Mimicking Behaviors in Separated Domains

Giuseppe De Giacomo[1] , **Dror Fried**[2] , **Fabio Patrizi**[1] , **Shufang Zhu**[1]
[1]Sapienza University of Rome
[2]The Open University of Israel
{degiacomo,patrizi,zhu}@diag.uniroma1.it, dfried@openu.ac.il

Abstract

Devising a strategy to make a system mimicking behaviors from another system is a problem that naturally arises in many areas of Computer Science. In this work, we interpret this problem in the context of intelligent agents, from the perspective of LTL_f, a formalism commonly used in AI for expressing finite-trace properties. Our model consists of two separated dynamic domains, $\mathcal{D}_A$ and $\mathcal{D}_B$, and an LTL_f specification that formalizes the notion of *mimicking* by mapping properties on behaviors (traces) of $\mathcal{D}_A$ into properties on behaviors of $\mathcal{D}_B$. The goal is to synthesize a strategy that step-by-step maps every behavior of $\mathcal{D}_A$ into a behavior of $\mathcal{D}_B$ so that the specification is met. We consider several forms of mapping specifications, ranging from simple ones to full LTL_f, and for each we study synthesis algorithms and computational properties.

1 Introduction

Mimicking a behavior from a system A to a system B is a common practice in Computer Science (CS) and Software Engineering (SE). Examples include a robot that has to real-time adapt a human behavior [Mitsunaga *et al.*, 2008], or simultaneous interpretation of a speaker [Yarmohammadi *et al.*, 2013; Zheng *et al.*, 2020]. The challenge in behavior mimicking is twofold. Firstly, a formal specification of *mimicking* is needed; indeed, being potentially different, systems A and B may show substantially different behaviors, not directly comparable, thus a relationship, or *map*, between them must be formally defined to capture when a behavior from A is correctly mimicked by one from B. Secondly, since B ignores what A will do next, B must monitor the actions performed by A and perform its own actions, in such a way that the resulting behavior of B mimics that of A.

In this work, we look at the problem of devising a strategy for mimicking behaviors when the mapping specification is expressed in *Linear Temporal Logic on finite traces* (LTL_f) [De Giacomo and Vardi, 2013], a formalism commonly used in AI for expressing finite-trace properties. In our framework, systems A and B are modeled by two separated dynamic domains, $\mathcal{D}_A$ and $\mathcal{D}_B$, in turn modeled as transition systems, over which there are agents A and B that respectively act, without affecting each other. The *mapping specification* is then a set of LTL_f formulas to be taken in conjunction, called *mappings*, that essentially relate the behaviors of A to those of B. While B has full knowledge of both domains and their states, it has no idea which action A will take next. Nevertheless, in order to perform mimicking, B must respond to every action that A performs on $\mathcal{D}_A$ by performing one action on $\mathcal{D}_B$. As this interplay proceeds, $\mathcal{D}_A$ and $\mathcal{D}_B$ traverse two respective sequences of states (traces) which we call the *behaviors* of A and B, respectively. The process carries on until either A or B (depending on the variant of the problem considered) decides to stop. The mimicking from A has been accomplished correctly, i.e., agent B *wins*, if the resulting traces satisfy the LTL_f mapping specification. Our goal is to synthesize a *strategy* for B, i.e., a function returning an action for B given those executed so far by agent A, which guarantees that B wins, i.e., is able to mimic, respecting the mappings, every behavior of A. We call this the Mimicking Behavior in Separated Domains (MBSD) problem.

The mapping specifications can vary, consequently changing the nature of the mimicking, and consequently the difficulty of synthesizing a strategy for B. We study three different types of mappings. The first is the class of *point-wise* mappings, which establish a sort of local connection between the two separated domains. *Point-wise* mapping specifications have the form $\bigwedge_{i\leq k} \Box(\phi_i \rightarrow \psi_i)$ (see Section 2.2 for proper LTL_f definition) where each ϕ_i is a Boolean property over $\mathcal{D}_A$ and each ψ_i is a Boolean property over $\mathcal{D}_B$. Point-wise mappings indicate invariants that are to be kept throughout the interaction between the agents. In Section 4.1 we give a detailed example of point-wise mappings from the Pac-Man world.

The second class is that of *target* mappings, which relate the ability of satisfying corresponding reachability goals (much in the same fashion as Planning) in the two separate domains. *Target* mapping specifications have the form $\bigwedge_{i\leq k}(\Diamond\phi_i \rightarrow \Diamond\psi_i)$, where ϕ_i and ψ_i are Boolean properties over $\mathcal{D}_A$ and $\mathcal{D}_B$, respectively. Target mappings define objective for A and B and require that if A meets its objective then B must meet its own as well, although not necessarily at the same time. We give a detailed example of target mappings in Section 5.1, from the Rubik's cube world. The last class is that of *general* LTL_f mappings. A general LTL_f map-

ping specification has the form of an arbitrary LTL_f formula Φ with properties over $\mathcal{D}_A$ and $\mathcal{D}_B$.

Our objective is to characterize solutions for strategy synthesis for mimicking behaviors under the types of mapping specifications described above, from both the algorithmic and the complexity point of view. The input we consider includes both domains $\mathcal{D}_A$ and $\mathcal{D}_B$, and the mapping specification. Since it is common to focus on problems in which either of the two is fixed (e.g. [De Giacomo and Rubin, 2018]), we provide solutions in terms of: *combined complexity*, where neither the size of the domain nor that of the mapping specification are fixed; *mapping complexity*, where domains' size are fixed but mapping specification's varies; and *domain complexity*, where the mapping specification's size is fixed but domains' vary.

For our analysis, we formalize the problem as a two-player game between agent A (Player 1) and agent B (Player 2) over a game graph that combines both domains $\mathcal{D}_A$ and $\mathcal{D}_B$, with the winning objective varying in the classes discussed above. We start with *point-wise* mappings where A decides when to stop and derive a solution in the form of a winning strategy for a safety game in PTIME wrt combined, mapping and domain complexity. The scenario becomes more complex for *target* mappings, where the agent B decides when to stop, and where some objectives met during the agent's interplay must be recorded. We devise an algorithm exponential in the number of constraints, and show that the problem is in PSPACE for combined and mapping complexity, and PTIME in domain complexity. To seal the complexity of the problem, we provide a PSPACE-hardness proof for combined complexity, already for simple acyclic graph structures. For domains whose transitions induce a tree-like structure, however, we show that the problem is still in PTIME for combined, mapping and domain complexity. Finally, we show that the problem with *general* LTL_f mapping specifications is in 2EXPTIME for combined and mapping complexity, due to the doubly-exponential blowup of the DFA construction for LTL_f formulas, and is PTIME in domain complexity.

The rest of the paper goes as follows. In Section 2 we give preliminaries, and we formalize our problem in Section 3. We give detailed examples and analyses of point-wise and target mapping specifications in Sections 4 and 5 respectively. We discuss solution for general mapping specifications in Section 6. Then we provide a more detailed discussion about related work in Section 7, and conclude in Section 8.[1]

2 Preliminaries

We briefly recall preliminary notions that will be used throughout the paper.

2.1 Boolean Formulas

Boolean (or propositional) formulas are defined, as standard, over a set of propositional variables (or, simply, *propositions*) $Prop$, by applying the Boolean connectives $\wedge$ (and), $\vee$ (or) and $\neg$ (not). Standard abbreviations are $\rightarrow$ (implies), $true$ (also denoted $\top$) and $false$ (also denoted $\bot$). A proposition $p \in Prop$ occurring in a formula is called an *atom*, a *literal* is an atom or a negated atom $\neg p$, and a *clause* is a disjunction of literals. A Boolean formula is in Conjunctive Normal Form (CNF), if it is a conjunction of clauses. The size of a Boolean formula φ, denoted $|\varphi|$, is the number of connectives occurring in φ. A Quantified Boolean Formula (QBF) is a Boolean formula, all of whose variables are universally or existentially quantified. A QBF formula is in Prenex Normal Form (PNF) if all quantifiers occur in the prefix of the formula. True Quantified Boolean Formulas (TQBF) is the language of all QBF formulas in PNF that evaluate to $true$. TQBF is known to be PSPACE-complete.

2.2 LTL_f Basics

Linear Temporal Logic over finite traces (LTL_f) is an extension of propositional logic to describe temporal properties on finite (unbounded) traces [De Giacomo and Vardi, 2013]. LTL_f has the same syntax as LTL, one of the most popular logics for temporal properties on infinite traces [Pnueli, 1977]. Given a set of propositions $Prop$, the formulas of LTL_f are generated by the following grammar:

$$\varphi ::= p \mid (\varphi_1 \wedge \varphi_2) \mid (\neg\varphi) \mid (\bigcirc\varphi) \mid (\varphi_1 \,\mathcal{U}\, \varphi_2)$$

where $p \in Prop$, $\bigcirc$ is the *next* temporal operator and $\mathcal{U}$ is the *until* temporal operator, both are common in LTL_f. We use common abbreviations for *eventually* $\Diamond\varphi \equiv true\,\mathcal{U}\,\varphi$ and *always* as $\Box\varphi \equiv \neg\Diamond\neg\varphi$.

A *word* over $Prop$ is a sequence $\pi = \pi_0\pi_1\cdots$, s.t. $\pi_i \subseteq 2^{Prop}$, for $i \geq 0$. Intuitively, π_i is interpreted as the set of propositions that are $true$ at instant i. In this paper we deal only with *finite*, nonempty words, i.e., $\pi = \pi_0 \cdots \pi_n \in (2^{Prop})^+$. $last(\pi)$ denotes the last instant (index) of π.

Given a finite word π and an LTL_f formula φ, we inductively define when φ is $true$ on π at instant $i \in \{0, \ldots, last(\pi)\}$, written $\pi, i \models \varphi$, as follows:

- $\pi, i \models p$ iff $p \in \pi_i$ (for $p \in Prop$);
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$;
- $\pi, i \models \bigcirc\varphi$ iff $i < last(\pi)$ and $\pi, i+1 \models \varphi$;
- $\pi, i \models \Box\varphi$ iff $\forall j.i \leq j \leq last(\pi)$ and $\pi, j \models \varphi$;
- $\pi, i \models \Diamond\varphi$ iff $\exists j.i \leq j \leq last(\pi)$ and $\pi, j \models \varphi$;
- $\pi, i \models \varphi_1 \,\mathcal{U}\, \varphi_2$ iff $\exists j.i \leq j \leq last(\pi)$ and $\pi, j \models \varphi_2$, and $\forall k.i \leq k < j$ we have that $\pi, k \models \varphi_1$.

  In this paper, we make extensive use of $\Box\varphi$ and $\Diamond\varphi$.

We say that $\pi \in (2^{Prop})^+$ *satisfies* an LTL_f formula φ, written $\pi \models \varphi$, if $\pi, 0 \models \varphi$. For every LTL_f formula φ defined over $Prop$, we can construct a Deterministic Finite Automaton (DFA) $\mathcal{F}_\varphi$ that accepts exactly the traces that satisfy φ [De Giacomo and Vardi, 2013]. More specifically, $\mathcal{F}_\varphi = (2^{Prop}, Q, q_0, \eta, acc)$, where 2^{Prop} is the alphabet of the DFA, Q is the finite set of states, $q_0 \in Q$ is the initial state, $\eta : Q \times 2^{Prop} \rightarrow Q$ is the transition function, and $acc \subseteq Q$ is a set of accepting states.

[1] The proofs are omitted due to the sake of brevity. Please refer to (https://arxiv.org/abs/2205.09201) for an extended version.

2.3 Two-player Games

A *(turn-based) two-player game* models a game between two players, Player 1 ($P1$) and Player 2 ($P2$), formalized as a pair $\mathcal{G} = (\mathcal{A}, W)$, with $\mathcal{A}$ the *game arena* and W the *winning objective*. The arena $\mathcal{A} = (U, V, u_0, \alpha, \beta)$ is essentially a bipartite-graph, where:

- U is a finite set of $P1$ nodes;
- V is a finite set of $P2$ nodes;
- $u_0 \in U$ is the initial node;
- $\alpha \subseteq U \times V$ is the transition relation of $P1$;
- $\beta \subseteq V \times U$ is the transition relation of $P2$.

Intuitively, a token initially in u_0 is moved in turns from nodes in U to nodes in V and vice-versa. $P1$ moves when the token is in a node $u \in U$, by choosing a destination node $v \in V$ for the token, such that $(u, v) \in \alpha$. $P2$ acts analogously, when the token is in a node $v \in V$, by choosing a node $u \in U$ according to β. Thus, $P1$ and $P2$ alternate their moves, with $P1$ playing first, until at some point, after $P2$ has moved, the game stops. As the token visits the nodes of the arena, it defines a sequence of alternating U and V nodes called *play*. If, when the game stops, the play meets W, then $P2$ wins, otherwise $P1$ wins.

Formally, a *play* (of $\mathcal{A}$) $\rho = \rho_0 \cdots \rho_n \in (U \cup V)^+$ is a finite, nonempty sequence of nodes such that:

- $\rho_0 = u_0$;
- $(\rho_i, \rho_{i+1}) \in \alpha$, for i even;
- $(\rho_i, \rho_{i+1}) \in \beta$, for i odd;
- n is even (which implies, by α and β, that $\rho_n \in U$).

Let $Plays_{\mathcal{A}}$ be the set of all plays of $\mathcal{A}$ and let $last(\rho) = n$ be the last position (index) of play ρ. $\rho|_U = \rho_0\rho_2 \cdots \rho_n$ is the *projection of ρ on U*. and $\rho|_V = \rho_1\rho_3 \cdots \rho_{n-1}$ is the *projection* of ρ on V. The *prefix* of ρ ending at the i-th state is denoted as $\rho^i = \rho_0 \cdots \rho_i$.

The *winning objective* W is a (compact) representation of a set of plays, called *winning plays*. $P2$ *wins* if the game produces a winning play, otherwise $P1$ wins. A *strategy* for $P2$ is a function $\sigma : V^+ \to U$, which returns a $P1$ node $u \in U$, given a finite sequence of $P2$ nodes. A strategy σ is said to be *memory-less* if, for every two sequences of nodes $w = w_0 \cdots w_n$ and $w' = w'_0 \cdots w'_m \in V^+$, whenever $w_n = w_m$, it holds that $\sigma(w) = \sigma(w')$; in other words, the move returned by σ is a function of the last node in the sequence. A play ρ is *compatible* with a $P2$ strategy σ if $\rho_{i+1} = \sigma(\rho^i|_V)$, for $i = 0, \ldots, last(\rho) - 1$. A $P2$ strategy σ is *winning* in $\mathcal{G} = (\mathcal{A}, W)$, if every play ρ compatible with σ is winning.

In this paper we consider two classes of games. The first class is that of *reachability games* in which for a set $g \subseteq U$ of $P1$ nodes, $W = \mathrm{Reach}(g)$, where $\mathrm{Reach}(g)$ (*reachability objective*) is the set of plays containing at least one node from g. Formally $\mathrm{Reach}(g) = \{\rho \in Plays_{\mathcal{A}} \mid \text{ there exists } k.0 \leq k \leq last(\rho) : \rho_k \in g\}$.

The second class is that of *safety games*, in which again for a set $g \subseteq U$ of $P1$ nodes, $W = \mathrm{Safe}(g)$, where $\mathrm{Safe}(g)$ (*safety objective*) is the set of plays where all $P1$ nodes are from g. Formally, $\mathrm{Safe}(g) = \{\rho \in Plays_{\mathcal{A}} \mid$ for all even $k.0 \leq k \leq last(\rho) : \rho_k \in g\}$. Both reachability and safety games can be solved in PTIME in the size of $\mathcal{G}$, and if there is a winning strategy for $P2$ in $\mathcal{G}$ then, and only then, there is a winning memory-less strategy for $P2$ in $\mathcal{G}$ [Martin, 1975].

3 Mimicking Behaviors in Separated Domains

The problem of mimicking behaviors involves two agents, A and B, each operating in its own domain, $\mathcal{D}_A$ and $\mathcal{D}_B$ respectively, and requires B to "correctly" mimic in $\mathcal{D}_B$, the behavior (i.e., a trace) exhibited by A in $\mathcal{D}_A$. The notion of "correct mimicking" is formalized by a *mapping specification*, or simply *mapping*, which is an LTL$_f$ formula, specifying when a behavior of A correctly maps into one of B. The agents alternate their moves on their respective domains, with A starting first, until one of the two decides to stop. Only one agent A and B, designated as the *stop* agent, has the power to stop the process, and can do so only after both A and B have moved in the last turn. The mapping constraint is evaluated only when the process has stopped.

The dynamic domains where agents operate are modeled as labelled transition systems.

Definition 1 (Dynamic Domain)**.** *A* dynamic domain *over a finite set $Prop$ is a tuple $\mathcal{D} = (S, s_0, \delta, \lambda)$, s.t.:*

- *S is the finite set of domain states;*
- *$s_0 \in S$ is the initial domain state;*
- *$\delta \subseteq \mathcal{S} \times \mathcal{S}$ is the transition relation;*
- *$\lambda : \mathcal{S} \mapsto 2^{Prop}$ is the state-labeling function.*

With a slight abuse of notation, for every state $s \in S$, we define the set of *possible successors* of s as $\delta(s) = \{s' \mid (s, s') \in \delta\}$. $\mathcal{D}$ is deterministic in the sense that given s, the agent operating in $\mathcal{D}$ can select the transition leading to the next state s' from those available in $\delta(s)$. Without loss of generality, we assume that $\mathcal{D}$ is *serial*, i.e., $\delta(s) \neq \emptyset$ for every state $s \in \mathcal{S}$. A *finite trace* of $\mathcal{D}$ is a sequence of states $\tau = s_0 \cdots s_n$ s.t. $s_{i+1} \in \delta(s_i)$, for $i = 0, \ldots, n-1$. *Infinite traces* are defined analogously, except that $i = 0, \ldots, \infty$. By $|\tau|$ we denote the length of τ, i.e., the (possibly infinite) number of states it contains. In the following, we simply use the term *trace* for a finite trace, and explicitly specify when it is infinite.

We next model the problem of mimicking behaviors by two dynamic systems over disjoint sets of propositions, together with an LTL$_f$ formula specifying the mapping, and the designation of the *stop* agent.

Definition 2. *An instance of the* Mimicking Behaviors in Separated Domains *(MBSD) problem is a tuple $\mathcal{P} = (\mathcal{D}_A, \mathcal{D}_B, \Phi, Ag_{stop})$, where:*

- *$\mathcal{D}_A = (S, s_0, \delta^A, \lambda^A)$ is a dynamic domain over $Prop^A$;*
- *$\mathcal{D}_B = (T, t_0, \delta^B, \lambda^B)$ is a dynamic domain over $Prop^B$, with $Prop^A \cap Prop^B = \emptyset$;*
- *Φ is the* mapping specification*, i.e., an LTL$_f$ formula over $Prop^A \cup Prop^B$;*
- *$Ag_{stop} \in \{A, B\}$ is the designated* stop agent*.*

Intuitively, a solution to the problem is a *strategy* for agent B that allows B to step-by-step map the observed behavior of agent A into one of its behaviors, in such a way that the mapping specification is satisfied, according to the formalization provided next.

Formally, a *strategy* for agent B is a function $\sigma : (S)^+ \rightarrow T$ which returns a state of $\mathcal{D}_B$, given a sequence of states of $\mathcal{D}_A$. Observe that this notion is fully general and is defined on *all* $\mathcal{D}_A$'s state sequences, even non-traces. Among such strategies, we want to characterize those that allow B to satisfy the mapping specification by executing actions only on $\mathcal{D}_B$.

We say that a strategy σ is *executable* in $\mathcal{P}$ if:

- $\sigma(s_0) = t_0$;
- $\sigma(\tau^A)$ is defined on every trace τ^A of $\mathcal{D}_A$;
- for every trace $\tau^A = s_0 \cdots s_n$ of $\mathcal{D}_A$, the sequence $\tau^B = \sigma(s_0)\sigma(s_0 s_1) \cdots \sigma(s_0 s_1 \cdots s_n)$ is a trace of $\mathcal{D}_B$ (of same length as that of τ^A).

When σ is executable, the trace τ^B as above is called the *trace induced by σ on τ^A*, and denoted as $\tilde{\sigma}(\tau^A)$.

For two traces $\tau^A = s_0 \cdots s_n$ and $\tau^B = t_0 \cdots t_n$ of $\mathcal{D}_A$ and $\mathcal{D}_B$, respectively, we define their *joint trace label*, denoted $\lambda(\tau^A, \tau^B)$ as the word over $2^{Prop^A \cup Prop^B}$ s.t. $\lambda(\tau^A, \tau^B) = (\lambda^A(s_0) \cup \lambda^B(t_0)) \cdots (\lambda^A(s_n) \cup \lambda^B(t_n))$. In words, $\lambda(\tau^A, \tau^B)$ is the word obtained by joining the labels of the states of τ_A and τ_B at same positions.

We can now characterize solution strategies.

Definition 3. *A strategy σ is a solution to an MBSD problem instance $\mathcal{P} = (\mathcal{D}_A, \mathcal{D}_B, \Phi, Ag_{stop})$, if σ is executable in $\mathcal{P}$ and either:*

1. *$Ag_{stop} = A$ and every trace τ^A of $\mathcal{D}_A$ is s.t. $\lambda(\tau^A, \tilde{\sigma}(\tau^A)) \models \Phi$; or*
2. *$Ag_{stop} = B$ and every infinite trace τ^A_∞ of $\mathcal{D}_A$ has a finite prefix τ^A s.t. $\lambda(\tau^A, \tilde{\sigma}(\tau^A)) \models \Phi$.*

The definition requires that the strategy σ be executable in $\mathcal{P}$, i.e., that σ returns an executable move for B, whenever A performs an executable move. Then, two cases are identified, which correspond to the possible designations of the stop agent. In case 1, the stop agent is A. In this case, since A can stop at any time point (unknown in advance by B), B must be able to *continuously* (i.e., step-by-step) mimic A's behavior, otherwise A could stop at a point where B fails to mimic. Case 2 is slightly different, as B can choose when to stop. In this case, σ must prescribe a sequence of moves, in response to A's, such that Φ is eventually (as opposed to continuously) satisfied, at which point B can stop the execution. Seen differently, σ must prevent A from moving indefinitely, over an infinite horizon (without B ever being able to mimic A).

4 Mimicking Behaviors with Point-wise Mapping Specifications

In this section, we explore mimicking specifications that are of *point-wise* nature. This setting requires that B, while mimicking A, constantly satisfies certain conditions, which can be regarded as *invariants*. Such a requirement is formally captured by the following specification, where φ_i and ψ_i are Boolean formulas over $\mathcal{D}_A$ and $\mathcal{D}_B$, respectively:

$$\varphi = \bigwedge_{i=1}^{k} \Box(\varphi_i \rightarrow \psi_i).$$

We first provide an illustrative example that demonstrates the use of point-wise mappings, then explore algorithmic and complexity results.

4.1 Point-wise Mapping Specifications in the Pac-Man World

In the popular game Pac-Man, the eponymous character moves in a maze to eat all the candies. Four erratic ghosts, Blinky, Pinky, Inky and Clyde, wander around, threatening Pac-Man, which cannot touch them or looses (we neglect the special candies with which Pac-Man can fight the ghosts). The ghosts cannot eat the candies. In the real game, the maze is continuous but, for simplicity, we consider a grid model where cells are identified by two coordinates. Also, we imagine a variant of the game where the ghosts can walk through walls. Pac-Man wins the stage when it has eaten all the candies. The ghosts end the game when this happens.

We model this scenario as an MBSD problem $\mathcal{Q} = (\mathcal{G}, \mathcal{P}, \Phi, A)$, with domains $\mathcal{P}$(ac-Man, agent B) and $\mathcal{G}$(hosts, agent A). In $\mathcal{P}$, states model Pac-Man's and candies's position, while transitions model Pac-Man's move actions. Pac-Man cannot walk through walls. A candy disappears when Pac-Man moves on it. Similarly, states of $\mathcal{G}$ model (all) ghosts' position, and transitions model ghosts' movements through cells. Each transition corresponds to a move of all ghosts at once. $\mathcal{G}$ does not model candies or walls, as they do not affect nor are affected by ghosts.

Assuming an $N \times N$ grid with some cells occupied by walls, domain $\mathcal{P} = (S, s_0, \delta^p, \lambda^p)$ is as follows, where C is the set of cells (x, y) not containing a wall:

- for every $(x, y) \in C$, introduce the Boolean propositions $p_{x,y}$ (Pac-Man at (x, y)) and $c_{x,y}$ (candy at (x, y)), and let $Prop^p$ be the set of all such propositions;
- $\mathcal{S} \subseteq 2^{(Prop^p)}$ is the set of all interpretations over $Prop^p$ (represented as subsets of $Prop^p$), such that:
  - every $s \in S$ contains exactly one proposition $p_{x,y}$ (Pac-Man occupies exactly one cell);
  - for every $s \in S$, if $p_{x,y} \in s$ then $c_{x,y} \notin s$ (if Pac-Man is in (x, y) the cell contains no candy);
- let $s_0 = \{p_{0,0}\} \cup \{c_{x,y} \mid (x, y) \in C \setminus (0, 0)\}$ (Pac-Man in $(0, 0)$; cells without Pac-Man or walls contain a candy);
- δ^p is such that $(s, s') \in \delta^p$ iff, for all $(x, y) \in C$:
  - if $p_{x,y} \in s$ then $p_{x',y'} \in s'$, with $(x, y) \in \{(x, y), (x, y+1), (x, y-1), (x+1, y), (x-1, y))\}$ (Pac-Man moves at most by one cell, either horizontally or diagonally);
  - if $c_{x,y} \in s$ and $p_{x,y} \notin s'$ then $c_{x,y} \in s'$ (all candies available in s remain so if not eaten by Pac-Man).
- $\lambda^p(s) = s$.

Domain $\mathcal{G} = (T, t_0, \delta^g, \lambda^g)$ is defined in a similar way (we omit the formal details): we use propositions $bk_{x,y}, pk_{x,y}, ik_{x,y}, cd_{x,y}$ for Blinky, Pinky, Inky and Clyde's position, respectively; T is the set of interpretations where each ghost occupies exactly one cell (possibly containing a wall; many ghosts may be in the same cell); the ghosts start at $(N/2, N/2)$ (t_0); δ^g models a 1-cell horizontal or diagonal move for all ghosts at once; λ^g is the identity.

Pac-Man's primary goal (besides eating all candies) is to stay alive, which we formalize with the following point-wise mapping:

$$\Phi = \bigwedge_{(x,y)\in C} \Box((bk_{x,y} \vee pk_{x,y} \vee ik_{x,y} \vee cl_{x,y}) \rightarrow \neg p_{x,y}).$$

Any strategy σ that is a solution to $\mathcal{Q} = (\mathcal{G}, \mathcal{P}, \Phi, B)$ keeps Pac-Man alive. To enforce Φ, Pac-Man needs a strategy that prevents ending up in a cell where a ghost is. Notice that, to compute σ, one cannot proceed greedily by considering only one step at a time, but must plan over all future evolutions, to guarantee that Pac-Man does not eventually get trapped. With such σ, no matter when the ghosts end the game, Pac-Man will never lose (and, in fact, it will win, if the ghosts stop when all candies on the maze have been eaten).

4.2 Solving MBSD with Point-wise Mapping Specifications

We show how to solve an MBSD instance $\mathcal{P}$ by reduction to the problem of finding a winning strategy in a two-player game, for which algorithms are well known [Martin, 1975]. Specifically, we construct a two-player game $\mathcal{G}_\mathcal{P} = (\mathcal{A}, W)$ that has a winning strategy iff $\mathcal{P}$ has a solution.

Given an MBSD instance $\mathcal{P} = (\mathcal{D}_A, \mathcal{D}_B, \Phi, Ag_{stop})$, with $\mathcal{D}_A = (S, s_0, \delta^A, \lambda^A)$ and $\mathcal{D}_B = (T, t_0, \delta^B, \lambda^B)$, we construct the game arena $\mathcal{A} = (U, V, u_0, \alpha, \beta)$, where:

- $U = S \times T$;
- $V = S \times T$;
- $u_0 = (s_0, t_0)$;
- $\alpha = \{((s,t),(s',t)) \mid (s,s') \in \delta^A\}$;
- $\beta = \{((s,t),(s,t')) \mid (t,t') \in \delta^B\}$.

Intuitively, the nodes of $\mathcal{A}$ represent joint state configurations of both $\mathcal{D}_A$ and $\mathcal{D}_B$ (initially in their respective initial states), while the transition functions account for the moves A (modeled by $P1$) and B (modeled by $P2$) can perform, imposing, at the same time, their strict alternation.

As for the winning objective W, the key idea is that, since in point-wise mappings the temporal operator $\Box$ (*always*) distributes over conjunction, and since $Ag_{stop} = A$, the conjuncts of the mapping are in fact propositional formulae to be guaranteed all along the agent behaviors, captured by plays of $\mathcal{A}$. This can be easily expressed as a safety objective on $\mathcal{A}$, as shown below.

Let $\Phi = \bigwedge_{i=1}^{k} \Box(\varphi_i \rightarrow \psi_i)$ be the (point-wise) mapping specification. We have that $\Phi \equiv \Box\Phi'$, where $\Phi' \equiv \bigwedge_{i=1}^{k}(\varphi_i \rightarrow \psi_i)$ is a Boolean formula where every φ_i is over $Prop^A$ only and every ψ_i over $Prop^B$ only. Therefore, in order to solve $\mathcal{P}$, we need to find a strategy σ such that for every trace τ^A of $\mathcal{D}_A$, $\lambda(\tau^A, \tilde{\sigma}(\tau^A)) \models \Box\Phi'$, that is, $\lambda^A(s_j) \cup \lambda^B(t_j) \models \Phi'$ for $j = 0, \ldots, |\tau^A|$. Thus we can set $W = \mathrm{Safe}(g)$, with $g = \{(s,t) \in U \mid \lambda^A(s) \cup \lambda^B(t) \models \Phi'\}$.

As a consequence of the above construction, we obtain the following result.

Lemma 1. *There is a solution to $\mathcal{P}$ if and only if there is a solution to the safety game $\mathcal{G}_\mathcal{P}$.*

Finally, the construction of the safety game $\mathcal{G}_\mathcal{P}$ together with Lemma 1 gives us the following result.

Theorem 1. *Solving MBSD for point-wise mapping specifications is in PTIME for combined complexity, mapping complexity and domain complexity.*

Observe that if $\mathcal{D}_A$ and $\mathcal{D}_B$ are represented compactly (logarithmically) using, e.g., logical formulas or PDDL specifications [Haslum *et al.*, 2019], then the domain (and hence the combined) complexity becomes EXPTIME, and mapping complexity remains PTIME. Similar considerations hold also for the other cases that we analyze throughout the paper.

5 Mimicking Behaviors with Target Mapping Specifications

We now explore mimicking specifications that are of *target* nature. In this setting, B has to mimic A in such a way that whenever A reaches a certain target, so does B, although not necessarily at the same time step: B is free to reach the required target at the same time, later, or even before A does. For this to be possible, B must have the power to stop the game, which is what we assume here. Formally, target mapping specifications are formulas of the following form, where φ_i and ψ_i are Boolean properties over $\mathcal{D}_A$ and $\mathcal{D}_B$, respectively:

$$\varphi = \bigwedge_{i=1}^{k} (\Diamond\varphi_i) \rightarrow (\Diamond\psi_i)$$

As before, we first give an illustrative example that demonstrates the use of target mappings, then we explore algorithmic and complexity results.

5.1 Target Mapping Specifications in Rubik's Cube

Two agents, teacher H and learner L are provided with two Rubik's cubes of different sizes: H has edge of size 4 whereas L has one of size 3. L wants to learn from H the main steps to solve the cube; to this end, H shows L how to reach certain *milestone* configurations on the cube of size 4 and asks L to replicate them on the cube of size 3, even in a different order. Milestones are simply combinations of solved faces, e.g., *red and green*, *white and blue and yellow*, or simply *white*. Obviously, L cannot blindly replicate H's moves, as the cubes are of different sizes and the actual sequences to solve the faces are different; thus, L must find its way to reach the same milestones as H, possibly in a different order. When L is tired, it can stop the learning process.

We model this scenario as an MBSD problem instance $\mathcal{R} = (\mathcal{H}, \mathcal{L}, \Phi, B)$, where $\mathcal{H}$ and $\mathcal{L}$ model, respectively, H's

and L's dynamic domain, i.e., the two cubes. The two domains are conceptually analogous but, modeling cubes of different sizes, they feature different sets of states and transitions, which correspond to cube configurations and possible moves, respectively. We model such domains parametrically w.r.t. the size E of the edge.

Fix the cube in some position, name the faces as U(p), D(own), L(eft), R(ight), F(ront), B(ack), let $Fac = \{U, D, L, R, F, B\}$, and associate a pair of integer coordinates to each position in a face, so that every position is identified by a triple $(f, x, y) \in Pos = Fac \times \{0, \ldots, E-1\}^2$. To model the color assigned to tile (f, x, y), we use propositions of the form $c_{f,x,y}$, with $c \in Col = \{white, green, red, yellow, blue, orange\}$. Let $Prop$ be the set of all such propositions. Finally, index the horizontal and vertical "slices" of the cube from 0 to $E-1$.

The (parametric) dynamic domain for a Rubik's cube with edge of size E is the domain $\mathcal{D}(E) = (S, s_0, \delta, \lambda)$, where:

- $S \subseteq 2^{Prop^E}$ is the set of all admissible (i.e., reachable) cube's configurations; among other constraints, omitted for brevity, this requires that, for every $s \in S$:
  - for every $(f, x, y) \in Pos$, there exists exactly one $c \in C$ such that $c_{f,x,y} \in s$ (every position has exactly one color);
- s_0 is an arbitrary state from S;
- δ allows a transition from s to s' iff s' models a configuration reachable from s by a $90°$ (clockwise or counterclockwise) rotation of one of its $2 * E$ slices;
- $\lambda(s) = s$.

We then define $\mathcal{H} = \mathcal{D}(4)$ and $\mathcal{L} = \mathcal{D}(3)$. To distinguish the elements of $\mathcal{H}$ from those of $\mathcal{L}$, we use a primed version in the latter, e.g., Pos' for positions, $c'_{f,x,y}$ for propositions, and so on.

As said, L's goal is to replicate the milestones shown by H. For every face $f \in Fac$, we define formula $C_f = \bigwedge_{(f,x,y) \in Pos} c_{f,x,y}$ to express that the tiles of face f have all the same color c. For $\mathcal{L}$, we correspondingly have $C'_f = \bigwedge_{(f,x,y) \in Pos'} c'_{f,x,y}$.

We report below an example of target mappings:

$$(\Diamond blue_R) \rightarrow (\Diamond blue'_R)$$
$$(\Diamond(red_U \wedge white_L)) \rightarrow (\Diamond(red'_U \wedge white'_L))$$
$$(\Diamond(red_U \wedge \neg white_L)) \rightarrow (\Diamond(red'_U \wedge \neg white'_L)).$$

Observe that L has many ways to fulfill H's requests: for instance, by reaching a configuration where $blue'_R \wedge red'_U \wedge white'_L$ holds, it has fulfilled the first and the second request, even if the configuration was reached before H showed the milestones. Obviously, however, the last request cannot be fulfilled at the same time as the second one, as $white'_L$ clearly excludes $\neg white'_L$, thus an additional effort by L is required to satisfy the specification.

5.2 Solving MBSD with Target Mapping Specifications

For target mappings as well, we reduce MBSD to strategy synthesis for a two-player game. To this end, assume an MBSD instance $\mathcal{P} = (\mathcal{D}_A, \mathcal{D}_B, \Phi, B)$ with mapping specification $\Phi = \bigwedge_{i=1}^{k} (\Diamond \varphi_i) \rightarrow (\Diamond \psi_i)$. To solve $\mathcal{P}$, we must find a strategy σ such that for every infinite trace $\tau^A_\infty = s_0 s_1 \cdots$ of $\mathcal{D}_A$ and every conjunct $(\Diamond \varphi_i) \rightarrow (\Diamond \psi_i)$ of Φ, if there exists an index j_i such that $\lambda^A(s_{j_i}) \models \varphi_i$, then there exist a finite prefix $\tau_A = s_0 \cdots s_n$ of τ^A_∞ and an index l_i such that, for $\sigma(\tau) = t_0 \cdots t_n$, we have that $\lambda^B(t_{l_i}) \models \psi_i$ (recall φ_i and ψ_i are Boolean formulae over $Prop^A$ only and $Prop^B$ only, respectively). As per Definition 3, this is equivalent to requiring that $\lambda(\tau^A, \tilde{\sigma}(\tau^A)) \models (\Diamond \varphi_i) \rightarrow (\Diamond \psi_i)$.

The challenge in constructing σ is that the index l_i may be equal, smaller or larger then j_i. Thus σ needs to record which φ_i or ψ_i were already met during the trace, up to the current point. Since the number of possible traces to the current state may be exponential, keeping count of all possible options may be expensive. We first discuss general domain structure, then in Section 5.2 we explore a very specific tree-like structure.

For general domains, there may exist many traces ending in a given state, and each such trace contains states that satisfy, in general, different sub-formulas φ_i and ψ_i occurring in the mappings. Thus satisfaction of sub-formulas cannot be associated to states as done before, but must be associated to traces. In fact, to check whether a target mapping is satisfied, it is enough to remember, for every $i = 1, \ldots, k$, whether A has satisfied φ_i and/or B has satisfied ψ_i, along a trace. This observation suggests to introduce a form of memory to record satisfaction of sub-formulas along traces. We do so by augmenting the game arena constructed in Section 4. In particular, we extend each node in the arena with an array of bits of size $2k$ to keep track of which sub-formulas φ_i and ψ_i were satisfied, along the play that led to the node, by some of the domain states contained in the nodes of the play.

Formally, let $M = (\{0,1\}^2)^k$ and let $[cd] = ((c_1, d_1), \ldots, (c_k, d_k))$ denote the generic element of M. Given an MBSD instance $\mathcal{P} = (\mathcal{D}_A, \mathcal{D}_B, \Phi, B)$, where $\mathcal{D}_A = (S, s_0, \delta^A, \lambda^A)$ and $\mathcal{D}_B = (T, t_0, \delta^B, \lambda^B)$, we define the game arena $\mathcal{A} = (U, V, u_0, \alpha, \beta)$ as follows:

- $U = S \times T \times M$;
- $V = S \times T \times M$;
- $u_0 = (s_0, t_0, [cd])$ such that, for every $i \leq k$, $c_i = 1$ iff $\lambda^A(s_0) \models \phi_i$ and $d_i = 1$ iff $\lambda^B(t_0) \models \psi_i$;
- $((s, t, [cd]), (s', t, [c'd])) \in \alpha$ iff $(s, s') \in \delta^A$, and for $i = 1, \ldots, k$, if $\lambda^A(s') \models \phi_i$ then $c'_i = 1$, otherwise $c'_i = c_i$;
- $((s, t, [cd]), (s, t', [cd'])) \in \beta$ iff $(t, t') \in \delta^B$, and for $i = 1, \ldots, k$, if $\lambda^B(t') \models \psi_i$ then $d'_i = 1$, otherwise $d'_i = d_i$.

We then define the game structure $\mathcal{G}_\mathcal{P} = (\mathcal{A}, W)$, where $W = \text{Reach}(g)$, with $g = \{u \in U \mid u = (s, t, [cd])$, where $[cd]$ is s.t. $c_i = 0$ or $d_i = 1$, for every $i = 1, \ldots, k\}$. Intuitively, g is the set of all nodes reached by a play such that if ϕ_i is satisfied in the play (by a state of $\mathcal{D}_A$ in some node of the play), then so is ψ_i, for $i = 1, \ldots, k$ (by a state of $\mathcal{D}_B$ in some node of the play). Thus, if a play contains a node from g then the corresponding traces of $\mathcal{D}_A$ and $\mathcal{D}_B$, combined, satisfy all the mapping's conjuncts.

As a consequence of this construction, we obtain the following result.

Lemma 2. *There is a solution to $\mathcal{P}$ if and only if there is a winning strategy for the reachability game $\mathcal{G}_{\mathcal{P}}$.*

Then, Lemma 2 gives us the following.

Theorem 2. *MBSD with target mapping specifications can be solved in time polynomial in $|\mathcal{D}_A \times \mathcal{D}_B| \times |\Phi| \times 4^k$, with Φ the mapping specification and k the number of its conjuncts.*

An immediate consequence of Theorem 2 is that, for mappings of fixed size, the domain-complexity of the problem is in PTIME. For combined complexity, note that the memory-keeping approach adopted in $\mathcal{G}_{\mathcal{P}}$ is of a monotonic nature, i.e., once set, the bits corresponding to the satisfaction of ψ_i and ϕ_i cannot be unset. We use this insight to tighten our result and show that the presented construction can be in fact carried out in PSPACE.

Theorem 3. *MBSD for target mapping specifications is in PSPACE for combined complexity and mapping complexity, and in PTIME for domain complexity.*

We continue our analysis of the case of MBSD target mapping specifications by exploring whether memory-keeping is avoidable and a more effective solution approach can be found. As the following result implies, this is, most likely, not the case.

Theorem 4. *MBSD for target mapping specifications is PSPACE-hard in combined complexity (even for $\mathcal{D}_A$, $\mathcal{D}_B$ as simple DAGs).*

MBSD for Tree-like Domains

We conclude this section by discussing a very specific tree-like domain structure. We say that a dynamic domain $\mathcal{D} = (S, s_0, \delta, \lambda)$ is *tree-like* if the transition relation δ induces a tree structure on the states, except for some states which may admit self-loops as their *only* outgoing transition (therefore such states would be leaves, if self-loops were not present). For this class of domains, the exponential blowup on the number of traces does not occur, as for every state s there exists only a unique trace ending in s (modulo a possible suffix due to self-loops).

Theorem 5. *Solving MBSD for target mapping specifications and tree-like $\mathcal{D}_A$ and $\mathcal{D}_B$ is in PTIME for combined complexity, domain complexity, and mapping complexity.*

As before, the combined and domain complexities are EXPTIME, for $\mathcal{D}_A$ and $\mathcal{D}_B$ described succinctly.

6 Solving MBSD with General Mapping Specifications

The final variant of mapping specifications that we study is of the most general form, where Φ can be any arbitrary LTL_f formula over $Prop^A \cup Prop^B$. For this, we exploit the fact that for every LTL_f formula Φ, there exists a DFA $\mathcal{F}_\Phi$ that accepts exactly the traces that satisfy Φ [De Giacomo and Vardi, 2013]. Depending on which agent stops, the problem specializes into one of the following:

- if A stops: find a strategy for B such that every trace always visits an accepting state of $\mathcal{F}_\Phi$;
- if B stops: find a strategy for B such that every trace eventually reaches an accepting state of $\mathcal{F}_\Phi$.

To solve this variant, we again reduce MBSD to a two-player game structure $\mathcal{G}_{\mathcal{P}} = (\mathcal{A}, W)$, as in our previous constructions, then solve a safety game, if A stops, and a reachability game, if B stops. To follow the mapping as the game proceeds, we incorporate $\mathcal{F}_\Phi$ into the arena. This requires a careful synchronization, as the propositional labels associated with the *states* of dynamic domains affect the *transitions* of the automaton.

Formally, given an MBSD instance $\mathcal{P} = (\mathcal{D}_A, \mathcal{D}_B, \Phi, Ag_{stop})$, where $\mathcal{D}_A = (S, s_0, \delta^A, \lambda^A)$ and $\mathcal{D}_B = (T, t_0, \delta^A, \lambda^A)$, we construct the DFA $\mathcal{F}_\Phi = (\Sigma, Q, q_0, \eta, acc)$ as in [De Giacomo and Vardi, 2013], where $\Sigma = 2^{Prop^A \cup Prop^B}$ is the input alphabet.

Then, we define a two-player game arena $\mathcal{A} = (U, V, u_0, \alpha, \beta)$ as follows:

- $U = S \times T \times Q$;
- $V = S \times T \times Q$;
- $u_0 = (s_0, t_0, q_0')$, where $q_0' = \eta(q_0, \lambda(s_0) \cup \lambda(t_0))$;
- $\alpha = \{(s,t,q), (s',t,q) \mid (s,s') \in \delta^A\}$;
- $\beta = \{(s,t,q), (s,t',q') \mid (t,t') \in \delta^B$ and $\eta(q, \lambda(s) \cup \lambda(t')) = q'\}$.

Intuitively, $\mathcal{A}$ models the synchronous product of the arena defined in Section 4, with the DFA $\mathcal{F}_\Phi$. As such, the DFA first needs to make a transition from its own initial state q_0 to read the labelling information of both initial states s_0 and t_0 of $\mathcal{D}_A$ and $\mathcal{D}_B$, respectively. This is already accounted for by q_0', in the initial state u_0 of the arena. At every step, from current node $u = (s,t,q)$, $P1$ first chooses the next state s' of $\mathcal{D}_A$, then $P2$ chooses a state t' of $\mathcal{D}_B$, both according to their transition relation, and finally $\mathcal{F}_\Phi$ progresses, according to its transition function η and by reading the labeling of s' and t', from q to $q' = \eta(q, \lambda^A(s') \cup \lambda^B(t'))$.

For the winning objective W, define the set of goal nodes $g = \{u \in U \mid u = (s,t,q)$ such that $q \in acc\}$. That is, g consists of the nodes in the arena where $\mathcal{F}_\Phi$ is in an accepting state. Then, we define $W = \text{Safe}(g)$ (to play a safety game), if $Ag_{stop} = A$, and $W = \text{Reach}(g)$ (to play a reachability game), if $Ag_{stop} = B$.

The following theorem states the correctness of the construction.

Theorem 6. *There is a solution to $\mathcal{P}$ if and only if there a solution to $\mathcal{G}_{\mathcal{P}}$.*

Clearly, the constructed winning strategy σ from the reduced game $\mathcal{G}_{\mathcal{P}}$ is a solution to $\mathcal{P}$.

Finally, we obtain the following complexity result for the problem in its most general form.

Theorem 7. *Solving MBSD for general mapping specifications can be done in 2EXPTIME in combined complexity and mapping complexity, and in PTIME in domain complexity.*

7 Related Work

Linear Temporal Logic on finite traces (LTL_f) [De Giacomo and Vardi, 2013] has been widely adopted in different areas of CS and AI, as a convenient way to specify finite-trace properties, due to the way it finely balances expressive power and reasoning complexity. It has been used, e.g., in Machine Learning to encode a-priori knowledge [Camacho *et al.*, 2019; De Giacomo *et al.*, 2019; Xie *et al.*, 2021]; in strategy synthesis to specify desired agent tasks [De Giacomo and Vardi, 2015; Zhu *et al.*, 2017; Camacho *et al.*, 2018]; in Business Process Management (BPM) as a specification language for process execution monitoring [Pesic *et al.*, 2007; De Giacomo *et al.*, 2014; Di Ciccio *et al.*, 2017]. It has also found application as a natural way to capture non-Markovian rewards in Markov Decision Processes (MDPs) [Brafman *et al.*, 2018], MDPs policy synthesis [Wells *et al.*, 2020], and non-Markovian planning and decision problems [Brafman and De Giacomo, 2019]. Here we show yet another use of LTL_f. We use it to relate the behaviors in two separated domains through mapping specifications so as to control the mimicking between the two domains.

Mimicking has been recently studied in Formal Methods [Amram *et al.*, 2021]. In [Amram *et al.*, 2021], the notion of *mimicking* is specified in *separated GR(k) formulas*, a strict fragment of LTL. This makes the setting there not suitable for specifying mimicking behaviors of intelligent agents, since an intelligent agent will not keep acting indefinitely long, but only for a finite (but unbounded) number of steps. Moreover, the distinctions between the two systems and the mimicking specification were not singled out. This makes it difficult to provide a precise computational complexity analysis with respect to the systems, and the mimicking specification, separately.

A strictly related work, though more specific, is *Automatic Behavior Composition* [De Giacomo *et al.*, 2013], where a set of available behaviors must be orchestrated in order to mimic a desired, unavailable, target behavior. That work deals with a specific mapping specification over actions, corresponding to the formal notion of *simulation* [Milner, 1971]. This current work devises a more general framework and a solution approach for a wider spectrum of mapping specifications, in a finite-trace framework.

Finally, we want to notice that our framework is similar to what studied in data integration and data exchange [Lenzerini, 2002; Fagin *et al.*, 2005; Giacomo *et al.*, 2007; Kolaitis, 2018], where there are source databases, target databases, and mapping between them that relate the data in one with the data in the other. While similar concepts can certainly be found in our framework, here we do not consider data but dynamic behaviors, an aspect which makes the technical development very different.

8 Conclusion and Discussion

We have studied the problem of mimicking behaviors in separated domains, in a finite-trace setting where the notion of *mimicking* is captured by LTL_f mapping specifications. The problem consists in finding a strategy that allows an agent B to mimic the behavior of another agent A. We have devised an approach for the general formulation, based on a reduction to suitable two-player games, and have derived corresponding complexity results. We have also identified two specializations of the problem, based on the form of their mappings, which show simpler approaches and better computational properties. For these, we have also provided illustrative examples.

A question that naturally arises, for which we have no conclusive answer yet, is to what extent domain separation and possibly separated types of conditions can be exploited to obtain complexity improvements in general, not only on the problems analyzed here. In this respect, we take the following few points for discussion.

We first note that the framework in [Amram *et al.*, 2021] can be adapted to an infinite-trace variant of MBSD, with target mapping specifications of the form $\Phi = \bigwedge_{l=1}^{k}(\bigwedge_{i=1}^{n_l} \Box\Diamond(\varphi_{l,i}) \rightarrow \bigwedge_{j=1}^{m_l} \Box\Diamond(\psi_{l,j}))$. The results in [Amram *et al.*, 2021], which build heavily on domain separation, can be tailored to obtain a polynomial-time algorithm for (explicit) separated domains in combined complexity. In contrast, Theorem 4 in this paper shows that the finite variant is PSPACE-hard already for much simpler mappings. This gap seems to suggest that domain separation cannot prevent the book-keeping that is possibly mandatory for the finite case. Note however that Theorem 2 of this paper can be easily extended to specifications of the form $\Phi' = \bigwedge_{l=1}^{k}(\bigwedge_{i=1}^{n_l} \Diamond(\varphi_{l,i}) \rightarrow \bigwedge_{j=1}^{m_l} \Diamond(\psi_{l,j}))$, yielding an algorithm of time polynomial in the domain size but exponential in the number of Boolean subformulas in Φ'.

A second point of observation is the following. While the result in Section 6 provides an upper bound for mappings expressed as general LTL_f formulas, one can consider a more relaxed form $\Phi = \bigwedge_{i \leq k}(\phi_i \rightarrow \psi_i)$ where each ϕ_i (resp. ψ_i) is an LTL_f formulas over $Prop^A$ (resp. $Prop^B$) only. While still PSPACE-hard (see Theorem 4), it is tempting to use some form of memory keeping as done in Theorem 2 to avoid the 2EXPTIME complexity. The challenge, however, is that every attempt to monitor satisfaction for even a single LTL_f subformula, whether ϕ_i or ψ_i, seems to require an LTL_f to DFA construction that already yields the 2EXPTIME cost. Another approach could be to construct a DFA separately for each LTL_f sub-formula, then combine them along with the product of the domains and continue as in Section 6. This however involves a game with a state space to explore that is the (non-minimized) product of the respective DFAs, and is typically much larger than the (minimized) DFA constructed directly from Φ (as observed in [Tabajara and Vardi, 2019; Zhu *et al.*, 2021]). Moreover, in practice, state-of-the-art tools for translating LTL_f to DFAs [Bansal *et al.*, 2020; De Giacomo and Favorito, 2021] tend to take maximal advantage of automata minimization. How to avoid the DFA construction in such separated mappings to gain computational complexity advantage is yet to be explored.

Acknowledgments

This work is partially supported by the ERC Advanced Grant WhiteMech (No. 834228), the EU ICT-48 2020 project

TAILOR (No. 952215), the PRIN project RIPER (No. 20203FFYLK), the JPMorgan AI Faculty Research Award "Resilience-based Generalized Planning and Strategic Reasoning". We thank Lucas M. Tabajara and Gera Weiss for insightful discussions.

References

[Amram *et al.*, 2021] Gal Amram, Suguman Bansal, Dror Fried, Lucas Martinelli Tabajara, Moshe Y. Vardi, and Gera Weiss. Adapting behaviors via reactive synthesis. In *CAV*, pages 870–893, 2021.

[Bansal *et al.*, 2020] Suguman Bansal, Yong Li, Lucas M. Tabajara, and Moshe Y. Vardi. Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In *AAAI*, pages 9766–9774, 2020.

[Brafman and De Giacomo, 2019] Ronen I. Brafman and Giuseppe De Giacomo. Planning for LTL_f /LDL_f goals in non-markovian fully observable nondeterministic domains. In *IJCAI*, pages 1602–1608, 2019.

[Brafman *et al.*, 2018] Ronen I. Brafman, Giuseppe De Giacomo, and Fabio Patrizi. LTL_f/LDL_f non-markovian rewards. In *AAAI*, pages 1771–1778, 2018.

[Camacho *et al.*, 2018] Alberto Camacho, Jorge A. Baier, Christian J. Muise, and Sheila A. McIlraith. Finite LTL Synthesis as Planning. In *ICAPS*, pages 29–38, 2018.

[Camacho *et al.*, 2019] Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *IJCAI*, pages 6065–6073, 2019.

[De Giacomo and Favorito, 2021] Giuseppe De Giacomo and Marco Favorito. Compositional approach to translate LTL_f/LDL_f into deterministic finite automata. In *ICAPS*, pages 122–130, 2021.

[De Giacomo and Rubin, 2018] Giuseppe De Giacomo and Sasha Rubin. Automata-theoretic foundations of FOND planning for LTL_f and LDL_f goals. In *IJCAI*, pages 4729–4735, 2018.

[De Giacomo and Vardi, 2013] Giuseppe De Giacomo and Moshe Y. Vardi. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*, pages 854–860, 2013.

[De Giacomo and Vardi, 2015] Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In *IJCAI*, pages 1558–1564, 2015.

[De Giacomo *et al.*, 2013] Giuseppe De Giacomo, Fabio Patrizi, and Sebastian Sardiña. Automatic behavior composition synthesis. *Artif. Intell.*, 196:106–142, 2013.

[De Giacomo *et al.*, 2014] Giuseppe De Giacomo, Riccardo De Masellis, Marco Grasso, Fabrizio Maria Maggi, and Marco Montali. Monitoring business metaconstraints based on LTL and LDL for finite traces. In *BPM*, pages 1–17, 2014.

[De Giacomo *et al.*, 2019] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Foundations for restraining bolts: Reinforcement learning with LTL_f/LDL_f restraining specifications. In *ICAPS*, pages 128–136, 2019.

[Di Ciccio *et al.*, 2017] Claudio Di Ciccio, Fabrizio Maria Maggi, Marco Montali, and Jan Mendling. Resolving inconsistencies and redundancies in declarative process models. *Inf. Syst.*, 64:425–446, 2017.

[Fagin *et al.*, 2005] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[Giacomo *et al.*, 2007] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. On reconciling data exchange, data integration, and peer data management. In *PODS*, pages 133–142, 2007.

[Haslum *et al.*, 2019] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning. 2019.

[Kolaitis, 2018] Phokion G. Kolaitis. Reflections on schema mappings, data exchange, and metadata management. In *PODS*, pages 107–109, 2018.

[Lenzerini, 2002] Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

[Martin, 1975] D.A. Martin. Borel Determinacy. *Annals of Mathematics*, 65:363–371, 1975.

[Milner, 1971] Robin Milner. An algebraic definition of simulation between programs. In *IJCAI*, pages 481–489, 1971.

[Mitsunaga *et al.*, 2008] Noriaki Mitsunaga, Christian Smith, Takayuki Kanda, Hiroshi Ishiguro, and Norihiro Hagita. Adapting robot behavior for human–robot interaction. *IEEE Transactions on Robotics*, 24(4):911–916, 2008.

[Pesic *et al.*, 2007] Maja Pesic, Helen Schonenberg, and Wil M. P. van der Aalst. DECLARE: full support for loosely-structured processes. In *EDOC*, pages 287–300, 2007.

[Pnueli, 1977] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[Tabajara and Vardi, 2019] Lucas Martinelli Tabajara and Moshe Y. Vardi. Partitioning techniques in LTLf synthesis. In *IJCAI*, pages 5599–5606, 2019.

[Wells *et al.*, 2020] Andrew M. Wells, Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi. LTL_f synthesis on probabilistic systems. In *GandALF*, volume 326 of *EPTCS*, pages 166–181, 2020.

[Xie *et al.*, 2021] Yaqi Xie, Fan Zhou, and Harold Soh. Embedding symbolic temporal knowledge into deep sequential models. pages 4267–4273, 2021.

[Yarmohammadi *et al.*, 2013] Mahsa Yarmohammadi, Vivek Kumar Rangarajan Sridhar, Srinivas Bangalore, and Baskaran Sankaran. Incremental segmentation and decoding strategies for simultaneous translation. In *IJCNLP*, pages 1032–1036, 2013.

[Zheng *et al.*, 2020] Baigong Zheng, Kaibo Liu, Renjie Zheng, Mingbo Ma, Hairong Liu, and Liang Huang. Simultaneous translation policies: From fixed to adaptive. In *ACL*, pages 2847–2853, 2020.

[Zhu *et al.*, 2017] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. Symbolic LTL_f synthesis. In *IJCAI*, pages 1362–1369, 2017.

[Zhu *et al.*, 2021] Shufang Zhu, Lucas M. Tabajara, Geguang Pu, and Moshe Y. Vardi. On the power of automata minimization in temporal synthesis. In *GandALF*, volume 346 of *EPTCS*, pages 117–134, 2021.