Journal of Artificial Intelligence Research 77 (2023) 1087-1112 Submitted 01/2023; published 07/2023

Mimicking Behaviors in Separated Domains

Giuseppe De Giacomo GIUSEPPE.DEGIACOMO@CS.OX.AC.UK
University of Oxford

Dror Fried DFRIED@OPENU.AC.IL
The Open University of Israel

Fabio Patrizi PATRIZI@DIAG.UNIROMA1.IT
Sapienza University of Rome

Shufang Zhu✉ SHUFANG.ZHU@CS.OX.AC.UK
University of Oxford

Abstract

Devising a strategy to make a system mimic 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), that includes a robot that has to real-time adapt a human behavior (Mitsunaga, Smith, Kanda, Ishiguro, & Hagita, 2008), or a simultaneous interpretation of a speaker (Yarmohammadi, Sridhar, Bangalore, & Sankaran, 2013; Zheng, Liu, Zheng, Ma, Liu, & Huang, 2020). As an example, consider the following.

In an assembly line, a human operator (system A) paints car components and a set of robots (system B) simultaneously replicate the actions on replicas of the same component, so that many components are painted at the same time in parallel. The actions available to each robot are different from those available to the operator. For instance, the operator can arbitrarily control the set of arms and paint sprays, while robots have only certain pre-defined routines available. While robot actions allow for reaching all the configurations (position and orientation) relevant to the painting task, some intermediate ones are not reachable, so obtaining a desired final configuration may require different maneuvers. As a result, not only the action space but also the state space of the operator is different (in fact, larger) from that of robots. Moreover, the robots are not uniform; e.g., some may have been replaced with newer/different models or some may be under maintenance and have certain features disabled so that they have different actions and state space even among themselves.

The task of each robot is to simultaneously replicate the painting steps performed by the operator on its components. Such steps consist of accommodating the component in the robot's painting area, in the same orientation as that of the operator's, then spraying the same color over it. However, since the state spaces and the actions of the operator and the robots are different, the robots cannot simply copycat the operator's behavior in a step-by-step fashion. Rather, a *mapping* must be established between the operator's behaviors and the robots', so as to capture when the latter correctly *mimics* the former. For instance, one may require that whenever a component is in the operator's painting area oriented in some way, every robot has the same component in its respective area and is accommodated in the same way, and that whenever the operator's component is painted in some color, so are the other ones.

This example highlights the two main challenges arising in behavior mimicking. The first one is the need for a *formal specification* of the notion of mimicking which allows for specifying when the behaviors of A can be considered as correctly *mimicked* by those of B, although they can be substantially different from those of B. The second one is to find a *strategy* for B which, based on the actions performed by A, performs its own actions in such a way that the resulting behavior mimics that of A, according to the formal specification.

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 & 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 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 a local connection between the two separated domains. *Point-wise* mapping specifications have the form $\bigwedge_{i \leq k} \Box(\phi_i \rightarrow \psi_i)$ (the proper LTL_f definition can be found in Section 2.2) 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. For instance, the requirement mentioned in the robot example above can be captured, e.g., by the formula: $\bigwedge_{t \in Types} \Box(in\_area_t^A \rightarrow in\_area_t^B)$, where $\mathcal{D}_A$ (respectively $\mathcal{D}_B$) models the operator's (robot's) domain and $in\_area_t^A$ ($in\_area_t^B$) represents that a

component of type t is in the operator's (robot's) painting area. 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 to the ability to satisfy 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 objectives 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. In the robot example above, the formula $\bigwedge_{t \in Types}(\Diamond painted_t^A \rightarrow \Diamond painted_t^B)$ requires that if the operator paints a component of type t, the same must be done, no matter when, by the robot, as well. 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 mapping specification has the form of an arbitrary LTL_f formula Φ with properties over $\mathcal{D}_A$ and $\mathcal{D}_B$. This obviously increases the expressiveness of the specifications but at the expense of complexity. Considering again the painting-robot example, one such specification could be the conjunction of the following formulas: $\bigwedge_{t \in Types} \Diamond painted_t^A$, $\bigwedge_{t \in Types} \Box(painted_t^A \rightarrow \Diamond painted_t^B)$, $\bigwedge_{t \in Types} \neg(\neg painted_t^A \,\mathcal{U}\, painted_t^B)$, which respectively, require that: the operator paints at least one component of each type (first formula); whenever the operator paints a component of type t, the robot also does, at the same or at a later time (second formula); the robot never paints a component of type t (strictly) before the operator does (third formula).

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 Φ. To better characterize the source of complexity, we analyze solutions in terms of: *combined complexity*, where neither $\mathcal{D}_A$, $\mathcal{D}_B$ nor Φ are fixed; *mapping complexity*, where $\mathcal{D}_A$ and $\mathcal{D}_B$ are fixed but Φ varies; and *domain complexity*, where Φ is fixed but $\mathcal{D}_A$ and $\mathcal{D}_B$ vary. This is analogous to what is done in planning when distinguishing complexity in terms of the domain and the goal (De Giacomo & Rubin, 2018).

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 given by the LTL_f mapping specification. In fact, we vary the class of specifications considered and analyze the corresponding resulting solution. 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 2EXPTIME-complete 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 a solution for general mapping specifications in Section 6. Finally we discuss related work in Section 7, and conclude in Section 8.

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 $p \in \mathit{Prop}$ 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 in which every propositional variable is quantified (sometimes called *bounded*) by either the existential quantifier $\exists$ or the universal quantifier $\forall$. A QBF formula is in Prenex Normal Form (PNF) if all of its quantifiers occur in the prefix of the formula. Thus, a PNF formula contains a quantifier-free part, that is preceded by blocks of quantifiers and their bounded variables. A switch in the blocks between the existential and universal quantifiers is called *alternation* of the quantifiers. True Quantified Boolean Formulas (TQBF) is the language of all QBF formulas in PNF that evaluate to $\top$. TQBF is known to be PSPACE-complete, see (Arora & Barak, 2009) for details.

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 & 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 $p \in \mathit{Prop}$, $\bigcirc$ is the *next* temporal operator and $\mathcal{U}$ is the *until* temporal operator. We use common abbreviations for *eventually* $\Diamond\varphi \equiv \top \,\mathcal{U}\, \varphi$ 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. $\pi_i \in 2^{\mathit{Prop}}$, 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., $\pi = \pi_0 \cdots \pi_n \in (2^{\mathit{Prop}})^+$. Moreover, $\mathit{last}(\pi)$ 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 $i \in \{0, \ldots, \mathit{last}(\pi)\}$, written $\pi, i \models \varphi$, as follows:

- $\pi, i \models p$ iff $p \in \pi_i$ (for $p \in \mathit{Prop}$);
- $\pi, i \models \varphi_1 \wedge \varphi_2$ 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 $i < \mathit{last}(\pi)$ and $\pi, i+1 \models \varphi$;
- $\pi, i \models \Box\varphi$ iff $\forall j.i \leq j \leq \mathit{last}(\pi)$ and $\pi, j \models \varphi$;
- $\pi, i \models \Diamond\varphi$ iff $\exists j.i \leq j \leq \mathit{last}(\pi)$ and $\pi, j \models \varphi$;
- $\pi, i \models \varphi_1 \mathcal{U} \varphi_2$ iff $\exists j.i \leq j \leq \mathit{last}(\pi)$ and $\pi, j \models \varphi_2$, and $\forall k.i \leq k < j$ we have that $\pi, k \models \varphi_1$.

We say that $\pi \in (2^{\mathit{Prop}})^+$ *satisfies* an LTL$_f$ formula φ, written $\pi \models \varphi$, if $\pi, 0 \models \varphi$. Two LTL$_f$ formulas that are satisfied by exactly the same words are called *logically equivalent*. We denote the logical equivalence relation by $\equiv$.

A Deterministic Finite Automaton (DFA) $\mathcal{F}_\varphi = (2^{\mathit{Prop}}, Q, q_0, \eta, \mathit{acc})$ is a finite state machine in which 2^{Prop} is the alphabet of the DFA, Q is the finite set of states, $q_0 \in Q$ is the initial state, $\eta : Q \times 2^{\mathit{Prop}} \to Q$ is the transition function, and $\mathit{acc} \subseteq Q$ is a set of accepting states. 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 & Vardi, 2013).

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 game 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 and $U \cap V = \emptyset$;
- $u_0 \in U$ 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*. In this work all plays are finite.

Formally, a (finite) *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. Moreover, $\rho|_U = \rho_0\rho_2 \cdots \rho_n$ 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* $W \subseteq (U \cup V)^+$ is a (compact) representation of a set of plays, called *winning plays*. If, when the game stops, the play ρ is a winning play, i.e., $\rho \in W$, then $P2$ wins, 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. Given a game $\mathcal{G} = (\mathcal{A}, W)$, solving $\mathcal{G}$ concerns constructing a winning strategy σ for $P2$.

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

It is worth noting that both reachability and safety games enjoy memoryless determinacy, i.e., 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 $\mathcal{G}$ (Grädel, Thomas, & Wilke, 2002). The proof is constructive in the sense that it can be immediately turned into an algorithm that computes the memoryless winning strategies. More specifically, computing a memoryless winning strategy for a reachability game requires conducting a least-fixpoint computation on the game area, while computing one for a safety game, instead, requires a greatest-fixpoint computation. Therefore, both reachability and safety games can be solved in PTIME in the size of $\mathcal{G}$. For more details, we refer to (Grädel et al., 2002).

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. Specifically, only one agent, A or 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 labeled transition systems.

Definition 1 (Dynamic Domain)**.** *A* dynamic domain *over a finite set of propositions Prop is a labeled transition system in form of a tuple $\mathcal{D} = (S, s_0, \delta, \lambda)$, such that:*

- *S is the finite set of domain states;*
- *$s_0 \in S$ is the initial domain state;*
- *$\delta \subseteq S \times S$ is the transition relation;*
- *$\lambda : S \mapsto 2^{Prop}$ 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., $|\tau|$ is the number of states in τ if τ is of a finite length and $|\tau|$ is ∞ if τ is on an infinite length. 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 a finite set pf propositions $Prop^A$;*
- *$\mathcal{D}_B = (T, t_0, \delta^B, \lambda^B)$ is a dynamic domain over a finite set pf propositions $Prop^B$, with $Prop^A \cap Prop^B = \emptyset$ and $S \cap T = \emptyset$;*
- *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 total function $\sigma : S^+ \to T$ 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:

- $\sigma(s_0) = t_0$;
- 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 with equal position indices.

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:

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

Observe that when point-wise mappings are used, the only interesting case is when the designated stop agent is A. Indeed, if agent B is the stop agent, then satisfying the target mapping is equivalent to satisfying it in the first state only since, after that, agent B can simply stop the game. To avoid such trivial settings, we assume that, when dealing with point-wise mappings, the designated stop agent is always A.

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 loses the game once touching them (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:

- $S = C \times 2^C$, with $s = ((x, y), Candies)$ modeling the state where:
  - Pac-man is in cell (x, y);
  - cell (w, z) contains a candy iff $(w, z) \in Candies$;

  Since Pac-man eats the candy in the cell it occupies, we require that $(x, y) \notin Candies$, i.e., that the cell occupied by Pac-man does not contain a candy;
- $s_0 = ((0, 0), Candies)$, for $Candies = C \setminus \{(0, 0)\}$, i.e., Pac-man is in $(0, 0)$ and every cell without Pac-man or walls contains a candy;
- δ^p is such that $(((x, y), Candies), ((x', y'), Candies')) \in \delta^p$ iff:
  - $(x', y') \in \{(x, y), (x, y+1), (x, y-1), (x+1, y), (x-1, y)\}$, i.e., Pac-man can move by one cell at most, either horizontally or vertically;
  - $Candies' = Candies \setminus \{(x', y')\}$, i.e., all candies remain where they are except the one possibly eaten by Pac-man.
- $Prop^p$ contains, for every cell $(x, y) \in C$, one proposition $p_{x,y}$ (Pac-man is in cell (x, y)) and one proposition $c_{x,y}$ (cell (x, y) contains a candy);
- $\lambda^p(((x, y), Candies)) = \{p_{x,y}\} \cup \{c_{w,z} \mid (w, z) \in Candies\}$.

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 $(N/2, N/2)$ as indicated in 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 (Grädel et al., 2002). 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 is 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 $W = \text{Safe}(g)$, with $g = \{(s,t) \in U \mid \lambda^A(s) \cup \lambda^B(t) \models \Phi'\}$.

Define the two-player game defined above as $\mathcal{G}_\mathcal{P} = (\mathcal{A}, W)$. 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 $\mathcal{G}_{\mathcal{P}}$.*

Proof. As an intuition, notice that once computed, a winning strategy for $\mathcal{G}_{\mathcal{P}}$ is essentially a solution to $\mathcal{P}$. This, indeed, can be obtained by projecting away the V component of all the nodes in a play ρ, thus transforming ρ into a trace of $\mathcal{D}_A$.

We now show the proof in detail. We first show that if there is a solution to $\mathcal{P}$ then there is a solution to $\mathcal{G}_{\mathcal{P}}$. For that, we first show that if σ is an executable strategy for $\mathcal{P}$ then σ can be reduced to a strategy σ' for $G_{\mathcal{P}}$. To this end, consider a play $\rho = \rho_0\rho_1\cdots\rho_n$, with $\rho_i = (s_i, t_i)$ and a state (s_{n+1}, t_n) such that $((s_n, t_n), (s_{n+1}, t_n)) \in \alpha$. Let $\tau = s_0 s_1 s_3 \cdots s_{n-1} s_{n+1} \in V^{n/2+1}$. By the definition of $\mathcal{G}_{\mathcal{P}}$, τ is a trace of $\mathcal{D}_A$. Therefore, since σ is executable, σ is defined on τ. Thus, for $\rho' = \rho \circ (s_{n+1}, t_n)$, where $\circ$ denotes concatenation, we can define $\sigma'(\rho') = (s_{n+1}, \sigma(\tau))$. Note that this is a proper definition since the trace $\tilde{\sigma}(\tau)$ induced by σ on τ is a trace in $\mathcal{D}_B$, hence $(t_n, \sigma(\tau)) \in \delta^B$. Thus σ' is a proper strategy for $\mathcal{G}_{\mathcal{P}}$.

Next, we need the following claim that describes the correspondence between σ and σ'.

Claim 1. *A sequence $\rho = \rho_0 \cdots \rho_n \in (U \cup V)^+$ is a play of $\mathcal{G}_{\mathcal{P}}$ compatible with σ' iff there exist a trace $\tau^A = s_0 \cdots s_n$ of $\mathcal{D}_A$ and a trace τ^B of $\mathcal{D}_B$ such that $\tau^B = \tilde{\sigma}(\tau^A) = t_0 \cdots t_n$ and $\rho = (s_0, t_0)(s_1, t_0) \cdots (s_n, t_{n-1})(s_n, t_n)$.*

For a proof of Claim 1, given a trace $\tau^A = s_0 \cdots s_n$ of $\mathcal{D}_A$, let $\tau^B = \tilde{\sigma}(\tau^A) = t_0 \cdots t_n$ be the trace of $\mathcal{D}_B$ induced by σ on τ^A. By the definition of $\mathcal{G}_{\mathcal{P}}$ and that of σ' provided above, it follows that the sequence $\rho = (s_0, t_0)(s_1, t_0) \cdots (s_n, t_{n-1})(s_n, t_n)$ is a play of $\mathcal{G}_{\mathcal{P}}$ compatible with σ'. On the other hand, for a play $\rho = (s_0, t_0) \cdots (s_n, t_n)$ compatible with σ', again by the definition of $\mathcal{G}_{\mathcal{P}}$ and σ', we have that the sequences $\tau_A = s_0 \cdots s_n$ and $\tau_B = t_0 \cdots t_n$ are traces of, respectively $\mathcal{D}_A$ and $\mathcal{D}_B$, such that $\tau_B = \tilde{\sigma}(\tau_A)$.

Back to proving Lemma 1, since σ is a solution, every trace τ^A in $\mathcal{D}_A$ is such that $\lambda(\tau^A, \tilde{\sigma}(\tau^A)) \models \Phi$. For $\tau^A = s_0 \cdots s_n$, let $\tau^B = \tilde{\sigma}(\tau^A) = t_0 \cdots t_n$. Because $\Phi = \Box\Phi'$ is a point-wise mapping, for every i, we have that $(\lambda^A(s_i), \lambda^B(t_i)) \models \Phi'$, that is, in $\mathcal{G}_{\mathcal{P}}$, $(s_i, t_i) \in g$. Now, let $\rho = (s_0, t_0)(s_1, t_0) \cdots (s_n, t_{n-1})(s_n, t_n)$ be a play in $\mathcal{G}_{\mathcal{P}}$ compatible with σ' (recall $(s_n, t_n) \in U$). By Claim 1, the sequences $\tau^A = s_0 \cdots s_n$ and $\tau^B = t_0 \cdots t_n$ are traces of $\mathcal{D}_A$ and $\mathcal{D}_B$, respectively, with $\tau^B = \tilde{\sigma}(\tau^A)$. Then $(\lambda^A(s_n), \lambda^B(t_n)) \models \Phi'$, that is $(s_n, t_n) \in g$. Since ρ is arbitrary, every play in $G_{\mathcal{P}}$ compatible with σ' ends in a g node, hence σ' is a winning strategy for $P2$ in Safe(g). That completes the first direction of the theorem.

For the other direction, assume that σ' is a strategy for $\mathcal{G}_{\mathcal{P}}$. Define a strategy σ'' for $\mathcal{P}$ as follows. Define first $\sigma''(s_0) = t_0$. Then, for a play $\rho = \rho_0\rho_1 \cdots \rho_n$, with $\rho_i = (s_i, t_i)$, and a state (s_{n+1}, t_n) such that $((s_n, t_n), (s_{n+1}, t_n)) \in \alpha$, note that $\tau = s_0 s_1 s_3 \cdots s_{n-1} \cdots s_{n+1} \in V^{n/2+1}$ is a trace of $\mathcal{D}_A$, and define $\sigma''(\tau) = \sigma'(\rho \circ (s_{n+1}, t_n))$. By the definition of $\mathcal{G}_{\mathcal{P}}$, it follows that $\tau' = t_0 t_2 \cdots t_n \sigma''(\tau)$ is a trace in $\mathcal{D}_B$, thus σ'' is an executable strategy in $\mathcal{P}$.

To describe the correspondence between σ' and σ'' we make the next claim, completely analogous to Claim 1.

Claim 2. *A sequence $\rho = \rho_0 \cdots \rho_n \in (U \cup V)^+$ is a play of $\mathcal{G}_{\mathcal{P}}$ compatible with σ' iff there exist a trace $\tau^A = s_0 \cdots s_n$ of $\mathcal{D}_A$ and a trace τ^B of $\mathcal{D}_B$ such that $\tau^B = \tilde{\sigma}''(\tau^A) = t_0 \cdots t_n$ and $\rho = (s_0, t_0)(s_1, t_0) \cdots (s_n, t_{n-1})(s_n, t_n)$.*

For a proof, given a trace $\tau^A = s_0 \cdots s_n$ of $\mathcal{D}_A$, let $\tau^B = \tilde{\sigma''}(\tau^A) = t_0 \cdots t_n$ be the trace of $\mathcal{D}_B$ induced by σ'' on τ^A. By the definition of σ'' provided above, it follows that the sequence $\rho = (s_0, t_0)(s_1, t_0) \cdots (s_n, t_{n-1})(s_n, t_n)$ is a play of $\mathcal{G}_\mathcal{P}$ compatible with σ'. On the other hand, for a play $\rho = (s_0, t_0) \cdots (s_n, t_n)$ compatible with σ', again by the definition of σ'', we have that the sequences $\tau_A = s_0 \cdots s_n$ and $\tau_B = t_0 \cdots t_n$ are traces of, respectively $\mathcal{D}_A$ and $\mathcal{D}_B$, such that $\tau_B = \tilde{\sigma}''(\tau_A)$.

Now to conclude Lemma 1, assume that σ' is a winning strategy for $P2$ in $\mathcal{G}_\mathcal{P}$, with winning objective $W = \text{Safe}(g)$. For a trace $\tau^A = s_0 \cdots s_n$ of $\mathcal{D}_B$, let $\tau^B = \tilde{\sigma''}(\tau^A) = t_0 \cdots t_n$. By Claim 2, we have that the sequence $\rho = (s_0, t_0)(s_1, t_0) \cdots (s_n, t_{n-1})(s_n, t_n)$ is a play of $\mathcal{G}_\mathcal{P}$ compatible with σ'. Moreover, since σ' is winning, for $i = 1, \ldots, 2n$, $\rho_i \in g$. But then, for all pairs (s, t) in ρ, we have that $(\lambda^A(s), \lambda^B(t)) \models \phi'$, that is $\lambda(\tau^A, \tau^B)$ satisfies $\Box\Phi$. Since τ^A is arbitrary, it follows that σ'' is a solution for $\mathcal{P}$, which completes the proof. □

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.*

Proof. Given an MBSD instance $\mathcal{P}$, we construct the safety game $\mathcal{G}_\mathcal{P}$ as shown. Observe that the construction of $\mathcal{G}_\mathcal{P}$ requires constructing the game arena $\mathcal{A}$, which can be done in time polynomial in $|\mathcal{D}_A| + |\mathcal{D}_B|$, and setting the set of states g, which takes at most time $\mathcal{O}(|\Phi'|)$ for each state in $\mathcal{A}$. Finally by Lemma 1 we have that $\mathcal{P}$ has a solution if and only if $\mathcal{G}_\mathcal{P}$ has a solution, where solving a safety game takes linear time in the size of $\mathcal{G}_\mathcal{P}$ (Martin, 1975). □

Observe that if $\mathcal{D}_A$ and $\mathcal{D}_B$ are represented compactly (logarithmically) using, e.g., logical formulas or any other convenient formalism for domain specification such as the Planning Domain Definition Language (PDDL) used in planning (Haslum, Lipovetzky, Magazzeni, & Muise, 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 consider in 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 mentioned before, target mapping specifications allow a constraint to be satisfied even if agent B reaches the target before agent A does. See Section 8 for a brief discussion about a more restricted type of specification.

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 a cube of size 4 whereas L has one cube 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., solve *red and green*, solve *white and blue and yellow*, or simply solve *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 edge size E (i.e., 3 or 4) of the relevant cube.

Each domain state corresponds to a color assignment to each of the $(E \times E)$ tiles in every face. To define such assignments, fix the cube in some position such that the faces U(p), D(own), L(eft), R(ight), F(ront), B(ack) can be naturally identified, and let $Faces = \{U, D, L, R, F, B\}$. Every tile can be easily identified by a triple $(f, x, y) \in Pos = Faces \times \{0, \ldots, E-1\}^2$, with f representing the face and (x, y) the position of the tile in it. Finally, define the set $Colors = \{white, green, red, yellow, blue, orange\}$ of possible tile colors. The (parametric) dynamic domain for a Rubik's cube with edge E is the domain $\mathcal{D}(E) = (S, s_0, \delta, \lambda)$ such that:

- $S \subseteq Colors^{Pos}$ is the set of possible (i.e., reachable from a solved cube) configurations, modeled as functions of type $s : Pos \to Colors$;
- $s_0 \in S$ is an arbitrary possible state;
- δ allows a transition from s to s' iff s' models a configuration reachable from s by a $\pm 90°$ rotation of one of its $3 \times E$ slices;
- for every tile $(f, x, y) \in Pos$ and color $c \in Colors$, $Prop$ contains one proposition of the form $c_{f,x,y}$, indicating that tile (f, x, y) is assigned color c;
- $\lambda(s) = \{c_{f,x,y} \mid s(f, x, y) = c\}$.

We then define $\mathcal{H} = \mathcal{D}(4)$ and $\mathcal{L} = \mathcal{D}(3)$. We use superscripts to distinguish the elements of $\mathcal{H}$ from those of $\mathcal{L}$, i.e., $Pos^{\mathcal{H}}$ for positions, $c^{\mathcal{H}}_{f,x,y}$ for propositions, and so on.

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

We report below an example of target mappings:

$$(\Diamond blue_R^{\mathcal{H}}) \rightarrow (\Diamond blue_R^{\mathcal{L}})$$
$$(\Diamond(red_U^{\mathcal{H}} \wedge white_L^{\mathcal{H}})) \rightarrow (\Diamond(red_U^{\mathcal{L}} \wedge white_L^{\mathcal{L}}))$$
$$(\Diamond(red_U^{\mathcal{H}} \wedge \neg white_L^{\mathcal{H}})) \rightarrow (\Diamond(red_U^{\mathcal{L}} \wedge \neg white_L^{\mathcal{L}})).$$

Observe that L has many ways to fulfill H's requests: for instance, by reaching a configuration where $blue_R^{\mathcal{L}} \wedge red_U^{\mathcal{L}} \wedge white_L^{\mathcal{L}}$ 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 $white_L^{\mathcal{L}}$ clearly excludes $\neg white_L^{\mathcal{L}}$, 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 $\tau_\infty^A = s_0 s_1 \cdots$ 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 exists 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 $\lambda(\tau^A, \tilde{\sigma}(\tau^A)) \models (\Diamond\varphi_i) \rightarrow (\Diamond\psi_i)$ for every i.

The challenge in constructing σ is that the index l_i may be equal, smaller or larger than 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 can be expensive. We first discuss a solution for a general domain structure, then in Section 5.2.2 we explore a solution for 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. Specifically, to check whether a target mapping is satisfied, we need 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, for every $i = 1,\ldots,k$, 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 (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, the full proof of which is in line of Lemma 1.

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.*

Proof. Given an MBSD instance $\mathcal{P}$ with target mapping specifications, we construct a reachability game $\mathcal{G}_\mathcal{P}$ as shown above, which has size $|\mathcal{D}_A \times \mathcal{D}_B| \times 4^k$ and construction time polynomial in $|\mathcal{D}_A \times \mathcal{D}_B| \times |\Phi| \times 4^k$. The result then follows from Lemma 2 and from the fact that reachability games can be solved in linear time in the size of the game. □

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 in fact be 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.*

Proof. Having shown PTIME membership for domain-complexity in Theorem 2, it remains to show membership in PSPACE for combined-complexity. Assume that $P2$ wins the game $\mathcal{G}_\mathcal{P}$ and let $\sigma_\mathcal{P}$ be a memory-less winning strategy for $P2$. First see that every play ρ in $\sigma_\mathcal{P}$ is finite. Therefore, since $\sigma_\mathcal{P}$ is memory-less then every play ρ in $\sigma_\mathcal{P}$ does not hold two identical V nodes. That means that wlog in every play, the $[cd]$ index in every game node changes after at most $2 \times |\mathcal{D}_A \times \mathcal{D}_B|$ steps (since there are two copies in $\mathcal{G}_\mathcal{P}$ of the domain product for $P1$ and $P2$). Next, we use the monotonicty property in $\mathcal{G}_\mathcal{P}$. Specifically, between every two consecutive game nodes $\rho_i = (s,t,[cd])$, $\rho_{i+1} = (s',t',[c'd'])$ for some i in ρ, every index in $[cd]$ can only remain as is or change from 0 to 1, therefore the bit index changes at most $2k$ times throughout the play.

Thus, we reduce $\mathcal{G}_\mathcal{P}$ to an identical game $\mathcal{G}'_\mathcal{P}$ that terminates either when reaching an accepting state (then $P2$ wins), or after $2 \times |\mathcal{D}_A \times \mathcal{D}_B| \times 2k$ moves (then $P1$ wins). Standard

Min-Max algorithms (e.g. (Russell & Norvig, 2020)) that work in space size polynomial to maximal strategy depth can be deployed to verify a winning strategy for $P2$ in $\mathcal{G}'_{\mathcal{P}}$. Then on one hand if there is a winning strategy for $\mathcal{G}'_{\mathcal{P}}$ then there is a winning strategy for $\mathcal{G}_{\mathcal{P}}$ (the same strategy). On the other hand, if there is a winning strategy for $\mathcal{G}_{\mathcal{P}}$ then there is a memory-less winning strategy for $\mathcal{G}_{\mathcal{P}}$ that terminates after at most $2 \times |\mathcal{D}_A \times \mathcal{D}_B| \times 2k$ moves, which means that there is a winning strategy for $P2$ in $\mathcal{G}'_{\mathcal{P}}$. □

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 directed acyclic graphs).*

Proof Outline. We give a proof sketch, see Section 5.2.1 below for the detailed proof.

A *QBF-CNF-1* formula is a QBF formula in a CNF form in which every clause contains at most one universal variable. The language TQBF-CNF-1, of all true QBF-CNF-1 formulas, is also PSPACE-complete (see Proposition 1 below for completion). We show a polynomial time reduction from TQBF-CNF-1 to MBSD.

Given a QBF-CNF-1 formula F, assume wlog that each alternation of the quantifiers holds exactly a single variable. Construct the following MBSD instance $\mathcal{P}_F$. Intuitively, the domains $\mathcal{D}_A$ and $\mathcal{D}_B$ are directed acyclic graphs (DAG) where $\mathcal{D}_A$ controls the universal variables and $\mathcal{D}_B$ controls the existential variables, see Figure 1 for a rough sketch of the domains graph for a QBF formula with universal variables x_1^A, x_2^A and existential variables x_1^B, x_2^B. The initial states are s_1^A for agent A and s_1^B for agent B. By traversing the domains in alternation, each agent can choose at every junction node depicted as s_i^A for D_A or s_i^B for D_B, between either a *true-path* through $\top$ depicted nodes, or *false-path* through $\bot$ depicted nodes, thus corresponds to setting assignments to propositions that are analogue to universal (agent A) or existential (agent B) variables. For example, by visiting $s_{1_\top}^A$, agent A satisfies a proposition called $p_{1_\top}^A$ that corresponds to assign the universal variable $x_1^A = \top$. The mapping Φ is set according to F where each clause corresponds to a specific conjunct. For example a clause $(x_1^A \vee x_2^B)$ becomes a conjunct $\Diamond(p_{1_\bot}^A) \rightarrow \Diamond(p_{2_\top}^B)$ of Φ, where $p_{1_\bot}^A, p_{2_\top}^B$ are propositions in $Prop^A$ and $Prop^B$ respectively. An additional conjunct is added to ensure that agent B does not stop ahead of time. Then a strategy for agent B of which path to choose at every junction node corresponds to a strategy of which existential variable to assign for F. As such, F evaluates to $\top$ if and only if there is a solution to the MBSD $\mathcal{P}_F$. □

5.2.1 Detailed proof of Theorem 4

We first provide a detailed proof of Theorem 4. Then for completness we prove that the language TQBF-CNF-1, used in the proof, is PSPACE-complete.

Given a QBF-CNF-1 formula F with n universal variables $x_1^A, \cdots x_n^A$ and n existential variables $x_1^B, \cdots x_n^B$, assume wlog that each alternation of the quantifiers holds exactly a single variable. Construct the following MBSD instance $\mathcal{P}_F$. Intuitively for $H \in \{A, B\}$, the separate $\mathcal{D}_H$ domains are DAGs, each composed of $n+1$ *major states* $s_1^H, s_2^H, \cdots s_{n+1}^H$ respectively. Let $Prop^H = \{p_{1_\top}^H, p_{1_\bot}^H, \cdots p_{n_\top}^H, p_{n_\bot}^H, p_*^H\}$. From s_i^H, for $1 \leq i \leq n$, Agent H

can move only to s^H_{i+1} through exactly one of the following paths: a directed *true-path* that visits a vertex $s^H_{i_\top}$ labeled by $\{p^H_{i_\top}\}$ or a directed *false-path* that visits a vertex $s^H_{i_\bot}$ labeled by $\{p^H_{i_\bot}\}$. From s^H_{n+1} there is only directed self-loop. Thus the choice of which path to take means whether the subformula $\Diamond(p^H_{i_\top})$ is satisfied (that corresponds to setting $x_i = \top$), or $\Diamond(p^H_{i_\bot})$ is satisfied (that corresponds to setting $x_i = \bot$), but there is no path that forms a trace that satisfies both subformulas. Finally label s^A_{n+1} with $\{p^A_*\}$ and s^B_{n+1} with $\{p^B_*\}$. Then $\Diamond(p^H_*)$ is true in every game played.

For the mapping specification Φ, note that every clause C at F is of the form $(l_{x^A} \vee C_B)$ or (C_B), where l_{x^A} is a literal of a universal variable (*universal literal*) and C_B is a disjunction of literals of existential constraint (*existential literals*). For every such C_B define C_B^{Prop} to be a disjunction of propositions from $Prop^B$ in which every *negated* (resp. *un-negated*) literal $l_{x^B_i}$ is replaced with $p^B_{i_\bot}$ (resp. $p^B_{i_\top}$). Next, for every clause C of F, add to Φ a conjunct ν_C as follows. If C is of the form $(l_{x^A_i} \vee C_B)$, set $\nu_C = (\Diamond(p^A_{i_\bot}) \rightarrow \Diamond(C_B^{Prop}))$ if $l_{x^A_i}$ is un-negated, and $\nu_C = (\Diamond(p^A_{i_\top}) \rightarrow \Diamond(C_B^{Prop}))$ if $l_{x^A_i}$ is negated (note that the negation has switched for p^A). If C is of the form (C_B), set $\nu_C = (\Diamond(p^A_*) \rightarrow \Diamond(C_B^{Prop}))$. Since a clause $(x^A \vee C_B)$ is logically equivalent to $(\neg x^A \rightarrow C_B^{Prop})$ and the clause (C_B) is logically equivalent to $(\top \rightarrow C_B^{Prop})$, the construction of Φ mirrors a clause C with its corresponding conjunct ν_C. To complete Φ, add a final conjunct $(\Diamond(p^A_*) \rightarrow \Diamond(p^B_*))$ called the *stopping-constraint*. Note that the stopping-constraint is true only when both agents reach s^H_{n+1}. Thus, the role of the stopping-constraint is to ensure that agent B does not stop the game before reaching its end. Finally set $Ag_{stop} = B$ to finish the construction of $\mathcal{P}$ as an MBSD problem with target mapping spcification as required.

We give an example of the construction. Let F be the QBF input as follows.

$$\begin{aligned} F = \forall x^A_1 \exists x^B_1 \forall x^A_2 \exists x^B_2 ((x^A_1 \vee x^B_1 \vee x^B_2) \\ \wedge (\neg x^A_2 \vee \neg x^B_1) \\ \wedge (x^B_1 \vee \neg x^B_2)) \end{aligned}$$

Then the MBSD $\mathcal{P}_F$ is constructed as follows. The domains $\mathcal{D}_A, \mathcal{D}_B$ are in Figure 1 where s^A_1, s^B_1 are the initial state for agents A, B respectively. s^A_3 has a proposition $\{p^A_*\}$ and s^B_3 has a proposition $\{p^B_*\}$. For every $H \in \{A, B\}$ and $i \in \{1, 2\}$ every node $s^H_{i_\top}$ has a proposition $p^H_{i_\top}$ and every node $s^H_{i_\bot}$ has a proposition $p^H_{i_\bot}$. The stop agent Ag_{stop} is set to B. The mapping specification is as follows:

$$\begin{aligned} \Phi = &(\Diamond(p^A_{1_\bot}) \rightarrow \Diamond(p^B_{1_\top} \vee p^B_{2_\top})) \\ &\wedge (\Diamond(p^A_{2_\top}) \rightarrow \Diamond(p^B_{1_\bot})) \\ &\wedge (\Diamond(p^A_*) \rightarrow \Diamond(p^B_{1_\top} \vee p^B_{2_\bot})) \\ &\wedge (\Diamond(p^A_*) \rightarrow \Diamond(p^B_*)) \end{aligned}$$

Back to the proof, obviously the construction of $\mathcal{P}$ is time-polynomial wrt $|F|$. Note that while the agents move in $\mathcal{D}_A, \mathcal{D}_B$, the only choices that each agent has are at every s^H_i, to decide whether to move through the *true-path* or the *false-path*. Also note that both agents always progress at the same pace. That is: agent A is in s^A_i iff agent B is in s^B_i.

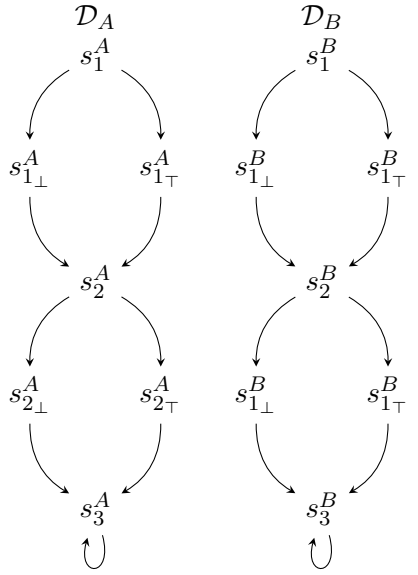

Figure 1: A rough sketch of the domains in the reduction construction in Theorem 4. The initial state for agent A is s_1^A and for agent B is s_1^B.

Finally, note that in every path that the agents take their respective domains, exactly one of $s_{i_\top}^H$ or $s_{i_\bot}^H$ can be visited, thus at every trace formed either $p_{i_\top}^H$ or $p_{i_\bot}^H$ are satisfied but not both. That means that $\Diamond(p_{i_\top}^H) \leftrightarrow \neg\Diamond(p_{i_\bot}^H)$ is always true.

Now assume that F evaluates to $\top$. Therefore there is a strategy σ_F for the existential player that sets F to be $\top$. Then we construct the following strategy $\sigma_\mathcal{P}$ for agent B: whenever agent A is at s_i^A and takes the *true-path* and thus satisfies $p_{i_\top}^A$ (resp. *false-path* to satisfy $p_{i_\bot}^A$), assign $x_i^A = \top$ (resp. $x_i^A = \bot$) in σ_F. If the result is $x_i^B = \top$ (resp. $x_i^B = \bot$) then set agent B to take the *true-path* and thus satisfy $p_{i_\top}^B$ (resp. *false-path to satisfy* $p_{i_\bot}^B$). Due to the mirroring between Φ and F, it follows that when both agents reach s_{n+1}^H (and therefore the stopping-constraint is true), we have that every clause C in F is true and thus so is its corresponding conjunct ν_C (recall that the subformula $\Diamond(p_*^A)$ is always true).

Next, assume that there is a winning strategy $\sigma_\mathcal{P}$ for $\mathcal{P}$. Then we similarly construct a strategy σ_F inductively as follows. At step 1, we are given an assignment for x_1^A and move agent A from s_1^A to the vertex corresponding to that assignment. That is, we move agent A through the *true-path* to $s_{1_\top}^A$, if $x_A^1 = \top$, and through the *false-path* to $s_{1_\bot}^A$, if $x_A^1 = \bot$. Then we assign the value to x_1^B that corresponds to the next move by agent B as dictated by $\sigma_\mathcal{P}$. That is, we assign $x_B^1 = \top$ if agent B takes the *true-path* and moves from s_1^B to $s_{1_\top}^B$, and assign $x_B^1 = \bot$ if agent B takes the *false-path* and moves from s_1^B to $s_{1_\bot}^B$. Suppose that we assigned values for the existential variables up to x_{i-1}^B. Then now both agents are posed at states s_i^A and s_i^B respectively. Then at step i, we again move agent A to the vertex corresponding to any assignment for x_i^A, and assign the value to x_i^B that corresponds to the next move by agent B as dictated by $\sigma_\mathcal{P}$. Following $\sigma_\mathcal{P}$ ensures that all the conjuncts of Φ

are true. Note that since the stopping-constraint is satisfied, agent B reaches s^B_{n+1}, which guarantees that σ_F is well defined for all variables. In addition, every clause C corresponding to a conjunct ν_C must also be true. For example, if $\Diamond(p^A_*) \rightarrow \Diamond(C^{Prop}_B)$ is true then, since $\Diamond(p^A_*)$ is always true, so must be $\Diamond(C^{Prop}_B)$, which means that a proposition in C^{Prop}_B is satisfied, thus a variable in C_B is set true in σ_F, hence C_B is true. That completes the proof. □

The PSPACE-hardness of TQBF-CNF-1 is not a hard exercise, for completion we bring a full proof.

Proposition 1. *TQBF-CNF-1 is PSPACE-complete.*

Proof. TQBF-CNF is known to be PSPACE-complete (Garey & Johnson, 1979). Obviously TQBF-CNF-1 is in PSPACE, we show PSPACE-hardness. Given a QBF-CNF formula F, we transform F to a QBF-CNF-1 formula F' such that F evaluates to $\top$ if and only if F' evaluates to $\top$. For that, we construct a formula F' from F as follows. We first add a fresh existential variable z_i for every universal variable x_i. In addition, conjunct F with clauses $(x_i \vee \neg z_i)$ and $(\neg x_i \vee z_i)$ that their conjunction is logically equivalent to $(x_i \leftrightarrow z_i)$. Finally, in every original clause C of F we replace every literal x_i with z_i and every literal $\neg x_i$ with $\neg z_i$. For the alternation order, we place the z_i anywhere after x_i (we can add dummy universal variables to keep the alternation interleaving order, as standard in such reductions). Since every original clause in F now contains only existential variables, we have that F' is indeed in the QBF-CNF-1 form that we described. Moreover, note that in F' every clause that holds a universal literal is of a size of 2.

Obviously, constructing F' from F is of polynomial time to $|F|$. Assume that F is true. Then there is a strategy σ_F for choosing existential variables such that F is true. Then define a strategy $\sigma_{F'}$ that copies σ_F, and for every choice for z_i, copy the assignment for x_i. That is set $z_i = \top$ iff x_i was set to $\top$. Since every x_i precedes z_i, this can be done. Then such a strategy sets F' to be true. Next assume F' is true. Then there is a strategy $\sigma_{F'}$ for choosing existential variables such that F is true. Then set a strategy for σ_F that just repeats $\sigma_{F'}$ while completely ignoring the assignment for z variables (this can be done since every assignment for z_i in $\sigma_{F'}$ has to be the same assignment that was set for x_i). Again, it follows that such a strategy sets F to be true. Thus, TQBF-CNF-1 is PSPACE-complete as well. □

5.2.2 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. *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.*

Proof. Given an MBSD instance $\mathcal{P}_{tree}$ with tree-like $\mathcal{D}_A$ and $\mathcal{D}_B$, consider the two-player game structure $\mathcal{G}_{\mathcal{P}_{tree}} = (\mathcal{A}, W)$ where the game arena $\mathcal{A} = (U, V, u_0, \alpha, \beta)$ is as described in Section 4.2 and $\Phi = \bigwedge_{i=1}^{k}(\Diamond\varphi_i) \rightarrow (\Diamond\psi_i)$ is the target mapping specification. It is immediate to see that, since $\mathcal{D}_A$ and $\mathcal{D}_B$ are tree-like, the arena $\mathcal{A}$ with the edges defined by α and β (which reflect those in $\mathcal{D}_A$ and $\mathcal{D}_B$), is tree-like as well.

Now, note that for every node $(s, t) \in U$ in the arena $\mathcal{A}$ and for each $i = 1, \ldots, k$, we can easily check whether the unique play ρ of $\mathcal{A}$ that ends in (s, t) either: (i) does not contain a node (s', t') for which $\lambda^A(s') \models \varphi_i$; or (ii) contains nodes (s', t') (s'', t'') (not necessarily disjoint) for which $\lambda^A(s'') \models \varphi_i$ and $\lambda^B(t'') \models \psi_i$. If that is the case, we call (s, t) an *i-accepting* node. Then, we define the *set of accepting states* as $g = \{u \in U \mid u \text{ is } i\text{-accepting, for every } i = 1, \ldots, k\}$, and the winning condition as $W = \text{Reach}(g)$. In this way, $\mathcal{G}_{\mathcal{P}_{tree}}$ is a reachability game, constructed in time polynomial in the size of $\mathcal{P}_{tree}$, and solvable in linear time in the size of $\mathcal{G}_{\mathcal{P}_{tree}}$. Result then follows since $\mathcal{P}_{tree}$ has a solution if and only if there is a solution to $\mathcal{G}_{\mathcal{P}_{tree}}$. □

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 & 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 & 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 is a solution to $\mathcal{G}_\mathcal{P}$.*

Proof. Let $Ag_{stop} = A$ (the case for $Ag_{stop} = B$ is similar), thus $\mathcal{G}_\mathcal{P} = (\mathcal{A}, \text{Safe}(g))$. By Definition 3, $\mathcal{P}$ has a solution σ iff for every trace τ^A of $\mathcal{D}_A$, we have that $\lambda(\tau^A, \tilde{\sigma}(\tau^A)) \models \Phi$. That is, $\lambda(\tau^A, \tilde{\sigma}(\tau^A))$ is accepted by $\mathcal{F}_\Phi$, i.e., the run on $\mathcal{F}_\Phi$ of $\lambda(\tau^A, \tilde{\sigma}(\tau^A))$ ends at an accepting state $q \in acc$. Due to the strict one-to-one correspondence between the transitions of $\mathcal{G}_\mathcal{P}$ with those of $\mathcal{D}_A$, $\mathcal{D}_B$ and $\mathcal{F}_\Phi$, we can simply transform σ to be such that $\sigma : V^+ \to U$. Hence, every play $\rho = \rho_0\rho_1 \cdots \rho_n$ of $\mathcal{A}$ compatible with σ is such that $\rho_k \in g$ for every even $k.0 \leq k \leq last(\rho)$. By definition of safety game, this holds iff σ is a winning strategy of $\mathcal{G}_\mathcal{P} = (\mathcal{A}, \text{Safe}(g))$. □

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

We conclude this section with the following theorems by which we obtain the complexity results 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.*

Proof. Constructing the DFA $\mathcal{F}_\Phi$ from the mapping specification Φ is in 2EXPTIME in the number of sub-formulas of Φ (De Giacomo & Vardi, 2013). Once $\mathcal{F}_\Phi$ is constructed, observe that the game arena $\mathcal{A}$ is the product of $\mathcal{D}_A$, $\mathcal{D}_B$ and the DFA $\mathcal{F}_\Phi$, which requires, to be constructed, polynomial time in the size of $|\mathcal{D}_A| + |\mathcal{D}_B| + |\mathcal{F}_\Phi|$. Moreover, both safety and reachability games can be solved in linear time in the size of $\mathcal{A}$, from which it follows that the MBSD problem for general mappings is in 2EXPTIME in combined complexity, PTIME in domain complexity, and 2EXPTIME in mapping complexity. □

Theorem 8. *Solving MBSD for general mapping specifications is 2EXPTIME-complete.*

Proof. Membership in 2EXPTIME was shown in Theorem 7, wrt combined complexity. The hardness can be obtained by an immediate reduction from LTL_f synthesis, known to be 2EXPTIME-complete (De Giacomo & Vardi, 2015). To see that, given an LTL_f formula Φ, construct an MBSD $\mathcal{P} = (\mathcal{D}_A, \mathcal{D}_B, \Phi, B)$ instance by setting trivial $\mathcal{D}_A$ and $\mathcal{D}_B$ to be such that the label of each state is simply $\top$. Then $\mathcal{P}$ has a solution if and only if Φ is realizable and a solution to $\mathcal{P}$ is a synthesized strategy for Φ. □

7. Related Work

Linear Temporal Logic on finite traces (LTL_f) (De Giacomo & Vardi, 2013) has been widely adopted in different areas of CS and AI for specifying desired tasks that are bound to terminate (Pesic, Schonenberg, & van der Aalst, 2007; De Giacomo & Vardi, 2015; Zhu, Tabajara, Li, Pu, & Vardi, 2017; Camacho, Baier, Muise, & McIlraith, 2018). Here, we adopt LTL_f for a similar purpose although, instead of specifying the tasks directly, we use LTL_f for specifying the desired properties of the mimicking between two domains.

Mimicking has been recently studied in Formal Methods (Amram, Bansal, Fried, Tabajara, Vardi, & Weiss, 2021). There, the notion of *mimicking* is specified in *separated GR(k) formulas*, a strict fragment of LTL. However, LTL (over infinite traces) does not appear to be a natural choice for specifying (mimicking) behavior properties, when these concern finitely many (though unbounded) steps. Obviously, it is well-known that LTL can encode LTL_f, but such encoding requires introducing book-keeping details that in turn lead to re-engineering of the domains to account for them in correctly phrasing the intended properties, thus leading to cumbersome and unintuitive specifications. This contrasts with LTL_f, which is specifically designed to deal with finite traces, and can thus be used *off-the-shelf* without additional efforts. Moreover, doing synthesis in LTL requires more sophisticated algorithms, a fact that, together with the need to suitably handle the details introduced by encoding LTL_f into LTL, may further complicate the identification of the sources of complexity. In this respect, it is worth observing that in (Amram et al., 2021) the problem is encoded as a single LTL formula, without distinguishing the two domains and the mimicking specification, thus no attempt is made to provide a fine computational complexity analysis with respect to the domains and the mimicking specification, considered separately, as we do in this work.

A strictly related work, though more specific, is *Automatic Behavior Composition* (De Giacomo, Patrizi, & Sardiña, 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). MBSD provides us with a more general framework, set for finite-traces, and a wider spectrum of mapping specifications. In addition, this work, as well as (Amram et al., 2021), focuses on *linear-time* specifications, as opposed to the *branching-time* underlying simulation in (De Giacomo et al., 2013).

Finally, we note that our framework is similar to studies in data integration and data exchange (Lenzerini, 2002; Fagin, Kolaitis, Miller, & Popa, 2005; De Giacomo, Lembo, Lenzerini, & Rosati, 2007; Kolaitis, 2018), where there are source databases, target databases, and mappings 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.

The target mapping specifications, extensively discussed in Section 5, consider a constraint to be satisfied even if agent B reaches a certain target before agent A does. This freedom seems natural to us since although B mimics A, B can still be much more clever than A, thus manages to achieve some targets ahead of A. As another example it may be that B's domain allows it to achieve the same targets as A does but in a different order than A. This specification however can be restricted, as to consider every constraint to be satisfied only if B reaches the relevant target *after* A does. This can be done by altering the target mapping specifications to be of the form:

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

While we choose not to elaborate, we believe that the results from Section 5 still hold for this type of specifications as well, with only minor adjustments required.

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 sub-formula, 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 & Vardi, 2019; Zhu, Tabajara, Pu, & Vardi, 2021)). Moreover, in practice, state-of-the-art tools for translating LTL_f to DFAs (Bansal, Li, Tabajara, & Vardi, 2020; De Giacomo & 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

We thank Lucas M. Tabajara, Maria Vaisberg Farberov, Moshe Y. Vardi, and Gera Weiss for their insightful comments. This work is supported in part by the ERC Advanced Grant WhiteMech (No. 834228), the PNRR MUR project PE0000013-FAIR and the Sapienza Project MARLeN.

References

Amram, G., Bansal, S., Fried, D., Tabajara, L. M., Vardi, M. Y., & Weiss, G. (2021). Adapting behaviors via reactive synthesis. In *CAV*, pp. 870–893.

Arora, S., & Barak, B. (2009). *Computational Complexity - A Modern Approach*. Cambridge University Press.

Bansal, S., Li, Y., Tabajara, L. M., & Vardi, M. Y. (2020). Hybrid compositional reasoning for reactive synthesis from finite-horizon specifications. In *AAAI*, pp. 9766–9774.

Camacho, A., Baier, J. A., Muise, C. J., & McIlraith, S. A. (2018). Finite LTL Synthesis as Planning. In *ICAPS*, pp. 29–38.

De Giacomo, G., & Favorito, M. (2021). Compositional approach to translate $\text{LTL}_f/\text{LDL}_f$ into deterministic finite automata. In *ICAPS*, pp. 122–130.

De Giacomo, G., Lembo, D., Lenzerini, M., & Rosati, R. (2007). On reconciling data exchange, data integration, and peer data management. In *PODS*, pp. 133–142. ACM.

De Giacomo, G., Patrizi, F., & Sardiña, S. (2013). Automatic behavior composition synthesis. *Artif. Intell.*, *196*, 106–142.

De Giacomo, G., & Rubin, S. (2018). Automata-theoretic foundations of FOND planning for LTL$_f$ and LDL$_f$ goals. In *IJCAI*, pp. 4729–4735.

De Giacomo, G., & Vardi, M. Y. (2013). Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*, pp. 854–860.

De Giacomo, G., & Vardi, M. Y. (2015). Synthesis for LTL and LDL on finite traces. In *IJCAI*, pp. 1558–1564.

Fagin, R., Kolaitis, P. G., Miller, R. J., & Popa, L. (2005). Data exchange: semantics and query answering. *Theor. Comput. Sci.*, *336*(1), 89–124.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Grädel, E., Thomas, W., & Wilke, T. (Eds.). (2002). *Automata, Logics, and Infinite Games: A Guide to Current Research*, Vol. 2500 of *Lecture Notes in Computer Science*. Springer.

Haslum, P., Lipovetzky, N., Magazzeni, D., & Muise, C. (2019). *An Introduction to the Planning Domain Definition Language*.

Kolaitis, P. G. (2018). Reflections on schema mappings, data exchange, and metadata management. In *PODS*, pp. 107–109. ACM.

Lenzerini, M. (2002). Data integration: A theoretical perspective. In *PODS*, pp. 233–246. ACM.

Martin, D. (1975). Borel Determinacy. *Annals of Mathematics*, *65*, 363–371.

Milner, R. (1971). An algebraic definition of simulation between programs. In *IJCAI*, pp. 481–489.

Mitsunaga, N., Smith, C., Kanda, T., Ishiguro, H., & Hagita, N. (2008). Adapting robot behavior for human–robot interaction. *IEEE Transactions on Robotics*, *24*(4), 911–916.

Pesic, M., Schonenberg, H., & van der Aalst, W. M. P. (2007). DECLARE: full support for loosely-structured processes. In *EDOC*, pp. 287–300.

Pnueli, A. (1977). The temporal logic of programs.. pp. 46–57.

Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson.

Tabajara, L. M., & Vardi, M. Y. (2019). Partitioning techniques in ltlf synthesis. In *IJCAI*, pp. 5599–5606.

Yarmohammadi, M., Sridhar, V. K. R., Bangalore, S., & Sankaran, B. (2013). Incremental segmentation and decoding strategies for simultaneous translation. In *IJCNLP*, pp. 1032–1036.

Zheng, B., Liu, K., Zheng, R., Ma, M., Liu, H., & Huang, L. (2020). Simultaneous translation policies: From fixed to adaptive. In *ACL*, pp. 2847–2853.

Zhu, S., Tabajara, L. M., Li, J., Pu, G., & Vardi, M. Y. (2017). Symbolic LTL$_f$ synthesis. In *IJCAI*, pp. 1362–1369.

Zhu, S., Tabajara, L. M., Pu, G., & Vardi, M. Y. (2021). On the power of automata minimization in temporal synthesis. In *GandALF*, Vol. 346 of *EPTCS*, pp. 117–134.