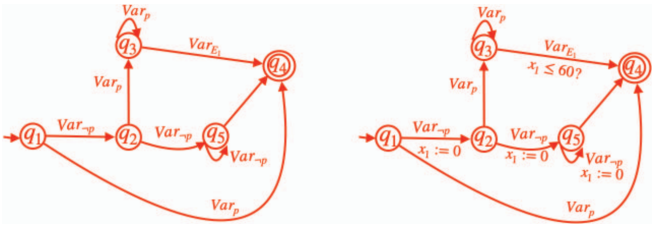

Figure 1: EDC[ℓ] to ECA : running example. On the left is $A(D')$ for D' from Example 4.1. Recall that for $p \in Var$, Var_p is the set of all subsets of Var containing p. In $A(D')$, the label on the edge from q_5 to q_4 is immaterial since the LHS $\lceil \neg p \rceil^\bullet \frown \lceil p \rceil$ of D' has already been violated, leading to acceptance. On the right is the ECA obtained from $A(D')$. $x_1 := 0$ denotes reset of x_1 and $(x_1 \leq 60)$ is the constraint. For brevity, we have not put the constraint $\neg(x_1 \leq 60)$ on edges not containing E_1. For example, the edge from q_2 to q_3 is labeled Var_p : this expands to two transitions on $\{p\}$ as well as $\{p, E_1\}$. On the former, the guard is $\neg(x_1 \leq 60)$, while on the latter it is $(x_1 \leq 60)$.

transitions of A where E_i is true amounts to checking φ_i on those transitions.

Algorithm 3 Aut2ECA(A): Convert automaton A over Var' to event clock automata $ECA(A)$

Input: Automaton A over Var', such that $L(A) = Untime(L(D'))$ for DDC+Past formula D' over Var'.
Output: $ECA(A)$ such that $L(ECA(A)) = L(D')$.

1: **for** each $\varphi_i = \ell \bowtie^\bullet_{B_i} c_i$ **do**
2: Introduce a clock variable x_i
3: **end for**
4: **for** each transition $q \xrightarrow{a} q'$ in A, $a \subseteq Var'$ **do**
5: Introduce guard ψ and reset $Y \subseteq \{x_1, \ldots, x_n\}$ as :
6: $\psi \leftarrow true$
7: **if** $a \models B_i$ **then**, $Y = Y \cup \{x_i\}$
8: **end if**
9: **for** $1 \leq i \leq n$ **do**
10: **if** $(a \models E_i) \wedge \varphi_i = (\ell \bowtie^\bullet_{B_i} c)$ **then**
11: $\psi = \psi \wedge (x_i \bowtie c)$
12: **end if**
13: **if** $(a \not\models E_i) \wedge \varphi_i = (\ell \bowtie^\bullet_{B_i} c)$ **then**
14: $\psi = \psi \wedge \neg(x_i \bowtie c)$
15: **end if**
16: **end for**
17: **end for**
18: Replacing all transitions of A, ECA(A) is obtained.

Example 4.3 (Illustration of Algorithm 3): We continue with our running example, and convert the automaton $A(D')$ obtained in Example 4.2 to an ECA. We introduce a clock x_1 for φ_1. The clock x_1 is reset on the transition of $A(D')$ where $\neg p$ is true. On the edges where E_1 is true, we check the constraint $x_1 \leq 60$. As mentioned in Figure 1, each of the labels on edges represent several transitions : as an example, $Var_{\neg p}$ represents transitions of all subsets of Var which do not contain p.

Theorem 4.5: EDC and EDC[ℓ] respectively can be encoded as language preserving ESA and ECA.

C. Reduction from TDC[ℓ] to IRTA

Next, we show that we can encode TDC[ℓ] into IRTA. The decidability of reachability for IRTA gives the decidability of the satisfiability for TDC[ℓ]. The translation is performed by Algorithms 4, 2 and 5 in order.

We begin with a TDC[ℓ] formula D over Var consisting of measurement constructs $\varphi_1, \ldots, \varphi_n$, each φ_i of the form $\ell \bowtie^\bullet_{\mathtt{tic}} c$. As seen in the case of EDC[ℓ], first we use Algorithm 4 which replaces all measurement constructs $\varphi_i = \ell \bowtie^\bullet_{\mathtt{tic}} c_i$ with $\lceil E_i \rceil^\bullet$ where E_i is a fresh propositional variable. This results in obtaining a DDC formula D' devoid of measurements, over Var and the fresh propositional variables added by Algorithm 4. We then convert D' into a deterministic finite automaton A using Algorithm 2 (omitting lines 7-9, since we do not need them for TDC). From this DFA A, we obtain IRTA(A) as given by Algorithm 5.

Algorithm 4 replaces measurement constructs $\ell \bowtie^\bullet_{\mathtt{tic}} c_i$ with the point formulae $\lceil E_i \rceil^\bullet$, where E_i is a fresh propositional variable. By the semantics of $\ell \bowtie^\bullet_{\mathtt{tic}} c_i$, we check the duration from a globally integer time, and see if it agrees with $\bowtie c_i$. For measuring the duration, we will use a clock x_i corresponding to $\ell \bowtie^\bullet_{\mathtt{tic}} c_i$ (line 2, Algorithm 5). In the automaton A' obtained from Algorithm 2 (i) reset x_i at any globally integer time, and (ii) check the constraint $x_i \bowtie c_i$ on edges where E_i is true. To check if a global time is integral or not, we use a new global clock z, which is reset to 0 each time it reaches 1. Thus, each time $z = 0$, we know that the global time is integral. All transitions of A' are duplicated by adding constraints $z = 0$ or $0 < z < 1$ (lines 9-11 of Algorithm 5). Note that the duplication is done to enable transitions to happen at integral times as well as non integral times depending on the time elapse. We can reset x_i on any transition having the guard $z = 0$ since it is a global integer time; we can reset any subset of $\{x_1, \ldots, x_n\}$ on these transitions (lines 13-19 of Algorithm 5). Point (i) above, namely, resetting x_i at globally integral times, is implemented like this. Point (ii) which checks $x_i \bowtie c_i$ is implemented in lines 25-29 of Algorithm 5. On edges where E_i is true, we check the constraint $x_i \bowtie c_i$, and on edges where E_i is not

true, we check the negation $\neg(x_i \bowtie c_i)$.

Theorem 4.6: TDC[ℓ] can be encoded as language preserving IRTA. Hence the satisfiability for TDC[ℓ] is decidable.

EDC is a fragment of logic LIDL [26] without the existential quantifiers. The addition of existential quantifiers to EDC[ℓ] and TDC[ℓ] gives logics that are equivalent to event recording timed automata as well as IRTA respectively.

D. Controller Synthesis for PTA against EDC[ℓ] and TDC[ℓ]

We consider controller synthesis problem for a PTA $\mathcal{T}$, a labelling function $\mathcal{L} : Q \to 2^{Var}$, an EDC[$\ell$] or TDC[$\ell$] formula ϕ compiled as a deterministic timed automaton $\mathcal{A}_\phi$. We can restrict our focus to a finite MDP abstraction of $\mathcal{T}$ and a finite automaton abstraction of $\mathcal{A}_\phi$. We will make use of the k-bounded digital clock abstraction of PTAs. Our presentation is similar to the classical region-graph construction but, for the sake of clarity of presentation, ignores complications due to strict guards. The strict constraints can be accommodated in a slight generalization of the digital-clock abstraction known as the boundary region graph [11]. The boundary region graph replaces every thick clock region with two regions (mimicking the inf and sup boundaries).

Definition 4.7 (Digital-Clocks Abstraction [16]): The k-bounded digital-clocks abstraction of a probabilistic timed automaton $\mathcal{T} = (Q, q_0, I, \Sigma, X, \gamma, E, \delta, F)$ is a finite Markov decision process $[\![\mathcal{T}]\!]^\bullet_k = (S, s_0, A, T)$ where:

- $S \subseteq Q \times V^k_{\mathbb{N}}$, the set of states;
- $s_0 = (q_0, \mathbf{0}) \subseteq Q \times V$, the set of states;
- $A = \mathbb{N} \times Act$ is the set of (integral timed) *actions*;
- $T : S \times A \to \mathcal{D}(S)$ is such that for $(q, \nu) \in S$ and $(t, a) \in A$, we have $T((q, \nu), (t, a)) = d$ if and only if $\nu + t \in E(q, a)$ and

$$d((q', \nu')) = \sum_{\substack{X \subseteq X \\ (\nu+t)[X:=0]=\nu'}} \delta(q, a)(X, q')$$

for all $(q', \nu') \in S$.

Note that if $\mathcal{T}$ is a timed automaton, then $[\![\mathcal{T}]\!]^\bullet_k$ is a finite automaton.

Theorem 4.8 (Digital-Clock Abstraction [30], [17]): Given a PTA $\mathcal{T} = (Q, q_0, I, \Sigma, X, \gamma, E, \delta, F)$ and a reachability probability objective, there exists a $k \in \mathbb{N}$ such that the optimal reachability probability in $[\![\mathcal{T}]\!]$ equals the optimal reachability probability in $[\![\mathcal{T}]\!]^\bullet_k$, and optimal strategy in $[\![\mathcal{T}]\!]^\bullet_k$ can be used to characterize an optimal policy in $\mathcal{T}$.

Let $\mathcal{T}$ be a PTA and ϕ be a DC specification in EDC[ℓ] or TDC[ℓ] subclasses and $\mathcal{T}_\phi$ be the corresponding timed automaton. Consider the finite MDP $\mathcal{M} = [\![\mathcal{T}]\!]^\bullet_k$ and the finite automaton $\mathcal{A} = [\![\mathcal{T}_\phi]\!]^\bullet_k$. Given an MDP $\mathcal{M} = (S, A, T)$, a labeling function $\mathcal{L} : S \to 2^{Var}$ and a finite automaton $\mathcal{A} = (S', A' = 2^{Var}, T')$, the product $\mathcal{M} \times \mathcal{A}$ is an MDP $(S^\times, A^\times, T^\times)$ where $S^\times = S \times S'$, $A^\times = A$, and $T^\times : S^\times \times A^\times \to \mathcal{D}(S^\times)$ is such that

$$T^\times((s, q), a)(s', q') = \begin{cases} T(s, a)(q) & \text{if } T'(q, \mathcal{L}(s)) = q' \\ 0 & \text{otherwise.} \end{cases}$$

Algorithm 4 ElimMeasure(D) : Eliminate Measurement Constructs from D, $D \in$ TDC[ℓ]

Input: TDC[ℓ] formula $D(\varphi_1, \ldots, \varphi_n)$ over Var, with measurement constructs $\varphi_1, \ldots, \varphi_n$, each $\varphi_i = \ell \bowtie^\bullet_{\mathtt{tic}} c_i$.
Output: DDC formula D' with no measurement constructs, over $Var' = Var \uplus \{E_1, \ldots, E_n\}$

1: **for** each φ_i **do**
2: Introduce a new propositional variable E_i
3: Replace φ_i by $\lceil E_i \rceil^\bullet$
4: **end for**
5: $D' = D[\varphi_i \leftarrow \lceil E_i \rceil^\bullet]$

Algorithm 5 Aut2IRTA(A): Convert automaton A (from Algorithm 2) over Var' to integer reset timed automata $IRTA(A)$

Input: Automaton A over $Var' = Var \uplus \{E_1, \ldots, E_n\}$, such that $L(A) = Untime(L(D'))$ for DDC D' over Var'.
Output: $IRTA(A)$ such that $L(IRTA(A)) = L(D')$.

1: **for** each $\varphi_i = \ell \bowtie^\bullet_{\mathtt{tic}} c_i$ **do**
2: Introduce a clock variable x_i
3: **end for**
4: Add an extra clock variable z to track global integral time
5: **for** each location q in A **do**
6: Add a loop on q which checks $z{=}1$ and resets z to 0
7: **end for**
8: **for** each transition $q \xrightarrow{a} q'$ in A, $a \subseteq Var'$ **do**
9: Replace with 2 transitions $(q, a, \emptyset, z{=}0, q')$ and $(q, a, \emptyset, 0{<}z{<}1, q')$
10: $(q, a, \emptyset, z{=}0, q')$ takes places at globally integral times
11: $(q, a, \emptyset, 0{<}z{<}1, q')$ takes places at globally non integral times
12: **end for**
13: **for** $1 \le i \le 2^n - 1$ **do**
14: $Y_i \leftarrow$ nonempty subset of $\{x_1, \ldots, x_n\}$ s.t. $Y_i \neq Y_j$ for $i \neq j$.
15: **end for**
16: **for** each transition $(q, a, \emptyset, z = 0, q')$ **do**
17: replicate into 2^n transitions
18: $(q, a, Y_1, z{=}0, q'), \ldots, (q, a, Y_{2^n}, z{=}0, q')$,
19: **end for**
20: **for** each transition (q, a, Y, ψ, q') **do**
21: Replace as follows
22: **if** $a \models E_i$ **then**, $Y \leftarrow Y \cup \{x_i\}$
23: **end if**
24: **for** $1 \le i \le n$ **do**
25: **if** $(a \models E_i)$ **then** $\psi = \psi \wedge (x_i \ op \ c)$
26: **end if**
27: **if** $(a \not\models E_i)$ **then** $\psi = \psi \wedge \neg(x_i \ op \ c)$
28: **end if**
29: **end for**
30: **end for**
31: Replacing all transitions of A, IRTA(A) is obtained.

Theorem 4.9 (Correctness [4]): Given an MDP $\mathcal{M}$ and a property automaton $\mathcal{A}$, the optimal probability of satisfaction of $\mathcal{A}$ is equal to the optimal probability of reaching the accepting states in $\mathcal{M} \times \mathcal{A}$. Moreover, optimal reachability probability strategy in $\mathcal{M} \times \mathcal{A}$ gives an optimal (memoryful) strategy in $\mathcal{M}$ to satisfy $\mathcal{A}$ with optimal probability.

Corollary 1: The controller synthesis problem for PTA against EDC$[\ell]$ and TDC$[\ell]$ specifications is decidable.

V. Reinforcement Learning

In this section, we study a learning based algorithm for controller synthesis for unknown PTAs with DC specifications. We say that a PTA $\mathcal{T} = (Q, q_0, \Sigma, X, \gamma, E, \delta, F)$ is "unknown" if the probability distribution of the transition function $\delta : Q \times \Sigma \to \mathcal{D}(2^X \times Q)$ is not explicitly known in advance, but can be sampled. From Theorem 4.9, the controller synthesis problem reduces to optimal reachability problem for a finite MDP. We first show how to solve this problem for finite MDPs in the next subsection, and present our algorithm in section V-B.

A. Optimizing Reachability Probability using Q-learning

The optimal reachability probability and corresponding optimal strategies were introduced in Section II-A. We will use average- and discounted- reward objectives as tools to find the optimal policy for the reachability probability problem. Let $(\mathcal{M} = (S, A, T), \rho)$ be a rewardful MDP. For a strategy σ and state $s \in S$ we define its *average* reward as

$$Avg_\sigma^{\mathcal{M}}(s) = \lim_{N\to\infty} \mathbb{E}_\sigma^{\mathcal{M}}(s) \left\{ \frac{1}{N} \sum_{1\le i\le N} \rho(X_{i-1}, Y_i) \right\};$$

and, for $\lambda \in [0, 1)$, its *discounted* objective as

$$Disct(\lambda)_\sigma^{\mathcal{M}}(s) = \lim_{N\to\infty} \mathbb{E}_\sigma^{\mathcal{M}}(s) \left\{ \sum_{1\le i\le N} \lambda^{i-1} \rho(X_{i-1}, Y_i) \right\}.$$

The optimal average reward $Avg_*^{\mathcal{M}}(s)$ and optimal discounted reward $Disct_*^{\mathcal{M}}(s)$, and corresponding optimal strategies are defined as supremum over all strategies.

The problem of learning an optimal strategy for the reachability probability objective for a finite MDP can be approximated by the problem of computing an optimal strategy for the discounted reward objective. The key idea is that one can modify the reward signal such that once the target states are reached, every step given a reward of 1 unit, while transitions before reaching the target states get a reward of 0 unit. With this construction, it is easy to see that if a trace reaches a target state, its average reward-per-transition is 1, while if it never reaches a target state average reward-per-transition is 0, i.e. the average reward with this modification is equal to the reachability probability for the trace. Hence, a strategy maximizing the average expected reward will also maximize reachability probability. While there are RL algorithms to solve the expected average reward problem, they require *weakly communicating* assumptions [32] on the MDP. It is well-known that for every finite MDP, there exists a discount factor $\lambda < 1$ such that λ-discount optimal strategies are also average optimal. Hence, leaving the discount factor as a hyperparameter, in practice discounted RL algorithms are used to compute average reward optimal strategies. Hence, we focus our RL presentation on discounted objective.

The optimal discounted value $Disct(\lambda)_*^{\mathcal{M}} = V : S \to \mathbb{R}$ in a reward MDP $(\mathcal{M}, \rho)$ can be characterized [27] using the following *Bellman optimality equations*:

$$V(s) = \max_{a\in A(s)} \left\{ \rho(s, a) + \lambda \sum_{s'\in S} p(s' \mid s, a) \cdot V(s') \right\}.$$

Exploiting Banach's fixed point theorem [5], these values can be approximated by the following sequence of iterates:

$$\begin{aligned} Q_{t+1}(s, a) &= \rho(s, a) + \lambda \sum_{s'\in S} p(s' \mid s, a) \cdot V_t(s') \\ V_{t+1}(s) &= \max_{a\in A(s)} Q_{t+1}(s, a). \end{aligned}$$

Let $Q_*(s, a) = \lim_{t\to\infty} Q_t(s, a)$ for all $s, a \in S\times A$. The Q-learning algorithm of Watkins [32] is a simple and popular RL algorithm for discounted reward objectives. The Q-learning algorithm assumes that instead of knowing the MDP beforehand, we are given data samples of the form $(s_t, a_t, r_{t+1}, s_{t+1}) \in S\times A\times\mathbb{R}\times S$ for $t = 0, 1, 2, \ldots$ such that $p(s_{t+1} \mid s_t, a) > 0$ and $\rho(s_t, a_t) = r_{t+1}$. Q-learning iteratively applies the following update to the state-action value function:

$$\begin{aligned} Q_{t+1}(s, a) = (1-\alpha_t) Q_t(s, a) \\ + \alpha_t (r_{t+1} + \lambda \cdot \max_{a'\in A(s)} Q_t(s_{t+1}, a')), \end{aligned}$$

if $(s,a)=(s_t,a_t)$ and $Q_{t+1}(s, a) = Q_t(s, a)$ otherwise; here $0 \le \alpha_t < 1$ is the learning-rate at the step $t \ge 0$. The Q-learning is known to converge to the optimal value in the limit under suitable constraints on the learning rate.

Theorem 5.1 (Convergence [33]): For finite MDPs with bounded rewards $|r_t| \le B$ and learning rates $0 \le \alpha_t < 1$ satisfying

$$\sum_{t=0}^{\infty} \alpha_t = \infty \text{ and } \sum_{t=0}^{\infty} \alpha_t^2 < \infty,$$

we have that $Q_t(s, a) \to Q_*(s, a)$ as $t \to \infty$ for all $s, a \in S\times A$ almost surely.

B. Q-learning for PTAs against DC specifications

Since we do not have access to the states of the PTA, we cannot take the product of the PTA with the automaton. However, we can simulate this product by using the timed automata as a monitor, and provide rewards to the RL agent every time an accepting state is visited. A RL agent maximizing this reward signal will converge towards a policy that maximizes the probability of satisfaction of the DC specification. Our RL scheme is sketched as Algorithm 6.

Algorithm 6 RLDC: RL Scheme for DC Specifications

Input: Access to a simulation engine for the unknown PTA $[\![\mathcal{T}]\!]^{\bullet}_{k} = (S, A, T)$ and property automaton $\mathcal{A} = (Q, \Sigma, \Delta, F)$; algorithm parameters step size α and discount factor $\delta \in (0, 1)$.

Output: Optimal Q values for each state action pair.

Initialize $Q((s, q), (t, a)) = 0$ for all $s \in S$, $q \in Q$ and $t \in \mathbb{N}_k$ and $a \in A$.
for each episode **do**
  $S := ((\ell_0, \mathbf{0}), q_0)$
  for each step of episode **do**
    Choose (t, a) from Q-table using ε-greedy policy
    Take action (t, a) observe (ℓ', ν').
    Set $q' = \Delta(q, \mathcal{L}(\ell'))$.
    Set $R = 1$ if $q' \in F$ else set $R = 0$.
    Set $S' = ((\ell', \nu'), q')$
    $Q(S, A) \mathrel{+}= \alpha[R + \lambda \max_{A'} Q(S', A') - Q(S, A)]$
    Set $S' = S$.
  end for
end for

VI. Case Study

We present here two case studies. The first uses variations of the standard frozen lake benchmark modified to add time delays along with some dense-time specifications expressed in DC. The second case study applies the RL algorithm to a real-world model of a simplified cardiac pacemaker with real-time requirements.

A. Benchmark I. Timed Frozen Lake

Frozen lake (see Figure 2)is a simple grid-world example where the goal of the agent is to move from the starting location (yellow grid) to the goal (green grid) without ever moving to a trap (red grids). At each step, the agent picks one of the cardinal directions as their move. The frozen lake environment is stochastic, where there is a probability the agent will move in the desired direction or to one of the adjacent directions. For example, the agent may select to move south, but due to noisy dynamics may end up moving to the cells to the east or west of the current cell with some fixed but unknown probabilities $(1/3)$. Figure 2 shows three frozen-lake arenas with variable grid sizes used in our case study.

The requirement specifications or rules for each case study are shown in Table I, and they are additive, so Case 2 includes all Case 1 rules. Each case study is progressively more complicated than the previous, requiring the discovery of more subtle strategies. The process of creating a case study starts with the

Table I: Frozen Lake Case Study Requirements

Study	Requirements
1	Standard Frozen Lake with no modifications
2	Adds "delay" action to mimic then no move is made; The agent may not take any action consecutively
3a	Adds time limit; Agent must complete within 60 time units
3b	Same as 3a except probabilities changed to 1/4:1/2:1/4

(a) (b) (c) (d)

Figure 2: Frozen Lake grids used in case study

$pick\_one$	$\lceil n \vee e \vee s \vee w \rceil^{\bullet}$
$no\_conflict$	$(\lceil n \rceil^{\bullet} \Rightarrow \lceil \neg e \wedge \neg s \wedge \neg w \wedge \neg wait \rceil^{\bullet}) \wedge (\lceil e \rceil^{\bullet} \Rightarrow \lceil \neg n \wedge \neg s \wedge \neg w \wedge \neg wait \rceil^{\bullet}) \wedge (\lceil s \rceil^{\bullet} \Rightarrow \lceil \neg n \wedge \neg e \wedge \neg w \wedge \neg wait \rceil^{\bullet}) \wedge (\lceil w \rceil^{\bullet} \Rightarrow \lceil \neg n \wedge \neg e \wedge \neg s \wedge \neg wait \rceil^{\bullet}) \wedge (\lceil wait \rceil^{\bullet} \Rightarrow \lceil \neg n \wedge \neg e \wedge \neg s \wedge \neg w \rceil^{\bullet})$
$\lceil reach\_goal \rceil$	$\Box(\neg goal)^{\frown} true^{\frown} \lceil goal \wedge \neg trap \rceil^{\bullet}$
$no\_consecutive\_req$	$\Box((\lceil n \rceil^{\bullet \frown} true \Rightarrow \ell =^{\bullet}_{n} 1)^{\frown} \lceil \neg n \rceil^{\bullet}) \wedge (\lceil e \rceil^{\bullet \frown} true \Rightarrow (\ell =^{\bullet}_{e} 1)^{\frown} \lceil \neg e \rceil^{\bullet}) \wedge (\lceil s \rceil^{\bullet \frown} true \Rightarrow (\ell =^{\bullet}_{s} 1)^{\frown} \lceil \neg s \rceil^{\bullet}) \wedge (\lceil w \rceil^{\bullet \frown} true \Rightarrow (\ell =^{\bullet}_{w} 1)^{\frown} \lceil \neg w \rceil^{\bullet}))$
trp	$\Box \lceil \neg trap \rceil$
no_con_wait	$\Box(\lceil wait \rceil^{\bullet \frown} true \Rightarrow (\ell =^{\bullet}_{wait} 1)^{\frown} \lceil \neg wait \rceil^{\bullet})$
Specification:	$pick\_one \wedge no\_conflict \wedge reach\_goal \wedge no\_consecutive\_req \wedge trp \wedge no\_con\_wait$

Table II: Frozen Lake Case 2 : EDC$[\ell]$ specification

encoding of the rules specification into a series of DC formulas which were then processed using DCVALID [10] into the Hanoi omega automata format. The automaton was then used as an input to MUNGOJERRIE [24] tool for reinforcement learning as well as computing the probability of the strategies learned. The specification for Case 2 is given in Table II as an example. The results are summarized in Table III

Table III: Frozen Lake Case Study Results

Grid Size	Rule Set	Size of Product	Prob of Satisfaction MC	Prob of Satisfaction RL	Episode Length	Episode Number	RL Train Time(secs)
4x4	1	311	0.780	0.780	1000	300,000	84.0
4x4	2	877	0.780	0.776	1000	300,000	59.2
4x4	3a	138,286	0.388	0.313	1000	75,000,000	10038.3
4x4	3b	138,286	0.454	0.382	1000	75,000,000	11,841.7
5x5	1	545	1.00	0.939	1000	300,000	147.3
5x5	2	1511	1.00	0.997	1000	300,000	72.9
5x5	3a	289,320	0.996	0.788	400	15,000,000	3501.2
6x6	1	820	1.00	0.947	2000	900,000	440.5
6x6	2	2270	1.00	0.982	2000	900,000	227.1
6x6	3a	431,655	0.999	0.876	400	15,000,000	3880.1
8x8	1	1424	1.00	1.00	2000	600,000	550.0
8x8	2	2984	0.23	0.0	1000	75,000,000	22.5

B. Benchmark II: Artificial Pacemaker

Our second case study models a simple cardiac pacemaker operating in VVI [6] mode. Such a device paces only the ventricle while also monitoring the ventricle for intrinsic heart activity. In standard pacing therapy, it is desired to pace the minimum required and rely on the patient's natural heart rhythm when present. This is both physiologically better for the patient and conserves the pacemaker battery for better longevity. In the absence of intrinsic activity, VVI mode outputs a ventricular pacing pulse (VP) at the end of a fixed interval. If heart activity is sensed, the pacemaker needs to decide if the sensed event can be considered a valid contraction, a ventricular sense (VS), or if it came in the earlier, refractory period of the cardiac cycle where the myocardium has not yet recovered from previous activity, a refractory ventricular sense (VR). The VR response makes for an interesting case study. When a VR occurs, the refractory period is reset and will continue to reset on every VR until a VS or VP occurs.

$lowvp$	$\lceil VP \rceil^\bullet \frown true \implies ((\ell =^\bullet_{VP} (pace-1)) \wedge \Box \lceil \neg VP \rceil)$
$onlyref$	$(\ell =^\bullet_{\mathtt{tic}} pace) \wedge$ $\Box((\ell =^\bullet_{\mathtt{tic}} ref) \wedge \Box \lceil \neg VS \wedge \neg VP \rceil \implies ((\ell =^\bullet_{\mathtt{tic}} (ref+1)) \frown \lceil \neg VS \rceil^\bullet))$
$pace\_req$	$((\ell =^\bullet_{\mathtt{tic}} pace) \frown \lceil VP \rceil^\bullet)$
no_pace_req	$(\ell =^\bullet_{\mathtt{tic}} pace-1) \wedge (\Box \lceil \neg VP \rceil)$
$repeatvp$	$\Box((\lceil VP \rceil^\bullet \frown true) \wedge onlyref \implies pace\_req)$
$vstrigger$	$[((\lceil VS \rceil^\bullet \frown true) \wedge onlyref) \implies pace\_req] \wedge [((\lceil VS \rceil^\bullet \frown true \implies no\_pace\_req)]$
$repeatvs$	$\Box[((\ell =^\bullet_{\mathtt{tic}} ref) \wedge \Box([\neg VS \wedge \neg VP])) \implies ((\ell =^\bullet_{\mathtt{tic}} ref+1) \frown vstrigger)]$
$boot$	$((onlyref \implies pace\_req) \wedge (\ell =^\bullet_{\mathtt{tic}} pace-1 \wedge \Box \lceil \neg VP \rceil))$
Specification:	$lowvp \wedge repeatvp \wedge repeatvs \wedge boot$

Table IV: Pacemaker : EDC$[\ell]$ and TDC$[\ell]$ specifications

We designed a set of DC specifications from the VVI requirements detailed in [22]. This is a simple pacemaker model, which will pace at a specified interval, inhibiting that scheduled pace should a non-refractory intrinsic ventricular pace occur. For this model, the pacing interval and refractory period are fixed, and there are no modern features such as rate or activity response. The DC model specification shown in Table IV consists of four parts. The first part $lowvp$ says that the pace actions should not happen more often that a certain time period. The second part $repeatvp$ says that a pace action must be followed by another pace action if there are no non-refractory intrinsic heart events, which is encoded in the formula $onlyref$. The $onlyref$ specification says that if an intrinsic event comes closer than refractory period to another one, then it is not a true intrinsic beat. The third part $repeatvs$ says that an intrinsic beat must be followed by a pacing event at the pacing interval rate unless a non-refractory event occurs during the pacing interval. The fourth part $boot$ states that the pacemaker observes the heart for non-refractory intrinsic heart events and then pace action must occur after pacing interval.

The heart environment is a PTA model written in the PRISM language [20]. The maximum time between two intrinsic heart beats is set to a fixed duration. At the start of each cardiac cycle, the model stochastically selects when during the intrinsic period it will beat. It then counts the time until the beat time is reached and an intrinsic beat signal is generated and the MDP resets the cardiac cycle. The heart MDP also resets immediately and begins a new cardiac cycle whenever the pace signal is asserted by the pacemaker model.

The results of this case study are given in Table V. For demonstration purposes, the pacing intervals and refractory periods were set to small values, easing the processing time. For training, the pacemaker model was not provided with the interval or period and was required to identify those values based on rewards received from the RL algorithm.

Table V: Pacemaker Case Study Results

Case	Refractory Period	V-V Interval	Size of Product	Prob of Satisfaction MC	Prob of Satisfaction RL	Episode Length	Episode Number	RL Train Time(secs)
vvi-2-6	2	6	120	1.00	1.00	1000	100,000	3.78
vvi-3-8	3	8	270	1.00	1.00	1000	100,000	3.14
vvi-4-10	4	10	884	1.00	1.00	1000	250,000	6.65
vvi-5-14	5	14	2882	1.00	1.00	1000	5,000,000	171.3
vvi-6-16	6	16	4360	1.00	0.00	1000	10,000,000	8.85

C. Discussion

In the frozen lake case study, the RL was required to discover that moving in the desired direction was not the best option, in most cases, and using the edge of the board constrained the number of random directions the agent would move. Growing the 4x4 grid by one row and column, as seen in Figure 2b, resulted in a less complicated path to the goal, moving along the top and right grid edges. While the number of moves to win had grown, the simpler path resulted in an increased learning rate, as can be seen when comparing the episodes and time required for learning between the rule 3a cases of the 4x4 and 5x5 cases. The 5x5 case was significantly faster. This improvement is of limited value though as the grid continues to grow. This can be seen with the 8x8 case where the rule preventing consecutive actions (ruleset 2) was sufficient to baffle the tabular Q-Learning algorithm. This test case was unable to find any valid path over 15 million tries leading to a very short run time due to each episode quickly failing to meet a rule. In each case where the RL's probability of satisfaction did not match the model checker, the RL was shown to still be learning and, given additional time, should match the exact values (MC computed using model checking). This can be seen in the simpler cases where the two values matched exactly or nearly exacting.

The pacemaker case study provided a very different challenge. The RL was not given any hint to the specification requirements. Given only a signal when the heart intrinsically paced and a reward when it operated properly, the pacemaker had to discern the pacing interval and refractory period. Table V shows that this was achievable through the number of episodes and time required quickly grew as the number of counts representing a complete pacing interval increased. *It should be noted that the pacemaker model is being used as an exemplar of a practical real-time system with hard timing requirements. Any life sustaining medical device, no matter how coded, must demonstrate to the FDA that it is safe and effective before it can be brought to market.*

VII. Conclusion

This paper presented a learning-based synthesis approach for systems modeled as probabilistic timed automata against two subclasses of DC specifications. The subclasses of DC presented in the paper are expressive enough to capture interesting case studies; moreover, their translation to well behaved subclasses of timed automata has been crucial in the convergence of the proposed RL algorithm.

The theoretical result enables the translation of real-time specifications to scalar rewards in a way that is correct (reward optimization implies optimization of specification satisfaction) and effective (reduce the problem to RL over finite MDPs). This allows one to exploit reward-maximizing learning algorithms to maximize property satisfaction. Unfortunately, when the model is not known, one can not a priori bound the number of learning episodes. We add the caveat, however, that the proposed approach should not be confused with the correct-by-construction synthesis approach where the goal is to design a formally correct controller from specifications. In contrast, our approach must be complemented with rigorous V&V of the learned designs.

REFERENCES

[1] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, Apr. 1994.

[2] R. Alur, L. Fix, and T. A. Henzinger, "Event-clock automata: A determinizable class of timed automata," *Theor. Comput. Sci.*, vol. 211, no. 1-2, pp. 253–273, 1999.

[3] J. An, N. Zhan, X. Li, M. Zhang, and W. Yi, "Model checking bounded continuous-time extended linear duration invariants," in *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC 2018, Porto, Portugal, April 11-13, 2018*, 2018, pp. 81–90.

[4] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[5] S. Banach, "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales," *Fundamenta Mathematicae*, no. 1, pp. 133–181, 1922. [Online]. Available: http://eudml.org/doc/213289

[6] A. D. Bernstein, J.-C. Daubert, R. D. Fletcher, D. L. Hayes, B. Lüderitz, D. W. Reynolds, M. H. Schoenfeld, and R. Sutton, "The revised NASPE/BPEG generic code for antibradycardia, adaptive-rate, and multisite pacing," *Pacing and clinical electrophysiology*, vol. 25, no. 2, pp. 260–264, 2002.

[7] P. Bouyer, T. Brihaye, V. Bruyère, and J.-F. Raskin, "On the optimal reachability problem on weighted timed automata," *Formal Methods in System Design*, vol. 31, no. 2, pp. 135–175, 2007.

[8] A. K. Bozkurt, Y. Wang, M. M. Zavlanos, and M. Pajic, "Control synthesis from linear temporal logic specifications using model-free reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 10 349–10 355.

[9] Z. Chaochen, M. R. Hansen, and P. Sestoft, "Decidability and undecidability results for duration calculus," in *STACS 93, 10th Annual Symposium on Theoretical Aspects of Computer Science, Würzburg, Germany, February 25-27, 1993, Proceedings*, 1993, pp. 58–68.

[10] "DCVALID–a tool for modelchecking Duration Calculus Formulae," accessed: 05/28/2021. [Online]. Available: http://www.tcs.tifr.res.in/~pandya/dcvalid103.html

[11] V. Forejt, M. Kwiatkowska, G. Norman, and A. Trivedi, "Expected reachability-time games," *Theor. Comput. Sci.*, vol. 631, pp. 139–160, 2016.

[12] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak, "Omega-regular objectives in model-free reinforcement learning," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2019, pp. 395–412, lNCS 11427.

[13] ——, "Faithful and effective reward schemes for model-free reinforcement learning of omega-regular objectives," in *Automated Technology for Verification and Analysis*, D. V. Hung and O. Sokolsky, Eds. Cham: Springer International Publishing, 2020, pp. 108–124.

[14] ——, "Model-free reinforcement learning for stochastic parity games," in *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, ser. LIPIcs, I. Konnov and L. Kovács, Eds., vol. 171. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 21:1–21:16.

[15] M. Hasanbeig, A. Abate, and D. Kroening, "Cautious reinforcement learning with logical constraints," in *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, A. E. F. Seghrouchni, G. Sukthankar, B. An, and N. Yorke-Smith, Eds. International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 483–491. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/3398761.3398821

[16] T. A. Henzinger, Z. Manna, and A. Pnueli, "What good are digital clocks?" in *Automata, Languages and Programming*, W. Kuich, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 545–558.

[17] M. Jurdzinski, M. Z. Kwiatkowska, G. Norman, and A. Trivedi, "Concavely-priced probabilistic timed automata," in *CONCUR 2009 - Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings*, ser. Lecture Notes in Computer Science, M. Bravetti and G. Zavattaro, Eds., vol. 5710. Springer, 2009, pp. 415–430.

[18] H. Kopetz, "Time-triggered real-time computing," *Annu. Rev. Control.*, vol. 27, no. 1, pp. 3–13, 2003. [Online]. Available: https://doi.org/10.1016/S1367-5788(03)00002-6

[19] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, 2003.

[20] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Computer Aided Verification (CAV)*, Jul. 2011, pp. 585–591, lNCS 6806.

[21] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston, "Automatic verification of real-time systems with discrete probability distributions," *Theoretical Computer Science*, vol. 282, pp. 101–150, June 2002.

[22] S. Q. R. Laboratory, "Pacemaker system specification," http://sqrl.mcmaster.ca/_SQRLDocuments/PACEMAKER.pdf, 2007, accessed: 2020-11-1.

[23] A. Lavaei, F. Somenzi, S. Soudjani, A. Trivedi, and M. Zamani, "Formal controller synthesis for continuous-space mdps via model-free reinforcement learning," in *11th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2020, Sydney, Australia, April 21-25, 2020*. IEEE, 2020, pp. 98–107.

[24] "Mungojerrie–formal reinforcement learning," accessed: 05/28/2021. [Online]. Available: https://plv.colorado.edu/wwwmungojerrie/

[25] P. K. Pandya, "Specifying and deciding quantified discrete-time duration calculus formulae using dcvalid: An automata theoretic approach," in *Proceedings of RTTOOLS*, 2001.

[26] ——, "Interval duration logic: Expressiveness and decidability," *Electron. Notes Theor. Comput. Sci.*, vol. 65, no. 6, pp. 254–272, 2002. [Online]. Available: https://doi.org/10.1016/S1571-0661(04)80480-8

[27] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons, Inc., 1994.

[28] D. Sadigh, E. Kim, S. Coogan, S. S. Sastry, and S. A. Seshia, "A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications," in *IEEE Conference on Decision and Control (CDC)*, Dec. 2014, pp. 1091–1096.

[29] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Jan. 2016.

[30] J. Sproston, "Discrete-time verification and control for probabilistic rectangular hybrid automata," in *2011 Eighth International Conference on Quantitative Evaluation of SysTems*, 2011, pp. 79–88.

[31] P. V. Suman, P. K. Pandya, S. N. Krishna, and L. Manasa, "Timed automata with integer resets: Language inclusion and expressiveness," in *Formal Modeling and Analysis of Timed Systems, 6th International Conference, FORMATS 2008, Saint Malo, France, September 15-17, 2008. Proceedings*, 2008, pp. 78–92.

[32] R. S. Sutton and A. G. Barto, *Reinforcement Learnging: An Introduction*, 2nd ed. MIT Press, 2018.

[33] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.