Context-Aware Temporal Logic for Probabilistic Systems

Mahmoud Elfar[0000−0002−5579−1255], Yu Wang[0000−0002−0431−1039], and Miroslav Pajic[0000−0002−5357−0117]

Duke University, Durham NC 27708, USA
{mahmoud.elfar,yu.wang94,miroslav.pajic}@duke.edu
http://cpsl.pratt.duke.edu

Abstract. In this paper, we introduce the context-aware probabilistic temporal logic (CAPTL) that provides an intuitive way to formalize system requirements by a set of PCTL objectives with a context-based priority structure. We formally present the syntax and semantics of CAPTL and propose a synthesis algorithm for CAPTL requirements. We also implement the algorithm based on the PRISM-games model checker. Finally, we demonstrate the usage of CAPTL on two case studies: a robotic task planning problem, and synthesizing error-resilient scheduler for micro-electrode-dot-array digital microfluidic biochips.

Keywords: Markov-decision process, temporal logic, model checking, probabilistic systems, synthesis

1 Introduction

The correct-by-design paradigm in Cyber-Physical Systems (CPS) has been a central concept during the design phase of various system components. This paradigm requires the abstraction of both the system behavior and the design requirements [22,23]. Typically, the system behavior is modeled as a discrete Kripke structure, with nondeterministic transitions representing various actions or choices that need to be resolved. In systems where probabilistic behavior is prevalent, formalisms such as Markov decision processes (MDPs) are best suited. The applications of correct-by-design synthesis paradigm span CPS fields such as robot path and behavior planning [6,18], smart power grids [24], safety-critical medical devices [15], and autonomous vehicles [25].

Temporal logic (TL) can be utilized to formalize CPS design requirements. For example, Linear Temporal Logic (LTL) [2] is used to capture safety and reachability requirements over Boolean predicates defined over the state space. Similarly, computation tree logic (CTL) [2] allows for expressing requirements over all computations branching from a given state. Probabilistic computation tree logic (PCTL) can be viewed as a probabilistic variation of CTL to reason about the satisfaction probabilities of temporal requirements.

The choice of which TL to use is both a science and an art. Nevertheless, fundamental factors include expressiveness (i.e., whether the design requirements of interest can be expressed by the logic), and the existence of model checkers that can verify the system model against the design requirement, synthesize winning strategies, or generate counterexamples. Although prevalent TLs can be inherently expressive, two notions are oftentimes overlooked, namely, how easy it is to correctly formalize the design requirements, and whether existing model checkers are optimized for such requirements. The more complex it becomes to formalize a given requirement, the more likely it is that human error is introduced in the process.

In particular, we focus in this paper on requirements that are naturally specified as a set of various objectives with an underlying priority structure. For instance, the objective of an embedded controller might be focused on achieving a primary task. However, whenever the chances of achieving such task fall below a certain threshold, the controller shall proceed with a fail-safe procedure. Such requirement, while being easy to state and understand, can prove challenging when formalized for two reasons. First, multiple objectives might be involved with a priority structure, i.e., one objective takes priority over another. Second, the context upon which the objectives are switched is of probabilistic nature, i.e., it requires the ability to prioritize objectives based on probabilistic invariants.

To this end, in this work we consider the problem of modeling and synthesis of CPS modeled as MDPs, with context-based probabilistic requirements, where a context is defined over probabilistic conditions. We tackle this problem by introducing the context-aware probabilistic temporal logic (CAPTL). CAPTL provides intuitive means to formalize design requirements as a set of objectives with a priority structure. For example, a requirement can be defined in terms of primary and secondary objectives, where switching from the former to the latter is based upon a probabilistic condition (i.e., a context). The ability to define context as probabilistic conditions sets CAPTL apart from similar TLs.

In addition to providing the syntax and semantics of CAPTL for MDPs, we investigate the problem of synthesizing winning strategies based on CAPTL requirements. Next, we demonstrate how the synthesis problem can be reduced to a set of PCTL-based synthesis sub-problems. Moreover, for deterministic CAPTL requirements with persistence objectives, we propose an optimized synthesis algorithm. Finally, we implement the algorithm on top of PRISM-games [19], and we show experimental results for two case studies where we synthesize a robotic task planner, and an error-resilient scheduler for microfluidic biochips.

The rest of this section discusses related work. Preliminaries and a motivating example are provided in Sec. 2. In Sec. 3 we introduce the syntax and semantics of CAPTL. The CAPTL-based synthesis problem is introduced in Sec. 4, where we first explore how a CAPTL requirement can be approached using PCTL, followed by our proposed synthesis algorithm. For evaluation, we consider two case studies in Sec. 5. Finally, we conclude the paper in Sec. 6. Full proofs can be found in the extended version of this paper [9].

Related Work. The problem of multi-objective model checking and synthesis has been studied in literature, spanning both MDPs and stochastic games, for various properties, including reachability, safety, probabilistic queries, and reward-based requirements [11,13,14]. Our work improves upon the multi-objective synthesis paradigm by enabling priorities over the multiple objectives as we will show in Sec. 2. One prevalent workaround is to define multiple reward structures, where states are assigned tuples of real numbers depicting how favorable they are with respect to multiple criteria. The synthesis problem is then reduced to an optimization problem over either a normalized version of the rewards (i.e., assigning weights), or one reward with logical constraints on the others [1,7]. Results are typically presented as Pareto curves, depicting feasible points in the reward space [14]. Our work differs in two aspects. First, we use probabilities as means to define priorities rather than reward structures. Second, the mechanics needed to define context-based priorities are an integral part of CAPTL.

Perhaps the closest notion to our context-based prioritization scheme are probabilistic invariant sets (PIS) [17]. Both CAPTL and PIS involve the identification of state-space subsets that maintain a probability measure within specific bounds. While prevalent in the field of probabilistic programs [3], PIS was not considered in the field of CPS synthesis, despite the fact that (non-probabilistic) invariant sets are used in controller design [4]. The problem of merging strategies for MDPs that correspond to different objectives has been investigated [5,27]. Our approach, however, is primarily focused on formalizing the notion of context-based priorities within the specification logic itself rather than altering the original model. While one can argue that PCTL alone can be used to define priorities by utilizing nested probabilistic operators, the nesting is typically limited to qualitative operators [20]. In contrast, CAPTL relaxes such limitation by allowing quantitative operators as well. Moreover, CAPTL-based synthesis provides an insight into which objective is being pursued at a given state.

2 Problem Setting

Preliminaries. For a measurable event E, we denote its probability by $\Pr(E)$. The powerset of A is denoted by $\mathcal{P}(A)$. We use $\mathbb{R}$, $\mathbb{N}$ and $\mathbb{B}$ for the set of reals, naturals and Booleans, respectively. For a sequence or a vector π, we write $\pi[i]$, $i \in \mathbb{N}$, to denote the i-th element of π.

We formally model the system as an MDP. MDPs feature both probabilistic and nondeterministic transitions, capturing both uncertain behaviors and nondeterministic choices in the modeled system, respectively. We adopt the following definition for a system model as an MDP [2].

Definition 1 (System Model). *A system model is an MDP* $\mathcal{M} = (S, Act, \mathbf{P}, s_0, AP, L)$ *where* S *is a finite set of states;* Act *is a finite set of actions;* $\mathbf{P} : S \times Act \times S \to [0, 1]$ *is a transition probability function s.t.* $\sum_{s' \in S} \mathbf{P}(s, a, s') \in \{0, 1\}$ *for* $a \in Act$*;* s_0 *is an initial state;* AP *is a set of atomic propositions; and* $L : S \to \mathcal{P}(AP)$ *is a labeling function.*

Given a system $\mathcal{M}$, a *path* is a sequence of states $\pi = s_0 s_1 \ldots$, such that $\mathbf{P}(s_i, a_i, s_{i+1}) > 0$ where $a_i \in Act(s_i)$ for all $i \geq 0$. The trace of π is defined as $trace(\pi) = L(s_0)L(s_1)\cdots$. We use $FPath_{\mathcal{M},s}$ ($IPath_{\mathcal{M},s}$) to denote the set of all finite (infinite) paths of $\mathcal{M}$ starting from $s \in S$. We use $Paths_{\mathcal{M},s}$ to denote the set of all finite and infinite paths starting from $s \in S$. If $\mathbf{P}(s,a,s') = p$ and $p > 0$, we write $s \xrightarrow{a,p} s'$ to denote that, with probability p, taking action a in state s will yield to state s'. We define the *cardinality* of $\mathcal{M}$ as $|\mathcal{M}| = |S| + |\mathbf{P}|$, where $|\mathbf{P}|$ is the number of non-zero entries in $\mathbf{P}$.

A *strategy* (also known as a policy or a scheduler) defines the behavior upon which nondeterministic transitions in $\mathcal{M}$ are resolved. A *memoryless* strategy uses only the current state to determine what action to take, while a *memory-based* strategy uses previous states as well. We focus in this work on pure memoryless strategies, which are shown to suffice for PCTL reachability properties [2].

Definition 2 (Strategy). *A (pure memoryless)* strategy *of* $\mathcal{M} = (S, Act, \mathbf{P}, s_0, AP, L)$ *is a function* $\sigma : S \to Act$ *that maps states to actions.*

By composing $\mathcal{M}$ and σ, nondeterministic choices in $\mathcal{M}$ are resolved, reducing the model to a *discrete-time Markov chain* (DTMC), denoted by $\mathcal{M}^\sigma$. We use $\mathrm{Pr}^\sigma_{\mathcal{M},s}$ to denote the probability measure defined over the set of infinite paths $IPath^\sigma_{\mathcal{M},s}$. The function $Reach(\mathcal{M}, s, \sigma)$ denotes the set of reachable states in $\mathcal{M}$ starting from $s \in S$ under strategy σ, while $Reach(\mathcal{M}, s)$ denotes the set of all reachable states from s under any strategy.

We use *probabilistic computation tree logic* (PCTL) to formalize system objectives as temporal properties with probabilistic bounds, following the grammar

$$\Phi ::= \top \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathbb{P}_J[\varphi], \quad \varphi ::= \mathsf{X}\Phi \mid \Phi \mathsf{U} \Phi \mid \Phi \mathsf{U}^{\leq k} \Phi,$$

where $J \subseteq [0,1]$, and X and U denote the *next* and *until* temporal modalities, respectively. Other derived modalities include $\lozenge$ (*eventually*), $\Box$ (*always*), and W (*weak until*). Given a system $\mathcal{M}$ and a strategy σ, the PCTL satisfaction semantics over $s \in S$ and $\pi \in Paths^\sigma_{\mathcal{M},s}$ is defined as follows [2,12]:

$$\begin{array}{ll}
s,\sigma \models a & \Leftrightarrow a \in L(s) \\
s,\sigma \models \neg\Phi & \Leftrightarrow s \not\models \Phi \\
s,\sigma \models \Phi_1 \wedge \Phi_2 & \Leftrightarrow s \models \Phi_1 \wedge s \models \Phi_2 \\
s,\sigma \models \mathbb{P}_J[\varphi] & \Leftrightarrow \Pr\{\pi \mid \pi \models \varphi\} \in J \\
\pi,\sigma \models \mathsf{X}\Phi & \Leftrightarrow \pi[1] \models \Phi \\
\pi,\sigma \models \Phi_1 \mathsf{U} \Phi_2 & \Leftrightarrow \exists j \geq 0. \big(\pi[j] \models \Phi_2 \wedge (\forall 0 \leq k < j.\, \pi[k] \models \Phi_1)\big) \\
\pi,\sigma \models \Phi_1 \mathsf{U}^{\leq n} \Phi_2 & \Leftrightarrow \exists 0 \leq j \leq n. \big(\pi[j] \models \Phi_2 \wedge (\forall 0 \leq k < j.\, \pi[k] \models \Phi_1)\big)
\end{array}$$

PCTL can be extended with *quantitative queries* of the form $\mathbb{P}_{\min}[\varphi]$ ($\mathbb{P}_{\max}[\Phi]$) to compute the minimum (maximum) probability of achieving φ [12,26], i.e.,

$$\mathbb{P}_{\min}[\varphi] = \inf_{\sigma \in \Sigma} \mathrm{Pr}^\sigma_{\mathcal{M},s}\left(\{\pi \mid \pi \models \varphi\}\right), \ \mathbb{P}_{\max}[\varphi] = \sup_{\sigma \in \Sigma} \mathrm{Pr}^\sigma_{\mathcal{M},s}\left(\{\pi \mid \pi \models \varphi\}\right).$$

We will denote such queries as $\mathbb{P}_{\mathrm{opt}}$ (read: optimal), where $\mathrm{opt} \in \{\max, \min\}$.

Motivating Example. Consider the simple grid-world shown in Fig. 1(left). The robot can move between rooms through doorways where obstacles can be probabilistically encountered (e.g., closed doors), requiring the robot to consume more power. The robot state is captured as a tuple $s : (\mathsf{g}, \mathsf{h}, \mathsf{x}, \mathsf{y})$, where $\mathsf{g} \in \{on, sleep, error\}$ is the robot's status, $\mathsf{h} \in \{0, 1, \ldots, 10\}$ is the robot's battery level, and x and y are its current coordinates. As shown in Fig. 1(right), the system can be modeled as $\mathcal{M} = (S, Act, \mathbf{P}, s_0, AP, L)$, where $Act = \{\mathsf{N}, \mathsf{S}, \mathsf{E}, \mathsf{W}, \mathsf{sleep}, \mathsf{error}\}$, and $s_0 = (0, 10, 1, 1)$. Suppose that the main objective for the robot is to reach the

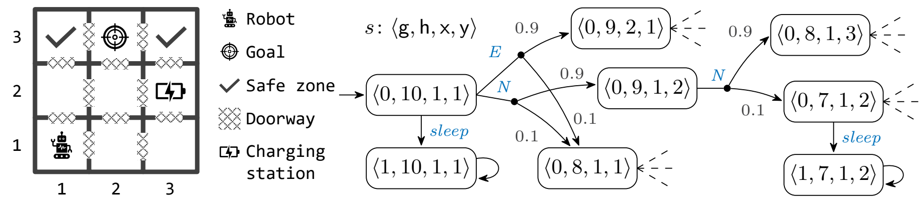

Fig. 1. A motivating example of a robot (left) and part of its model (right).

goal with a charge $h > 3$ (objective A). However, if the probability of achieving objective A is less than 0.8, the robot should prioritize reaching the charging station and switch to *sleep* mode (objective B). Moreover, if the probability of achieving objective B falls below 0.7, the robot should stop and switch to *err* mode, preferably in one of the safe zones (objective C).

Now let us examine how such requirements can be formalized. Let $\varphi_A = \Diamond(goal \wedge (h > 3) \wedge on)$, $\varphi_B = \Diamond(chrg \wedge (h > 3) \wedge on)$, and $\varphi_C = \Diamond(error)$. One can use PCTL to capture each objective separately as the reachability queries $\Phi_A = \mathbb{P}_{\max}[\varphi_A]$, $\Phi_B = \mathbb{P}_{\max}[\varphi_B]$, and $\Phi_C = \mathbb{P}_{\max}[\varphi_C]$. A multi-objective query $\Phi_1 = \Phi_A \vee \Phi_B \vee \Phi_C$ does not capture the underlying priority structure in the original requirements. In fact, an optimal strategy for Φ_1 always chooses the actions that reflect the objective with the highest probability of success, resulting in a strategy where the robot simply signals an error from the very initial state. Similarly, the use of $\Phi_2 = \mathbb{P}_{\max}[\varphi_A \mathsf{W} \varphi_B]$ does not provide means to specify the context upon which switching from φ_A to φ_B occurs. Attempts featuring multi-objective queries with nested operators, such as $\Phi_3 = \mathbb{P}_{\max}[\varphi_A \wedge \mathbb{P}_{\max \geq 0.8}[\varphi_A]] \vee \mathbb{P}_{\max}[\varphi_B \wedge \mathbb{P}_{\max < 0.8}[\varphi_A]]$, have several drawbacks. First, correctly formalizing the requirement is typically cumbersome and hard to troubleshoot. Second, to the best of our knowledge, nested queries in the form of $\mathbb{P}_{\mathrm{opt} \in J}$ are not supported by model checkers. Third, the semantics of the formalized requirement is potentially different from the original one. For instance, Φ_3 allows the system to pursue φ_A even after switching to φ_B if the probability of achieving φ_A rises again above 0.8 — a behavior that was not called for in the original requirement.

Consequently, in this paper we focus on two problems: the formalization of PCTL objectives with an underlying context-based priority structure, and the synthesis of strategies for such objectives. The first problem is addressed by introducing CAPTL in Sec. 3, while the second is addressed in Sec. 4. We will use this motivating example as a running one throughout the rest of this paper.

3 Context-Aware Temporal Logic

CAPTL Syntax. CAPTL features two pertinent notions, namely, objectives and contexts. Let $\mathcal{M}$ be our system model, and let Ξ be the set of all possible PCTL path formulas defined for $\mathcal{M}$. In CAPTL, we define an *objective* q as a conjunctive optimization query $q = \bigwedge_{i=1}^{m} \mathbb{P}_{\mathrm{opt}}\left[\varphi_i\right],\ \varphi_i \in \Xi,\ m > 0$. When $m > 1$, q resembles a multi-objective optimization query in the conjunctive form. Otherwise, in the simplest form where $m = 1$, q is a single-objective query.

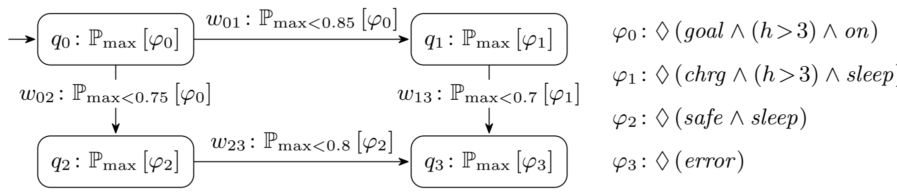

Fig. 2. The CAPTL requirement for the running example.

A *context* $w_{\langle q,q'\rangle}$ marks a state where switching from objective q to objective q' is required. Formally, we define a context w over Ξ as a set of satisfaction queries in the disjunctive normal form $w = \bigvee_{j=1}^{n} \bigwedge_{i=1}^{m} \mathbb{P}_{\mathrm{opt}\in J_{ij}}[\varphi_{i,j}]\,,\ \varphi_{ij} \in \Xi,\ J \subseteq [0,1]\,.$ Intuitively, in a state where $w_{\langle q,q'\rangle}$ is satisfied, the system switches from q to q'. Notice that the context definition utilizes the operator $\mathbb{P}_{\mathrm{opt}\in J_{ij}}$ with an interval, i.e., a context is evaluated at a given state as a boolean value in $\mathbb{B}$. In contrast, the objective definition utilizes the operator $\mathbb{P}_{\mathrm{opt}}$ without intervals, i.e., a quantitative optimization query that can return a numerical value in $[0,1]$.

A *CAPTL requirement* defines a set of objectives to be satisfied, in addition to a set of contexts, representing the probabilistic conditions upon which objectives are prioritized. Formally, we define the syntax of a CAPTL requirement as follows.

Definition 3 (CAPTL Requirement). *Given a set of PCTL path formulas* Ξ, *a* CAPTL requirement *is a tuple* $\mathcal{A} = (Q, W, \Xi, \hookrightarrow, q_0)$ *where*

- $Q \subset \{\bigwedge_{i=1}^{m} \mathbb{P}_{\mathrm{opt}}[\varphi_i] \mid \varphi_i \in \Xi\}$ *is a finite nonempty set of objectives over* Ξ,
- $W \subset \left\{\bigvee_{j=1}^{n} \bigwedge_{i=1}^{m} \mathbb{P}_{\mathrm{opt}\in J_{ij}}[\varphi_{i,j}] \mid \varphi_{ij} \in \Xi, J_{ij} \subseteq [0,1]\right\}$ *is a set of contexts,*
- $\hookrightarrow \subseteq Q \times W \times Q$ *is a conditional transition relation, and*
- $q_0 \in Q$ *is an initial objective.*

In a CAPTL requirement $\mathcal{A}$, each state $q \in Q$ represents an objective, i.e., an optimization query to be satisfied. The conditional transition relation $\hookrightarrow$ defines how objectives are allowed to change. For instance, if $q \overset{w}{\hookrightarrow} q'$, a shorthand for $(q, w, q') \in \hookrightarrow$, then the objectives are switched from q to q' if w is satisfied. Notice that contexts are used as labels for the conditional transition relation. In the rest of this paper, we will overload the notation and use $W : Q \to \mathcal{P}(W)$ to denote the set of contexts emerging from a given objective. We will also use $Q(q, w) = q'$ to denote that objective q has a context w that leads to q'.

Example 1. For the running example, Fig. 2 shows an example of a CAPTL requirement $\mathcal{A}$ where $Q = \{q_0, q_1, q_2, q_3\}$, $W = \{w_{01}, w_{02}, w_{13}, w_{23}\}$, and $\hookrightarrow = \{\langle q_0,w_{01},q_1\rangle, \langle q_0,w_{02},q_2\rangle, \langle q_1,w_{13},q_3\rangle, \langle q_0,w_{23},q_3\rangle\}$. The requirement starts by prioritizing $q_0 = \mathbb{P}_{\max}[\varphi_0]$. If $\mathbb{P}_{\max}[\varphi_0] \in [0.75, 0.85)$, the context w_{01} becomes true, and by executing $q_0 \overset{w_{01}}{\hookrightarrow} q_1$, $q_1 = \mathbb{P}_{\max}[\varphi_1]$ is prioritized. Similarly, if $\mathbb{P}_{\max}[\varphi_0] \in [0, 0.75)$, w_{02} becomes true, executing $q_0 \overset{w_{02}}{\hookrightarrow} q_2$ where $q_2 = \mathbb{P}_{\max}[\varphi_2]$ is prioritized. Notice that objectives can have a single context, e.g., $W(q_1) = \{w_{13}\}$; multiple contexts, e.g., $W(q_0) = \{w_{01}, w_{02}\}$; or none, e.g., $W(q_3) = \varnothing$.

CAPTL Semantics for MDPs. We progressively define CAPTL semantics for MDPs by first defining the satisfaction semantics for objectives and contexts. Let $q = \mathbb{P}_{\max}[\varphi]$ be the objective at state s, and let Σ be the set of all strategies for $\mathcal{M}$. We say that $s, \sigma^* \models q$ if $\sigma^* \in \Sigma$ such that

$$\Pr_{\mathcal{M}}^{\sigma^*,s} = \sup_{\sigma\in\Sigma} \Pr^{\sigma}_{\mathcal{M},s}\left(\left\{\pi \in \mathit{Paths}^{\sigma}_{\mathcal{M},s} \mid \pi \models \varphi\right\}\right). \tag{1}$$

In that case, we call σ^* a *local strategy*, i.e., an optimal strategy w.r.t. $\langle q, s\rangle$.

Definition 4 (Local Strategy). *Let $q_i = \mathbb{P}_{\text{opt}}[\varphi_i]$ be an objective. A* local (optimal) strategy *for $\langle q_i, s_i\rangle$ is a strategy $\sigma_{\langle q_i,s_i\rangle} \in \Sigma$ such that*

$$\Pr_{\mathcal{M},s_i}^{\sigma_{\langle q_i,s_i\rangle}} = \operatorname*{opt}_{\sigma\in\Sigma} \Pr^{\sigma}_{\mathcal{M},s_i}\left(\left\{\pi \in \mathit{Paths}^{\sigma}_{\mathcal{M},s_i} \mid \pi \models \varphi_i\right\}\right)$$

Next, let $(q, w, q') \in \hookrightarrow$, where $w = \mathbb{P}_{\leq c}[\varphi]$. Let $s_k \in \mathit{Reach}(\mathcal{M}, s, \sigma^*)$, where σ^* is the local strategy for $\langle q, s\rangle$. We say that $s_k \models w$ if

$$\sup_{\sigma\in\Sigma} \Pr^{\sigma}_{\mathcal{M},s_k}\left(\left\{\pi \in \mathit{Paths}^{\sigma}_{\mathcal{M},s_k} \mid \pi \models \varphi\right\}\right) \leq c. \tag{2}$$

Note that contrary to (1), the set of paths $\{\pi\}$ in (2) is *not* limited to those induced by the local strategy σ^*. Moreover, if $\exists\pi = s \ldots s_i \ldots s_k \in \mathit{FPath}^{\sigma}_{\mathcal{M},s}$ s.t. $s_i \models w$, and $s_i \not\models w$ for all $i < k$, then s_k is called a *switching state*, i.e., the first state on a path π to satisfy w, triggering a switch from q to q'.

Definition 5 (Switching Set). *Let $q = \mathbb{P}_{\text{opt}}[\varphi]$ and $\sigma^* \in \Sigma$ such that $s_0, \sigma^* \models q$. The corresponding* switching set *$S_q \subseteq \mathit{Reach}(\mathcal{M}, s_0, \sigma^*)$ is defined as*

$$S_q = \left\{ s_k \mid \exists\pi = s_0 \ldots s_i \ldots s_k \in \mathit{FPath}^{\sigma^*}_{\mathcal{M},s_0} \text{ s.t. } s_i \not\models \bigvee_{w\in W(q)} w,\ \forall i < k;\ s_k \models \bigvee_{w\in W(q)} w \right\}.$$

We use $S_q^{q'}$ to denote the set of switching states from q to q'.

An objective is *active* in a state s if it is being pursued at that state.

Definition 6 (Active Objective). *Let $\mathcal{A} = (Q, W, \Xi, \hookrightarrow, q_0)$ and $\mathcal{M} = (S, \mathit{Act}, \mathbf{P}, s_0, AP, L)$. An* activation function *$g : S \to \mathcal{P}(Q)$ is defined inductively as: (i) $g(s_0) \ni q_0$; and (ii) $g(s) \ni q'$ if $g(s) \ni q$ and $s \in S_q^{q'}$. We say objective $q \in Q$ is* active *at state $s \in S$ if $g(s) \ni q$.*

As captured in Definition 4, local strategies are tied to their respective objectives. Consequently, a local strategy is switched whenever an objective is switched as well, and the new local strategy substitutes its predecessor. We call the set of local strategies a *strategy profile*, and the resulting behavior a *protocol*.

Definition 7 (Protocol). *Let $\mathcal{A} = (Q, W, \Xi, \hookrightarrow, q_0)$ and $\mathcal{M} = (S, \mathit{Act}, \mathbf{P}, s_0, AP, L)$. Given a strategy profile $\sigma = \left\{\sigma_{\langle q,s\rangle} \ldots\right\}$, the induced* (optimal) protocol *is a (partial) function $\Pi : Q \times S \nrightarrow \mathit{Act} \cup \mathcal{P}(W)$ such that*

- *$\Pi(q, s) = \sigma_{\langle q,s\rangle}(s) \in \mathit{Act}$ iff $q \in g(s)$ and $s \notin S_q$; and*
- *$\Pi(q, s) \ni w_{\langle q,q'\rangle}$, where $w_{\langle q,q'\rangle} \in W$, iff $q \in g(s)$ and $s \in S_q^{q'}$.*

Given $\langle q, s\rangle$, a protocol assigns either an optimal action based on the local strategy associated with q, or a context to switch the active objective itself. We will use $\mathfrak{P}$ to denote the set of all possible protocols.

Definition 8 (System-Protocol Composition). *Let $\mathcal{M} = (S, Act, \mathbf{P}, s_0, AP, L)$ and $\varPi : Q \times S \nrightarrow Act \cup \mathcal{P}(W)$ be a compatible protocol. Their composition is defined as $\mathcal{M}^{\varPi} = \left(\hat{Q}, Act \cup W, \hat{\mathbf{P}}, \hat{s}_0, \hat{L}\right)$ where $\hat{Q} \subseteq Q \times S$, $\hat{s}_0 = \langle q_0, s_0 \rangle$, and*

$$\hat{\mathbf{P}}\left(\langle q, s\rangle, a, \langle q', s'\rangle\right) = \begin{cases} \mathbf{P}\left(s, a, s'\right) & \text{if } \varPi(q, s) = a,\ q' = q, \\ 1 & \text{if } \varPi(q, s) = w,\ s' = s,\ q' = Q(q, w), \\ 0 & \text{otherwise.} \end{cases}$$

We now define the CAPTL satisfaction semantics as follows.

Definition 9 (CAPTL Satisfaction Semantics). *Let $\mathcal{A} = (Q, W, \varXi, \hookrightarrow, q_0)$, $\mathcal{M} = (S, Act, \mathbf{P}, s_0, AP, L)$, and $\varPi : Q \times S \nrightarrow Act \cup \mathcal{P}(W)$. The* CAPTL satisfaction semantics *is defined inductively as follows:*

$$\mathcal{M}, \varPi \models\ q \Leftrightarrow \Pr_{\mathcal{M}^{\varPi}}\left(\{\pi \in Paths_{\mathcal{M}^{\varPi}} \mid last(\pi) = \langle q, s'\rangle, s' \models q\}\right) \geqslant 1,$$
$$\mathcal{M}, \varPi \models_c \mathcal{A} \Leftrightarrow \Pr_{\mathcal{M}^{\varPi}}\left(\{\pi \in Paths_{\mathcal{M}^{\varPi}} \mid last(\pi) = \langle q, s'\rangle, s' \models q, q \in Q\}\right) = c,$$
$$\mathcal{M}, \varPi \models\ \mathcal{A} \Leftrightarrow \mathcal{M}, \varPi \models_{\geqslant 1} \mathcal{A}.$$

CAPTL semantics dictate that $\mathcal{M}$ and $\varPi$ satisfy $\mathcal{A}$ if every path $\pi \in Paths_{\mathcal{M}^{\varPi}}$ ends with a state $s \in S$ where $q \ni g(s)$ and $s \models q$, i.e., the system reaches some state s where some objective q is both active and satisfied.

CAPTL Fragments. A CAPTL requirement is *nondeterministic* if for some $q \in Q$, $\exists w_i, w_j \in W(q)$ such that $S_q^{q_i} \cap S_q^{q_j} \neq \varnothing$. That is, at least one objective has two or more contexts that can be active at the same state. If that is not the case, then the CAPTL requirement is *deterministic*. We now identify a fragment of deterministic CAPTL requirements where the following two conditions are met. First, every $q \in Q$ is a quantitative PCTL persistence objective. Second, every $w \in W(q)$ is a qualitative PCTL persistence objective over the same persistence set as in q. This is formally captured in the following definition.

Definition 10 (Persistence CAPTL). *A CAPTL requirement $\mathcal{A} = (Q, W, \varXi, \hookrightarrow, q_0)$ is* persistent *if (i) every $q \in Q$ is of the form $q = \mathbb{P}_{\max}[\Diamond\Box B]$ for some $B \subseteq S$ and (ii) if $W(q) \neq \varnothing$ then for any $w_{\langle q, q_j\rangle} \in W(q)$, it holds that $w_{\langle q, q_j\rangle} = \mathbb{P}_{\max \in J_j}[\Diamond\Box B]$ where (J_j) are disjoint intervals satisfying $\cup_j J_j = [0, c)$ for some $0 < c \leq 1$.*

A persistence CAPTL (P-CAPTL) requirement allows for defining persistence objectives, where each objective maximizes the probability of (i.e., prioritizes) reaching a corresponding persistence set. Contexts in this case can be understood as lower bounds of their respective objectives. That is, an objective is pursued as long as, at any transient state, the probability of achieving such objective does not drop below a certain threshold. The requirement also ensures that at most one context is satisfied at any state, eliminating any nondeterminism in $\mathcal{A}$.

Example 2. Continuing Example 1, Fig. 3 shows the persistence CAPTL requirement for the robot. Notice that all objectives are in the form $\mathbb{P}_{\max}[\Diamond\Box B]$. Also, the intervals $[0.75, 0.85)$ and $[0, 0.75)$ of w_{01} and w_{02}, respectively, are disjoint, hence at most one context in $W(q_0) = \{w_{01}, w_{02}\}$ can be satisfied at any state.

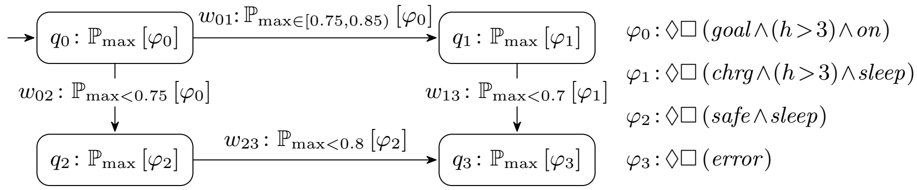

Fig. 3. The persistence CAPTL requirement for the running example.

4 CAPTL-Based Synthesis

In this section we first define the synthesis problem for CAPTL requirements. Next, we examine a general procedure for deterministic CAPTL where the synthesis problem is reduced to solving a set of PCTL-based strategy synthesis problems. Finally, we utilize the underlying structure of persistence properties to propose a synthesis procedure optimized for P-CAPTL requirements.

In the rest of this section, let $\mathcal{M} = (S, Act, \mathbf{P}, s_0, AP, L)$ and $\mathcal{A} = (Q, W, \Xi, \hookrightarrow, q_0)$. We assume that a probabilistic model checker is given (e.g., PRISM-games [19] or UPPAAL STRATEGO [8]) that can accept an MDP-based model $\mathcal{M}$ and a PCTL formula Φ as inputs, and provides the following functions:

- REACH $:: (\mathcal{M}, s) \mapsto R \subseteq S$ returns the set of reachable states $R = Reach(\mathcal{M}, s)$.
- VERIFY $:: (\mathcal{M}, s, \Phi) \mapsto b \in \mathbb{B}$ returns the Boolean value $\top$ if $\mathcal{M}, s \models \Phi$, and returns $\bot$ otherwise.
- SYNTH $:: (\mathcal{M}, s, \Phi) \mapsto (\sigma, c)$ returns a policy $\sigma \in \Sigma$ s.t. $\Pr(\mathcal{M}_s^\sigma \models \Phi) = c$ for some $c \in [0, 1]$.

We also assume that the model checker functions terminate in finite time and return correct answers. We now define the CAPTL synthesis problem as follows.

Definition 11 (CAPTL Synthesis Problem). *Given* $\mathcal{M} = (S, Act, \mathbf{P}, s_0, AP, L)$ *and* $\mathcal{A} = (Q, W, \Xi, \hookrightarrow, q_0)$*, the* CAPTL synthesis problem *seeks to find a protocol* $\Pi : Q \times S \nrightarrow Act \cup W$ *such that* $\mathcal{M}, \Pi \models \mathcal{A}$.

PCTL-Based Approach. The synthesis problem can be reduced to solving a set of PCTL-based synthesis queries as demonstrated in Algorithm 1. Starting with $\langle q_0, s_0 \rangle$, the algorithm verifies whether any context $w \in W(q_0)$ is satisfied, and if true, adds w to the protocol and switches to the next objective. If no context is satisfied, the algorithm synthesizes a local strategy and adds the corresponding optimal action to the protocol.

Proposition 1. *Algorithm 1 terminates; and returns* Π, c *iff* $\mathcal{M}, \Pi \models_c \mathcal{A}$.

Synthesis for P-CAPTL. We now propose a synthesis algorithm optimized for persistence CAPTL. To this end, we show that for a given persistence objective, synthesizing a local strategy in the initial state suffices. In a manner similar to switching states (see Definition 5), we devise a partition of reachable states for every objective. We will use those concepts to define a system-CAPTL composition and show that it is bisimilar to $\mathcal{M}^\Pi$.

Algorithm 1: PCTL-Based Synthesis

Input: $\mathcal{M} = (S, Act, \mathbf{P}, s_0, AP, L)$, $\mathcal{A} = (Q, W, \Xi, \hookrightarrow, q_0)$
Result: Π, c such that $\mathcal{M}, \Pi \models_c \mathcal{A}$
foreach $q \in Q$ **do** $\hat{S}_q \leftarrow \varnothing,\ \bar{S}_q \leftarrow \varnothing$
$\Pi \leftarrow \varnothing,\ \hat{S}_{q_0} \leftarrow \{s_0\},\ q \leftarrow q_0,\ \mathbf{C} \leftarrow \mathbf{0}_{Q\times S} \in [0,1]^{Q\times S},\ \ repeat \leftarrow \top$
while $\hat{S}_q \neq \varnothing$ **do**
  Let $s \in \hat{S}_q,\ \hat{S}_q \leftarrow \hat{S}_q \setminus \{s\},\ \bar{S}_q \leftarrow \bar{S}_q \cup \{s\}$
  while *repeat* **do** *repeat* $\leftarrow \bot$
    foreach $w \in W(q)$ **do**
      if VERIFY $(\mathcal{M}, s, w) = \top$ **then**
        $\Pi \leftarrow \Pi \cup \{(s, q, w)\},\ q \leftarrow Q(q, w),\ repeat \leftarrow \top,$ **break**
  $(\sigma, \mathbf{C}(q, s)) \leftarrow$ SYNTH $(\mathcal{M}; s, q),\ \Pi \leftarrow \Pi \cup \{(s, q, \sigma(s))\}$
  $\hat{S}_q \leftarrow \hat{S}_q \cup \left(Post\,(\mathcal{M}, s, \sigma(s)) \setminus \bar{S}_q\right)$
$c \leftarrow$ VERIFY $\left(\mathcal{M}^{\Pi},\ \langle q_0, s_0\rangle,\ \mathbb{P}\left[\Diamond \bigvee_{q\in Q}(\langle q, s\rangle \wedge \mathbf{C}(q, s) = 1)\right]\right)$

Let $R = Reach(\mathcal{M}, s_0)$. We first note that given $\mathcal{M}$ and $q = \mathbb{P}_{\text{opt}}[\Diamond\Box B]$, existing model checking and synthesis algorithms typically compute a least fixed point (LFP) vector $\mathbf{x}_q \in [0,1]^{|R|}$, where $\mathbf{x}_q[s]$ is the optimal probability of satisfying $\Diamond\Box B$ at state $s \in R$ (e.g., see [2,16]). That is, when SYNTH $(\mathcal{M}, s_0, q)$ is called, $\mathbf{x}_q$ is computed, but only $c = \mathbf{x}_q[s_0]$ is returned (i.e., the value at the initial state). We exploit this fact by implementing a function REACHP :: $(\mathcal{M}, s, q) \mapsto \mathbf{x}_q$ that returns the LFP vector $\mathbf{x}_q$ associated with q.

Lemma 1 (Local Strategy Dominance). *Let* $\mathcal{M} = (S, Act, \mathbf{P}, s_0, AP, L)$ *and* $q = \mathbb{P}_{\max}[\Diamond\Box B]$. *For all* $s \in Reach(\mathcal{M}, s_0)$, $\sigma_{\langle q,s\rangle} = \sigma_{\langle q,s_0\rangle}\big|_{Reach(\mathcal{M},s)}$.

Lemma 1 signifies that a local strategy for q in the initial state (i.e., $\sigma_{\langle q,s_0\rangle}$) subsumes all local strategies for the same probabilistic reachability objective in every $s \in R$. Next, for every $q \in Q$, let us define the following partition of R:

- $R_q^q = \{s \in R \mid \forall w = \mathbb{P}_{\max\in J}[\Diamond\Box B] \in W(q),\ \mathbf{x}_q[s] \notin J\}$, i.e., the states in R where, if q is active, keep pursuing q.
- $R_q^{q'} = \{s \in R \mid \exists w = \mathbb{P}_{\max\in J}[\Diamond\Box B] \in W(q), \mathbf{x}_q[s] \in J,\ Q(q, w) = q'\}$, i.e., the states in R where, if q is active, switch to q'.

Lemma 2 (Partitioning). *Let* $\mathcal{M} = (S, Act, \mathbf{P}, s_0, AP, L)$, $\mathcal{A} = (Q, W, \Xi, \hookrightarrow, q_0)$, *and* $R = Reach(\mathcal{M}, s_0)$. *For every* $q \in Q$, $\bigcup_{q'\in Q} R_q^{q'} = R$; *and* $R_q^{q'} \cap R_q^{q''} = \varnothing$ *for every* $q' \neq q''$.

Proof Sketch. From Definition 10, the intervals $(J_w)_{w\in W(q)}$ are disjoint; hence $(R_q^{q'})_{q'\neq q}$ are disjoint as well, and that $R_q^q = R / \left(\bigcup_{q'\neq q} R_q^{q'}\right)$. □

Example 3. Returning to the P-CAPTL requirement specified in the running example (see Fig. 3), Fig. 4 depicts the partitioning of the state-space based on q_0, q_1, q_2 and q_3. Notice that for any $q \in Q$, the sets $(R_q^{q'})_{q'\in Q}$ are pairwise disjoint, where $\cup_{q'\in Q} R_q^{q'} = Reach(\mathcal{M}, s_0)$. For example, $R_{q_0}^{q_0}$, $R_{q_0}^{q_1}$ and $R_{q_0}^{q_2}$ do not intersect, and their union spans $R = Reach(\mathcal{M}, s_0)$. In this case, $R_{q_0}^{q_3} = \varnothing$ since there is no direct context emerging from q_0 to q_3.

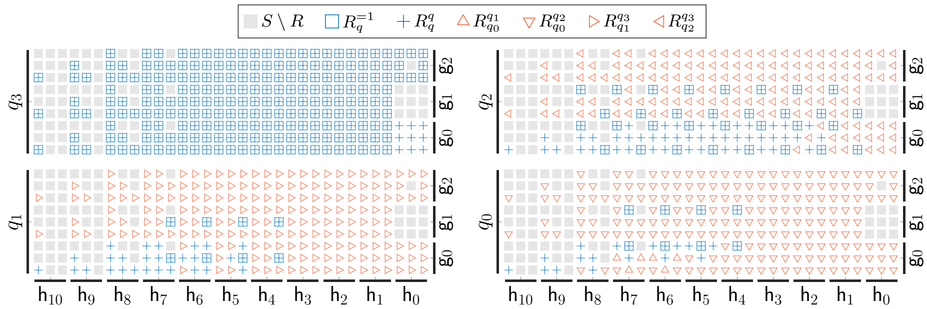

Fig. 4. Partitioning the state-space of the running example using q_0, q_1, q_2, and q_3. For example, $\langle q_1, \mathsf{g}_0, \mathsf{h}_5, 2, 1\rangle = \triangleright$ indicates that $s : \langle g, h, x, y\rangle = \langle 0, 5, 2, 1\rangle \in R^{q_3}_{q_1}$.

Definition 12 (System-CAPTL Composition). *Let* $\mathcal{M} = (S, Act, \mathbf{P}, s_0, AP, L)$*,* $\mathcal{A} = (Q, W, \Xi, \hookrightarrow, q_0)$*, and* $\sigma = \{\sigma_{\langle q,s_0\rangle} \mid q \in Q\}$*. Their* composition *is defined as the automaton* $\mathcal{M}^\sigma_\mathcal{A} = (V, \overline{Act}, \mathbf{P}_v, \rightarrow', v_0, AP, \bar{L})$ *where* $V \subseteq S \times Q \times \Gamma$*, and* $\Gamma = \{①, ②\}$*;* $\overline{Act} = Act \cup W \cup \{\tau\}$*, where* τ *is a stutter action;* $v_0 = \langle s_0, q_0, ②\rangle$*;* $\bar{L} : V \to \mathcal{P}(AP)$ *such that* $\bar{L}(\langle s, q, \gamma\rangle) = L(s)$*; and the transition relation* $\rightarrow'$ *is defined using the following compositional rules:*

$$[\mathrm{R1}]\,\frac{s \xrightarrow{a,p} s' \wedge \sigma_{\langle q,s_0\rangle}(s) = a}{\langle s, q, ①\rangle \xrightarrow{a,p}{}' \langle s', q, ②\rangle} \quad [\mathrm{R2}]\,\frac{s \in R^q_q}{\langle s, q, ②\rangle \xrightarrow{\tau}{}' \langle s, q, ①\rangle} \quad [\mathrm{R3}]\,\frac{s \in R^{q'}_q}{\langle s, q, ②\rangle \xrightarrow{w_{\langle q,q'\rangle}}{}' \langle s, q', ②\rangle}.$$

The rules in Definition 12 are interpreted as follows. The state space V is partitioned into $V_{①}$ (where $\mathcal{M}$ actions are allowed) and $V_{②}$ (where $\mathcal{A}$ actions are allowed), resembling a turn-based 2-player game. [R1] ensures that, if q is active in s, then only the transitions with the optimal action $\sigma_{\langle q,s_0\rangle}(s)$ are allowed. [R2] ensures that, if $s \in R^q_q$, the active objective remains unchanged. If $s \in R^{q'}_q$, however, [R3] enforces switching the active objective to q'. The action τ is a stutter since $\forall v \xrightarrow{\tau} v'$, $\bar{L}(v) = \bar{L}(v')$.

Lemma 3 (Induced DTMC). $\mathcal{M}^\sigma_\mathcal{A}$ *constructed using Definition 12 is a DTMC.*

Lemma 3 dictates that the probability measure $\Pr_{\mathcal{M}^\sigma_\mathcal{A}}$ is well-defined. We will now use the notion of *stutter equivalence* [2] to prove that $\mathcal{M}^\sigma_\mathcal{A}$ is bisimilar to $\mathcal{M}^\Pi$. Basically, two paths π_1 and π_2 are stutter-equivalent, denoted by $\pi_1 \triangleq \pi_2$, if there exists a finite sequence $A_0 \ldots A_n \in (\mathcal{P}(AP))^+$ such that $trace(\pi), trace(\hat{\pi}) \in A_0^+ A_1^+ \ldots A_n^+$, where $A^+ = \{A, AA, \ldots\}$ is the set of finite, non-empty repetitions.

Theorem 1 (Stutter-Equivalence). *Let* $\mathcal{M}$*,* $\mathcal{A}$*, and* $\Pi \in \mathfrak{P}$ *be such that* $\mathcal{M}, \Pi \models \mathcal{A}$*. For every* $\pi \in FPath_{\mathcal{M}^\Pi}$ *there exists* $\hat{\pi} \in FPath_{\mathcal{M}^\sigma_\mathcal{A}}$ *such that* $\pi \triangleq \hat{\pi}$ *and* $\Pr_{\mathcal{M}^\Pi}(\pi) = \Pr_{\mathcal{M}^\sigma_\mathcal{A}}(\hat{\pi})$*. For every* $\hat{\pi} \in FPath_{\mathcal{M}^\sigma_\mathcal{A}}$*, where* $last(\hat{\pi}) \in V_{②}$*, there exists* $\hat{\pi} \in FPath_{\mathcal{M}^\Pi}$ *such that* $\hat{\pi} \triangleq \pi$ *and* $\Pr_{\mathcal{M}^\sigma_\mathcal{A}}(\hat{\pi}) = \Pr_{\mathcal{M}^\Pi}(\pi)$*.*

Proof Sketch. We show that for every execution fragment $\varrho_1 = \langle s, q\rangle \xrightarrow{a,p} \langle s, q'\rangle$ there exists $\hat{\varrho}_1 = \langle s, q, ②\rangle \xrightarrow{\tau} \langle s, q, ①\rangle \xrightarrow{a,p} \langle s', q, ②\rangle$. Moreover, for every $\varrho_2 =$

$\langle s, q\rangle \xrightarrow{w} \langle s', q\rangle$ there exists $\hat{\varrho}_2 = \langle s, q, ②\rangle \xrightarrow{w} \langle s, q', ②\rangle$. Using induction, we show that for every arbitrary execution ϱ there exists $\hat{\varrho}$ such that $\varrho \triangleq \hat{\varrho}$, where

$$\begin{aligned} trace(\varrho) &= (A_0 + A_0A_0)\ (A_1 + A_1A_1)\ \dots\ (A_n + A_nA_n) \in (\mathcal{P}(AP))^+ \\ trace(\hat{\varrho}) &= (A_0A_0)\qquad\quad (A_1A_1)\qquad\quad \dots\ (A_nA_n)\qquad\quad \in (\mathcal{P}(AP))^+ \end{aligned}$$

and $\Pr(\varrho) = \Pr(\hat{\varrho})$. Similarly, the other direction can be shown for every $last(\hat{\varrho})$ that ends with $last(\hat{\varrho}) \in V_{②}$. □

We use Theorem 1 to devise the protocol synthesis procedure summarized in Algorithm 2. In the first part (lines 1–8), the procedure starts by synthesizing a local strategy $\sigma_{\langle q_0, s_0\rangle}$ and obtaining the associated LFP vector $\mathbf{x}_{q_0} \in [0,1]^R$. Next, R is partitioned using $\mathbf{x}_{q_0}$ to obtain $(R^q_{q_0})_{q\in Q}$. If $R^q_{q_0} \neq \varnothing$ for some $q \neq q_0$, the same procedure is repeated for q to obtain $\langle q, s_0\rangle$, $\mathbf{x}_q$ and $(R^{q'}_q)_{q'\in Q}$. In the second part (lines 9–16), three modules are constructed based on Definition 12. The resulting parallel composition constitutes $\mathcal{M}^\sigma_\mathcal{A}$, which mimics a stochastic 2-player game between $\hat{\mathcal{M}}$ (player ①) and $\hat{\mathcal{A}}$ (player ②), where the players' choices are already resolved by $\hat{\sigma}$. Finally, Π is populated by a query that checks for the CAPTL satisfaction condition (line 17), i.e., a state $\langle s, q_i, \gamma\rangle$ is reached where $q_i = \mathbb{P}_{\max}[\Diamond\Box B_i]$ is active, and $\Box B_i$ holds. Notice that, based on the results from Lemma 1, Algorithm 2 synthesizes a local strategy at most once for every $q \in Q$, compared to Algorithm 1 where synthesis is performed at every reachable state.

Theorem 2. *Algorithm 2 terminates; and returns Π, c iff $\mathcal{M}, \Pi \models_c \mathcal{A}$.*

Example 4 (Protocol Synthesis). For the CAPTL requirement in Example 2 (see Fig. 3), Fig. 5 shows a visual representation of the protocol synthesized using Algorithm 2, where blue markers indicate actions in Act, and red markers indicate actions in W. While pursuing q_0, the robot can achieve the task by moving N(▲), N(▲), E(▶) if no obstacles are encountered, or if obstacles are encountered only once while moving E(▶). Switching from q_0 to q_1 via w_{01}(△) occurs in one state $(0, 7, 1, 2)$; while switching from q_0 to q_2 via w_{02}(▽) occurs in four states $(0, 8, 1, 1)$, $(0, 4, 3, 1)$, $(2, 7, 2, 3)$ and $(0, 4, 1, 3)$.

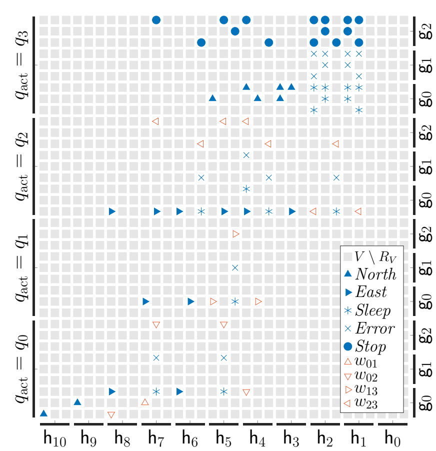

Fig. 5. The protocol synthesized based on the CAPTL requirement in Fig. 3, where $R_V = Reach(\mathcal{M}^\sigma_\mathcal{A}, v_0)$.

5 Experimental Evaluation

We demonstrate the use of CAPTL for protocol synthesis and analysis on two case studies. The first extends the robot task planning problem introduced in Sec. 2. The second considers the problem of synthesizing an error-resilient scheduler

Algorithm 2: Synthesis Procedure for P-CAPTL

Input: $\mathcal{M} = (S, Act, \mathbf{P}, s_0, AP, L)$, $\mathcal{A} = (Q, W, \Xi, \hookrightarrow, q_0)$
Result: Π, c such that $\mathcal{M}, \Pi \models_c \mathcal{A}$

foreach $(q, q') \in Q \times Q$ **do** $R_q^{q'} \leftarrow \varnothing$ // Initialize
$\Pi \leftarrow \varnothing, \quad \hat{Q} \leftarrow \{q_0\}, \quad \bar{Q} \leftarrow \varnothing, \quad R \leftarrow \textsc{Reach}(\mathcal{M}, s_0)$
while $\hat{Q} \neq \varnothing$ **do** // Partition R
  Let $q \in \hat{Q}, \quad \hat{Q} \leftarrow \hat{Q} \setminus \{q\}, \quad \bar{Q} \leftarrow \bar{Q} \cup \{q\}, \; R_q^q \leftarrow R$
  $\sigma_{\langle q,s_0\rangle} \leftarrow \textsc{Synth}\,(\mathcal{M}; s_0, q), \; \mathbf{x}_q \leftarrow \textsc{ReachP}\left(\mathcal{M}, s_0, \sigma_{\langle q,s_0\rangle}\right)$
  foreach $w \in W(q)$ *where* $q' = Q(q, w)$ **do**
    $R_q^{q'} \leftarrow \{s \mid \mathbf{x}_q[s] \in J_w\}, \; R_q^q \leftarrow R_q^q \setminus R_q^{q'}$
    if $R_q^{q'} \neq \varnothing \wedge q' \notin \bar{Q}$ **then** $\hat{Q} \leftarrow \hat{Q} \cup \{q'\}$
construct $\hat{\mathcal{M}}$ *module* **such that** // Construct $\mathcal{M}_{\mathcal{A}}^{\sigma}$
  foreach $[a]\, s \rightarrow p_i : (s_i')$ **do** add $[a]\, s \wedge ① \rightarrow p_i : (s_i') \wedge (②)$
construct $\hat{\mathcal{A}}$ *module* **such that**
  foreach $q \in \bar{Q}$ **do** add $[\tau]\, q_{\text{act}} = q \wedge ② \wedge L(R_q^q; s) \rightarrow (q_{\text{act}} = q) \wedge (①)$
  foreach $q \xrightarrow{w} q'$ **do** add $[w]\, q_{\text{act}}{=}q \wedge ② \wedge L(R_q^{q'}; s) \rightarrow (q_{\text{act}} = q') \wedge (②)$
construct $\hat{\sigma}$ *module* **such that**
  foreach $\sigma_{\langle q,s_0\rangle} \neq \varnothing$ *and* $s \in R$ **do** add $[\sigma_{\langle q,s_0\rangle}(s)]\, q_{\text{act}} = q \wedge s \rightarrow \top$
$\mathcal{M}_{\mathcal{A}}^{\sigma} \leftarrow \hat{\mathcal{M}} \parallel \hat{\mathcal{A}} \parallel \hat{\sigma}$
$(\Pi, c) \leftarrow \textsc{Synth}\left(\mathcal{M}_{\mathcal{A}}^{\sigma}, \langle q_0, s_0, ②\rangle, \mathbb{P}[\bigvee_{q_i \in Q} \lozenge\Box(q_{\text{act}} = q_i) \wedge B_i]\right)$

for digital microfluidic biochips. To this end, we implemented Algorithm 2 in MATLAB on top of a modified version of PRISM-games [19] (v4.4), where ReachP functionality was added. The experiments presented in this section were run on an Intel Core i7 2.6GHz CPU with 16GB RAM.

Robotic Task Planner. Table 1 summarizes the performance results for running Algorithm 2 on various sizes of the running example. Notice that the number of choices in $\mathcal{M}_{\mathcal{A}}^{\sigma}$ always matches the number of states, which agrees with the results from Lemma 3. In the three models, q_0 is always active in s_0, and thus is always verified. As the grid size grows larger, the probability of reaching the goal — and hence satisfying q_0 — becomes lower, dropping below 0.85 at the initial state in both (6×6) and (9×9). As a result, q_1 is never active (and hence is never verified) in the second and third models. We also notice that the total time required to run Algorithm 2 does not necessarily grow as the size of the problem grows. In fact, the total time required for (6×6) and (9×9) is lower than the one for (3×3). This is primarily due to the fact that q_1 is never reached or verified in the second and third models as we described. When comparing the model size for $\mathcal{M}$ and $\mathcal{M}_{\mathcal{A}}^{\sigma}$, we notice that $|\mathcal{M}_{\mathcal{A}}^{\sigma}| < |\mathcal{M}|$, with the difference being in orders of magnitude for larger models. However, the time required to construct $\mathcal{M}_{\mathcal{A}}^{\sigma}$ is longer than the time required to construct $\mathcal{M}$.

MEDA-Biochip Scheduler. We now consider synthesizing error-resilient scheduler for micro-electrode-dot-array (MEDA) digital microfluidic biochips, where we borrow examples from [10,21]. A biochip segment consists of a $\mathsf{W} \times \mathsf{H}$ matrix of on-chip actuators and sensors to manipulate microfluidic droplets, and

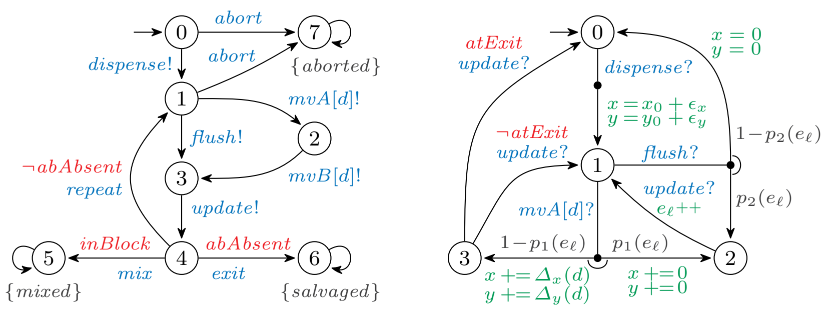

Fig. 6. The MEDA biochip scheduler model (left) and the droplet model (right).

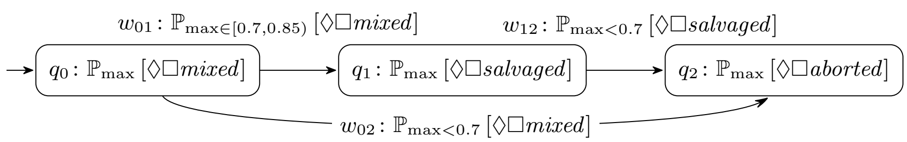

Fig. 7. P-CAPTL requirement for a MEDA-biochip segment scheduler.

is further partitioned into 3×3 blocks. Two reservoirs are used to dispense droplets A and B. Various activation patterns can be applied to manipulate the droplets, including moving (moving droplets individually), flushing (moving both droplets at the same time in the same direction) and mixing (merging two droplets occupying the same block). As the biochip degrades, the actuators become less reliable, and an actuation command may not result in the droplet moving as expected. The probability of an error occurring is proportional to the total number of errors occurred in the same block.

Fig. 6 shows part of the segment scheduler (left) and the droplet (right) models. Initially, the scheduler can dispense both droplets through the *dispense* action, where the droplet location (x, y) can probabilistically deviate from the dispenser location (x_0, y_0) with error ϵ. Subsequently, droplets can be individually manipulated via $mvA[d]$ and $mvB[d]$ actions where d is the direction, or together via *flush*. The probability of successful manipulation $(1 - p(e_\ell))$ depends on both the number of errors within the same block (e_ℓ) and the activation pattern used. The scheduler executes *update* to sense droplet locations and register errors.

The primary task of the scheduler is to perform a mixing operation within the given segment (q_0). However, if the droplets are dispensed and (due to faulty blocks) the probability of a successful mixing operation is below 0.85 (w_{01}), salvaging the dispensed droplets by moving them to an adjacent segment is prioritized (q_1). If the mixing probability drops below 0.7 (w_{02}), or if the salvaging probability drops below 0.7 (w_{12}), the scheduler is to abort the operation (q_2). The aforementioned requirements are formalized using CAPTL as shown in Fig. 7. The set of objectives is $Q = \{q_0, q_1, q_2\}$, and the set of contexts is defined as $W = \{w_{01}, w_{02}, w_{12}\}$. The performance results for running Algorithm 2 on three different segment sizes is reported in Table 1.

Table 1. Protocol synthesis performance results for the robotic task planner (C1) and the MEDA-biochip scheduler (C2). (St.: states, Tr.: transitions, Ch.: choices).

Model	$\mathcal{M}$ Size			$\mathcal{M}^{\sigma}_{\mathcal{A}}$ Size			Construction/Synthesis Time (sec)							
Size	St.	Tr.	Ch.	St.	Tr.	Ch.	$\mathcal{M}$	q_0	q_1	q_2	q_3	$\mathcal{M}^{\sigma}_{\mathcal{A}}$	$q_{\mathcal{A}}$	Total
C1 3×3	233	1,117	745	142	163	142	0.438	0.031	0.029	0.033	0.106	0.557	0.052	25.5
6×6	595	2,692	1,874	159	190	159	0.495	0.041	–	0.083	0.260	0.662	0.112	24.2
9×9	733	3,242	2,278	96	116	96	0.508	0.037	–	0.059	0.313	0.691	0.083	21.9
C2 8×5	2,851	8,269	5,678	2,576	2,929	2,576	1.308	2.348	0.433	3.122	–	17.95	3.585	60.53
11×5	8,498	25,502	17,214	4,167	4,673	4,167	2.013	7.212	1.577	9.928	–	79.77	5.84	149.6
11×8	15,290	47,602	31,316	3,223	3,653	3,223	2.065	12.36	2.536	18.61	–	109.2	4.498	218.5
14×8	61,489	201,469	130,718	1,016	1,339	1,016	4.545	48.07	10.67	68.40	–	289.9	1.289	450.4

6 Conclusion

In this paper we have introduced context-aware probabilistic temporal logic (CAPTL). The logic provides intuitive means to formalize requirements that comprises a number of objectives with an underlying priority structure. CAPTL allows for defining context (i.e., probabilistic conditions) as the basis for switching between two different objectives. We have presented CAPTL syntax and semantics for MDPs. We have also investigated the CAPTL synthesis problem, both from PCTL and CAPTL-based approaches, where we have shown that the latter provides significant performance improvements. To demonstrate our work, we have presented two case studies. As this work has primarily considered CAPTL semantics for MDPs, further investigation is required to generalize the results for stochastic multi-player games. Another research direction involves expanding the results to include PCTL fragments beyond persistence objectives, such as safety, bounded reachability and reward-based objectives.

Acknowledgments. This work was supported in part by the NSF CNS-1652544 and ECCS-1914796, ONR N00014-20-1-2745 and N00014-17-1-2504, as well as AFOSR FA9550-19-1-0169 awards.

References

1. Baier, C., Dubslaff, C., Korenčiak, L., Kučera, A., Řehák, V.: Synthesis of optimal resilient control strategies. In: International Symposium on Automated Technology for Verification and Analysis. pp. 417–434. Springer (2017)
2. Baier, C., Katoen, J.P., Larsen, K.G.: Principles of Model Checking. MIT press (2008)
3. Barthe, G., Espitau, T., Fioriti, L.M.F., Hsu, J.: Synthesizing probabilistic invariants via doob's decomposition. In: International Conference on Computer Aided Verification. pp. 43–61. Springer (2016)
4. Blanchini, F.: Set invariance in control. Automatica **35**(11), 1747–1767 (1999)
5. Boutilier, C., Brafman, R.I., Geib, C.: Prioritized goal decomposition of markov decision processes: Toward a synthesis of classical and decision theoretic planning. In: IJCAI. pp. 1156–1162 (1997)
6. Bozkurt, A.K., Wang, Y., Zavlanos, M.M., Pajic, M.: Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In: Proceedings of 2020 IEEE Int. Conf. on Robotics and Automation (ICRA)
7. Brázdil, T., Kučera, A., Novotnỳ, P.: Optimizing the expected mean payoff in energy markov decision processes. In: International Symposium on Automated Technology for Verification and Analysis. pp. 32–49. Springer (2016)

8. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Uppaal stratego. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 206–211. Springer (2015)
9. Elfar, M., Wang, Y., Pajic, M.: Context-aware temporal logic for probabilistic systems. arXiv e-prints arXiv:2007.05793 (2020), `https://arxiv.org/abs/2007.05793`
10. Elfar, M., Zhong, Z., Li, Z., Chakrabarty, K., Pajic, M.: Synthesis of error-recovery protocols for micro-electrode-dot-array digital microfluidic biochips. ACM Transactions on Embedded Computing Systems (TECS) **16**(5s), 1–22 (2017)
11. Etessami, K., Kwiatkowska, M., Vardi, M.Y., Yannakakis, M.: Multi-objective model checking of markov decision processes. In: Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. pp. 50–65. Springer (2007)
12. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: International School on Formal Methods for the Design of Computer, Communication and Software Systems. pp. 53–113. Springer (2011)
13. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Quantitative multi-objective verification for probabilistic systems. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 112–127. Springer (2011)
14. Forejt, V., Kwiatkowska, M., Parker, D.: Pareto curves for probabilistic model checking. In: International Symposium on Automated Technology for Verification and Analysis. pp. 317–332. Springer (2012)
15. Jiang, Z., Pajic, M., Moarref, S., Alur, R., Mangharam, R.: Modeling and verification of a dual chamber implantable pacemaker. In: Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. pp. 188–203. Springer (2012)
16. Kattenbelt, M., Kwiatkowska, M., Norman, G., Parker, D.: A game-based abstraction-refinement framework for markov decision processes. Formal Methods in System Design **36**(3), 246–280 (2010)
17. Kofman, E., De Doná, J.A., Seron, M.M.: Probabilistic set invariance and ultimate boundedness. Automatica **48**(10), 2670–2676 (2012)
18. Kress-Gazit, H., Lahijanian, M., Raman, V.: Synthesis for robots: Guarantees and feedback for robot behavior. Annual Review of Control, Robotics, and Autonomous Systems **1**, 211–236 (2018)
19. Kwiatkowska, M., Parker, D., Wiltsche, C.: Prism-games: verification and strategy synthesis for stochastic multi-player games with multiple objectives. International Journal on Software Tools for Technology Transfer **20**(2), 195–210 (2018)
20. Lahijanian, M., Andersson, S., Belta, C.: Control of markov decision processes from pctl specifications. In: Proceedings of the 2011 American Control Conference. pp. 311–316. IEEE (2011)
21. Li, Z., Lai, K.Y.T., Yu, P.H., Chakrabarty, K., Pajic, M., Ho, T.Y., Lee, C.Y.: Error recovery in a micro-electrode-dot-array digital microfluidic biochip. In: Proceedings of the 35th International Conference on Computer-Aided Design. pp. 1–8 (2016)
22. Neema, S., Sztipanovits, J., Karsai, G., Butts, K.: Constraint-based design-space exploration and model synthesis. In: International Workshop on Embedded Software. pp. 290–305. Springer (2003)
23. Pajic, M., Mangharam, R., Sokolsky, O., Arney, D., Goldman, J., Lee, I.: Model-driven safety analysis of closed-loop medical systems. IEEE Transactions on Industrial Informatics **10**(1), 3–16 (2012)
24. Puggelli, A., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Robust strategy synthesis for probabilistic systems applied to risk-limiting renewable-energy pricing. In: 2014 International Conference on Embedded Software (EMSOFT). pp. 1–10. IEEE (2014)
25. Seshia, S.A., Sadigh, D., Sastry, S.S.: Formal methods for semi-autonomous driving. In: 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC). pp. 1–5
26. Svoreňová, M., Kwiatkowska, M.: Quantitative verification and strategy synthesis for stochastic games. European Journal of Control **30**, 15–30 (2016)
27. Wiltsche, C.: Assume-Guarantee Strategy Synthesis for Stochastic Games. Ph.D. thesis, University of Oxford (2015)

A Proofs

Proposition 1. *Algorithm 1 terminates; and returns Π, c iff $\mathcal{M}, \Pi \models_c \mathcal{A}$.*

Proof. We break the proof into two parts: termination and correctness.

Termination. We first note that $|S|, |Q|, |W| < \infty$ by definition.

- The **foreach** loop (line 6) terminates either by exhausting all $w \in W(q)$ (which is finite), or by breaking whenever $repeat = \top$. The loop can only run indefinitely if $|W(q)| = \infty$.
- For the inner **while** loop (line 5), the only way to remain indefinitely in that loop is for $repeat = \top$ to always hold, which is only set whenever $\mathcal{M}, s \models w$. However, whenever $\mathcal{M}, s \models w$ holds, q is updated (line 8). Since Q is finite and $\mathcal{A}$ is acyclic, recursion over Q ends in a finite number of loops, ending with a $q \in Q$ where $W(q) = \varnothing$. Hence, $\mathcal{M}, s \models w$ cannot hold indefinitely.
- For the outermost **while** loop (line 3), line 4 dictates that the set S_q shrinks by one state s each and every loop, which is also added to $\bar{S}_q$. Hence, for $S_q \neq \varnothing$ to hold indefinitely for some $q \in Q$, $Post\,(\mathcal{M}, s, \sigma(s)) \setminus \bar{S}_q \neq \varnothing$ must always hold (line 10). However, this mandates that $\bar{S}_q$ can grow indefinitely. Since $|S| < \infty$ by definition, $\bar{S}_q$ cannot grow indefinitely, and the loop eventually terminate in a finite number of iterations.

Correctness. Initially, $s = s_0$ and $q = q_0$. We identify the following cases:

- (a) Case $W(q) = \varnothing$. Then $repeat = \bot$, σ is synthesized such that $\mathcal{M}_s^\sigma \models q$, and $(s, q, \sigma(s))$ is added to Π.
- (b) Case $\forall w \in W(q)$, $\mathcal{M}, s \not\models w$. Then $repeat = \bot$, σ is synthesized such that $\mathcal{M}_s^\sigma \models q$, and $(s, q, \sigma(s))$ is added to Π.
- (c) Case $\exists w \in W(q)$, $\mathcal{M}, s \models w$. Then from Definition 10 we conclude that $\forall \bar{w} \in W(q) \setminus \{w\}$ it holds that $\mathcal{M}, s \not\models \bar{w}$. Consequently, $repeat = \top$, (s, q, w) is added to Π, and q is updated. Since $\mathcal{A}$ is finite and acyclic, the loop eventually halts with condition (a) or (b) becoming true.

□

Lemma 1 (Local Strategy Dominance). *Let $\mathcal{M} = (S, Act, \mathbf{P}, s_0, AP, L)$ and $q = \mathbb{P}_{\max}[\lozenge\Box B]$. For all $s \in Reach(\mathcal{M}, s_0)$, $\sigma_{\langle q,s\rangle} = \sigma_{\langle q,s_0\rangle}\big|_{Reach(\mathcal{M},s)}$.*

Proof. In the first part of the proof, we establish the used notation. In the second part, we show that if $s \in Reach(\mathcal{M}, s_0)$, then the domain of $\sigma_{\langle q,s\rangle}$ is subset of the domain of $\sigma_{\langle q,s_0\rangle}$. In the last part, we show that $\sigma_{\langle q,s\rangle} = \sigma_{\langle q,s_0\rangle}\big|_{Reach(\mathcal{M},s)}$

Notation. For a function $f : A \to B$, we will use $dom(f)$ to denote the domain of f, and $f(a)\downarrow \Leftrightarrow a \in A$, $f(a)\uparrow \Leftrightarrow a \notin A$. We will use $\Sigma_s : S \nrightarrow Act$ to denote the set of all possible (pure memoryless) strategies from state $s \in S$. We assume that for every $\sigma \in \Sigma_s$, $\sigma(s')$ is defined for every $s' \in Reach(\mathcal{M}, s)$.

Well-Definedness. Now, let us consider an arbitrary strategy $\sigma_{s_0} \in \Sigma_{s_0}$. Hence, $dom(\sigma_{s_0}) = Reach(\mathcal{M}, s_0)$. Moreover, consider an arbitrary state $s \in Reach(\mathcal{M}, s_0)$, and an associated strategy $\sigma_s \in \Sigma_s$. In such case, $dom(\sigma_s) = Reach(\mathcal{M}, s)$. Assume we can find a state $s' \in S$ such that $\sigma_s(s') \downarrow$ and $\sigma_{s_0}(s') \uparrow$. The assumptions imply that

$$\frac{\sigma_s(s') \downarrow}{s' \in Reach(\mathcal{M}, s)}, \quad \frac{\sigma_{s_0}(s') \uparrow}{s' \notin Reach(\mathcal{M}, s_0)} \tag{3}$$

However,

$$\frac{s \in Reach(\mathcal{M}, s_0)}{Reach(\mathcal{M}, s) \subseteq Reach(\mathcal{M}, s_0)} \tag{4}$$

Since the inferences from (3) and (4) contradict each other, we conclude that $\sigma_s(s') \downarrow \implies \sigma_{s_0}(s') \downarrow$. That is, for every $s \in Reach(\mathcal{M}, s_0)$, $dom(\sigma_s) \subseteq dom(\sigma_{s_0})$.

Equivalence. We now prove that $\sigma_{\langle q,s\rangle}(s') = \sigma_{\langle q,s_0\rangle}(s')$ for every $s' \in Reach(\mathcal{M}, s)$. To this end, we first recall a well-established result on the existence of memoryless strategies for probabilistic reachability requirements. For $\mathcal{M}$ and $q = \mathbb{P}_{\max}[\Diamond\Box B]$, let $R = Reach(\mathcal{M}, s_0)$. The maximum probability of reaching the target set B from a state $s \in R$ can be formulated as

$$p_{\max}(s, q) = \sup_{\sigma \in \Sigma} \Pr^{\sigma}_{\mathcal{M},s}\left(\left\{s \ldots t_0 t_1 \ldots \in Paths^{\sigma}_{\mathcal{M},s} | \forall i \geq 0, t_i \in B\right\}\right).$$

As the computation of $p_{\max}(s', q)$ at state $s' \in Reach(\mathcal{M}, s_0)$ is independent of the path that lead to s', the optimal action associated with s' is also independent of such path. Therefore, $\sigma_{\langle q,s\rangle}(s') = \sigma_{\langle q,s_0\rangle}(s')$.

□

Lemma 2 (Partitioning). *Let $\mathcal{M} = (S, Act, \mathbf{P}, s_0, AP, L)$, $\mathcal{A} = (Q, W, \Xi, \hookrightarrow, q_0)$, and $R = Reach(\mathcal{M}, s_0)$. For every $q \in Q$, $\bigcup_{q' \in Q} R^{q'}_q = R$; and $R^{q'}_q \cap R^{q''}_q = \varnothing$ for every $q' \neq q''$.*

Proof. For every $q \in Q$, we identify two cases:

- Case $W(q) = \varnothing$. In this case, $R^{q'}_q = \varnothing$ for every $q' \neq q$, and $R^q_q = R \setminus \cap_{q' \neq q} R^{q'}_q = R$. Therefore, the lemma holds.
- Case $W(q) \neq \varnothing$. By definition, $s \in R^q_q$ implies that $\mathbf{x}_q[s] \notin J'$ for every $w_{\langle q,q'\rangle} = \mathbb{P}_{\max \in J'}[\Diamond\Box B] \in W(q)$. Therefore, $R^q_q \cap R^{q'}_q = \varnothing$ holds for every $q' \neq q$ (a). Next, let us assume that we find $s \in R$ such that $s \in R^{q'}_q$ and $s \in R^{q''}_q$, where $q' \neq q''$ and $w_{\langle q,q''\rangle} = \mathbb{P}_{\max \in J''}[\Diamond\Box B] \in W(q)$. This implies that $J' \cap J'' \neq \varnothing$, which contradicts Definition 10. Hence, such s does not exist, and $R^{q'}_q \cup R^{q''}_q = \varnothing$ holds for every $q' \neq q''$(b). From (a) and (b), we conclude that the lemma holds.

□

Lemma 3 (Induced DTMC). *$\mathcal{M}^{\sigma}_{\mathcal{A}}$ constructed using Definition 12 is a DTMC.*

Proof. Assume that there exists a state $v \in V$ with at least two actions $\{\alpha_1, \alpha_2\} \in \overline{Act}(v)$. We identify two disjoint subsets of V, namely, $V_{①} = \{v \in V \mid v = (s, q, ①)\}$ and $V_{②} = \{v \in V \mid v = (s, q, ②)\}$. In case $v = (s, q, ①) \in V_{①}$, only transitions defined by R1 are allowed. Since for any q at most one action $\sigma_{\langle q, s_0 \rangle}(s)$ is allowed, we conclude that $v \notin V_{①}$. In case $v = (s, q, ②) \in V_{②}$, only transitions defined by R2 and R3 are allowed. Our assumption requires that $s \in S_q^q$ and $s \in S_q^{q'}$. Since $S_q^q \cap S_q^{q'} = \varnothing$ by definition, it contradicts with our assumption, hence $v \notin V_{②}$. Consequently, $v \notin V$, which contradicts our assumption. We conclude that $\mathcal{M}_{\mathcal{A}}^{\sigma}$ has no nondeterministic choices. □

Theorem 1 (Stutter-Equivalence). *Let $\mathcal{M}$, $\mathcal{A}$, and $\Pi \in \mathfrak{P}$ be such that $\mathcal{M}, \Pi \models \mathcal{A}$. For every $\pi \in FPath_{\mathcal{M}^{\Pi}}$ there exists $\hat{\pi} \in FPath_{\mathcal{M}_{\mathcal{A}}^{\sigma}}$ such that $\pi \triangleq \hat{\pi}$ and $\Pr_{\mathcal{M}^{\Pi}}(\pi) = \Pr_{\mathcal{M}_{\mathcal{A}}^{\sigma}}(\hat{\pi})$. For every $\hat{\pi} \in FPath_{\mathcal{M}_{\mathcal{A}}^{\sigma}}$, where $last(\hat{\pi}) \in V_{②}$, there exists $\hat{\pi} \in FPath_{\mathcal{M}^{\Pi}}$ such that $\hat{\pi} \triangleq \pi$ and $\Pr_{\mathcal{M}_{\mathcal{A}}^{\sigma}}(\hat{\pi}) = \Pr_{\mathcal{M}^{\Pi}}(\pi)$.*

Proof. The transitions of $\mathcal{M}^{\Pi}$ can be partitioned into two subsets $\rightarrow_{Act}$ and $\rightarrow_{W}$ where the transitions take the forms $\langle s, q \rangle \xrightarrow{a,p} \langle s', q \rangle$ and $\langle s, q \rangle \xrightarrow{w} \langle s, q' \rangle$, respectively. Starting from $\langle s_0, q_0 \rangle$, let us assume that $\langle s_0, q_0 \rangle \xrightarrow{a,p} \langle s_1, q_0 \rangle$, which is based on $\sigma_{\langle s_0, q_0 \rangle}(s_0)$. Similarly, $\mathcal{M}_{\mathcal{A}}^{\sigma}$ exhibits the execution fragment

$$\langle s_0, q_0, ② \rangle \xrightarrow{\tau} \langle s_0, q_0, ① \rangle \xrightarrow{\hat{a},\hat{p}} \langle \hat{s}_1, q_0, ② \rangle.$$

Since $\hat{a} = \sigma_{\langle s_0, q_0 \rangle}(s_0)$, we conclude that $\hat{a} = a$, and hence $\hat{p} = p$ and $\hat{s}_1 = s_1$. From Lemma 1, we know that $\sigma_{\langle s_0, q \rangle}(s) = \sigma_{\langle s, q \rangle}(s)$. Hence, for every execution fragment ϱ_{Act} in $\mathcal{M}^{\Pi}$, where no objective switching occurs, we can find an execution $\hat{\varrho}_{Act}$ in $\mathcal{M}_{\mathcal{A}}^{\sigma}$ such that

$$\begin{aligned}
\varrho_{Act} &= \langle s_0, q_0 \rangle \qquad\qquad\qquad\quad \xrightarrow{a_1,p_1} \langle s_1, q_0 \rangle \quad \xrightarrow{a_2,p_2} \cdots \langle s_i, q_0 \rangle \\
\hat{\varrho}_{Act} &= \langle s_0, q_0, ② \rangle \xrightarrow{\tau} \langle s_0, q_0, ① \rangle \xrightarrow{a_1,p_1} \langle s_1, q_0, ② \rangle \xrightarrow{a_2,p_2} \cdots \langle s_i, q_0, ② \rangle,
\end{aligned}$$

$$\begin{aligned}
trace(\varrho_{Act}) &= \underbrace{L(s_0)}_{1-\text{time}} \quad \underbrace{L(s_1)}_{1-\text{time}} \quad \cdots \quad \underbrace{L(s_i)}_{1-\text{time}} \\
trace(\hat{\varrho}_{Act}) &= \underbrace{L(s_0)L(s_0)}_{2-\text{times}} \underbrace{L(s_1)L(s_1)}_{2-\text{times}} \cdots \underbrace{L(s_i)L(s_i)}_{2-\text{times}},
\end{aligned}$$

$$\begin{aligned}
\Pr(\varrho_{Act}) &= (p_0) \cdot (p_1) \cdots (p_i) \\
\Pr(\hat{\varrho}_{Act}) &= (p_0) \cdot (p_1) \cdots (p_i)
\end{aligned}$$

Therefore, $\varrho_{Act} \triangleq \hat{\varrho}_{Act}$. Now, consider an execution fragment that ends with switching the active objective. In that case, for every execution fragment ϱ we can find $\hat{\varrho}$ such that

$$\begin{aligned}
\varrho_W &= \langle s, q \rangle \quad\;\; \xrightarrow{w_{\langle q, q' \rangle}} \langle s, q' \rangle \\
\hat{\varrho}_W &= \langle s, q, ② \rangle \xrightarrow{w_{\langle q, q' \rangle}} \langle s, q', ② \rangle,
\end{aligned}$$

$$trace(\varrho_W) = \underbrace{L(s)L(s)}_{2-\text{times}}$$
$$trace(\hat{\varrho}_W) = \underbrace{L(s)L(s)}_{2-\text{times}},$$

and

$$\Pr(\varrho_W) = 1$$
$$\Pr(\hat{\varrho}_W) = 1$$

Therefore, $\varrho_W \triangleq \hat{\varrho}_W$. Using induction, we can show that for every arbitrary execution ϱ there exists $\hat{\varrho}$ such that $\varrho \triangleq \hat{\varrho}$, where

$$trace(\varrho) = (A_0 + A_0A_0)\ (A_1 + A_1A_1)\ \dots\ (A_n + A_nA_n) \in (\mathcal{P}(AP))^+$$
$$trace(\hat{\varrho}) = (A_0A_0)\ (A_1A_1)\ \dots\ (A_nA_n) \in (\mathcal{P}(AP))^+$$

and $\Pr(\varrho) = \Pr(\hat{\varrho})$. □

Theorem 2. *Algorithm 2 terminates; and returns Π, c iff $\mathcal{M}, \Pi \models_c \mathcal{A}$.*

Proof. We break the proof into two parts: termination and correctness.

Termination. We first note that $|S|, |Q|, |W| < \infty$ by definition.

- The **foreach** loop (line 6) terminates by exhausting all $w \in W(q)$ (which is finite). The loop can only run indefinitely if $|W(q)| = \infty$.
- For the **while** loop (line 3), line 4 dictates that the set $\hat{Q}$ shrinks by one element q each and every loop, which is also added to $\bar{Q}$. Moreover, line–8 dictates that an objective q' is added to $\hat{Q}$ only if it is not in $\bar{Q}$. That is, every objective $q \in Q$ can be added at most once to $\bar{Q}$. Since $|Q| < \infty$ by definition, the condition $\hat{Q} = \varnothing$ is met in a finite number of iterations.
- Since $|\mathcal{M}| < \infty$, every **construct** code blocks also terminates in a finite number of iterations.

Correctness. From Theorem 1, we know that the paths in $\mathcal{M}^{\Pi}$ and $\mathcal{M}^{\sigma}_{\mathcal{A}}$ are stutter equivalent and probabilistically bisimilar. Hence, for every $q_i \in Q$, the two probability measures

$$\Pr_{\mathcal{M}^{\Pi}}\left(\{\pi \mid \pi \models \Diamond\langle q_i, s\rangle \wedge \mathbf{C}(q_i, s) = 1\}\right),$$
$$\Pr_{\mathcal{M}^{\sigma}_{\mathcal{A}}}\left(\{\hat{\pi} \mid \hat{\pi} \models \Diamond\Box(q_{\text{act}} = q_i) \wedge B_i\}\right)$$

are equivalent. Hence, the algorithm returns a correct answer. □