Synthesis with Mandatory Stop Actions

Giuseppe De Giacomo , Antonio Di Stasio , Giuseppe Perelli , Shufang Zhu*
Sapienza University of Rome, Rome, Italy
{degiacomo,distasio,perelli,zhu}@diag.uniroma1.it

Abstract

We study the impact of the need for the agent to obligatorily instruct the action stop in her strategies. More specifically we consider synthesis (i.e., planning) for LTL_f goals under LTL environment specifications in the case the agent must mandatorily stop at a certain point. We show that this obligation makes it impossible to exploit the liveness part of the LTL environment specifications to achieve her goal, effectively reducing the environment specifications to their safety part only. This has a deep impact on the efficiency of solving the synthesis, which can sidestep handling Büchi determinization associated to LTL synthesis, in favor of finite-state automata manipulation as in LTL_f synthesis. Next, we add to the agent goal, expressed in LTL_f, a safety goal, expressed in LTL. Safety goals must hold forever, even when the agent stops, since the environment can still continue its evolution. Hence the agent, before stopping, must ensure that her safety goal will be maintained even after she stops. To do synthesis in this case, we devise an effective approach that mixes a synthesis technique based on finite-state automata (as in the case of LTL_f goals) and model-checking of nondeterministic Büchi automata. In this way, again, we sidestep Büchi automata determinization, hence getting a synthesis technique that is intrinsically simpler than standard LTL synthesis.

1 Introduction

One of the most important questions for an intelligent agent is to reason and plan its actions to achieve its goal exploiting a model of its environment. Often such environment appears nondeterministic to the agent either because it is, or because the model, which is an approximation of the reality, is not detailed enough. This case is addressed in Planning by fully observable nondeterministic domains (FOND) planners (Cimatti et al. 2003; Sardiña and D'Ippolito 2015; Rodriguez et al. 2021), possibly adding fairness assumptions on the nondeterministic effects. More recently it has been advocated to extend the usual reachability goals used in Planning to goals expressed in LTL_f (or variants) (Camacho et al. 2017; De Giacomo and Rubin 2018; He et al. 2019). LTL_f is Linear-time Temporal Logic (the most popular logic to express temporal properties in Formal Verification (Pnueli 1977)), but interpreted on finite traces, see (De Giacomo and Vardi 2013).[1] This logic is particularly fitting for expressing temporally-extended goals in Planning since it retains the fact that ultimately the goal must be achieved and the plan terminated (Baier and McIlraith 2006).

While one can use LTL_f for expressing nondeterministic planning domains[2], LTL_f cannot be used to express fairness which is indeed a property of infinite traces. More generally, as explicitly observed in (Camacho, Bienvenu, and McIlraith 2018), the environment specifications should be expressed in LTL over infinite traces. Indeed the agent is supposed to finish her task and move on after a while, but the environment is not supposed to ever stop working. This has led to study synthesis (i.e. devise a strategy or plan) for LTL_f goals under LTL environment specifications (Camacho, Bienvenu, and McIlraith 2018; Aminof et al. 2018; Aminof et al. 2019; Camacho, Bienvenu, and McIlraith 2019).

It is interesting to observe that synthesis for LTL_f goals under LTL environment specifications can be reduced to standard LTL synthesis (Pnueli and Rosner 1989), as observed, e.g., in (Camacho, Bienvenu, and McIlraith 2018). Indeed this observation is just an obvious consideration, since LTL_f is a fragment of LTL as shown in (De Giacomo and Vardi 2013), and synthesis under LTL environment specifications can be reduced to standard synthesis of an implication of LTL formulas (although some subtleties should be considered (Aminof et al. 2019)). However, synthesis for LTL does not scale in general due to the determinization of nondeterministic Büchi automata, for which we do not have good algorithms, see e.g., (Finkbeiner 2016). Notably, synthesis in LTL_f (De Giacomo and Vardi 2015; Zhu et al. 2017) does not suffer from the same problem, because determinization is done through the standard subset construction which is easy to implement and often (af-

*Corresponding Author

[1] In this paper we will focus on LTL_f, but all results still hold if we use instead LDL_f (De Giacomo and Vardi 2013), which is a more expressive variant that captures monadic-second order logic on finite traces, instead of first-order logic on finite traces as LTL_f.

[2] Fully observable nondeterministic planning domains can be seen as safety properties in LTL: the environment forever reacts to actions as specified by the planning domain. Safety properties are properties on infinite traces, but if they can be broken at all, they can be broken with a finite prefix. This allows for capturing safety environment specification (but not safety goals) in LTL_f (De Giacomo et al. 2020a).

ter minimization) does not give rise to the typical exponential blowup of determinization (Tabakov and Vardi 2005; Tabakov, Rozier, and Vardi 2012).

For these reasons, several specific forms of LTL environment specifications have been considered, for example, allowing only for safety or co-safety properties (Camacho, Bienvenu, and McIlraith 2018), or only simple form of fairness and stability (Zhu et al. 2020), or using in the environment specifications both LTL_f (e.g., for representing nondeterministic planning domains or other safety properties) and LTL (e.g., for liveness/fairness), but separating the contributions of the two by limiting the second one as much as possible (De Giacomo et al. 2020a).

In this paper we consider synthesis for LTL_f goals under LTL environment specifications without any restriction on the form of the specifications themselves, but with a fundamental requirement on the kind of strategies/plans that the agent can adopt: the agent needs to obligatorily instruct the action stop in her strategies. The impact of this requirement is notable: the agent cannot wait for the environment to spontaneously bring about conditions that would allow to fulfil the goal. For example, if an agent in a shared kitchen wants to have the dish washed, and waiting guarantees that eventually somebody else will do the dishes, then waiting until somebody does the dishes would not be a good strategy for the agent. This is because when the dishes are done, and the agent can stop, would not be controlled by the agent itself. We show that adding this mandatory stop requirement has a deep impact: every LTL environment specification reduces to only its safety part (remember that all LTL formulas can be considered formed by a safety part and a liveness part (Alpern and Schneider 1987)). This allows us to devise a synthesis technique based on finite-state automata, which essentially reduces to LTL_f synthesis only.

Next, we add to the LTL_f goal, a safety goal expressed in LTL. While the LTL_f goal will be satisfied after a finite number of steps, the safety goal needs to remain true forever, even after the agent has stopped and cannot act to keep it true anymore. Indeed, when the agent stops, the liveness part of the environment specifications is still active and the environment can freely evolve, possibly breaking the agent safety goal. So the agent before stopping must put the system in a situation where whatever the environment does after, will not break the agent safety goal. To handle this case, we devise a synthesis technique that mixes a synthesis technique based on finite-state automata, as in the case of LTL_f goals, and model-checking of nondeterministic Büchi automata. In this way, again, we sidestep the difficulty of Büchi automata determinization, hence getting a synthesis technique that is intrinsically simpler than standard LTL synthesis.

Outline. The rest of the paper is organized as follows. In Section 2 we give some preliminary notions. In Section 3 we introduce our problem of synthesis with mandatory stop actions, giving a formal definition and illustrating it with examples. In Section 4 we present a correct and optimal synthesis technique for this case which is basically similar to the one used for LTL_f synthesis. In Section 5 we add safety goals, and in Section 6 we present a correct and optimal technique to handle this case, which although uses Büchi automata, avoids their determinization. We conclude the paper with a short discussion.

2 Preliminaries

LTL and LTL_f. LTL is one of the most popular logics for temporal properties (Pnueli 1977). Given a set of propositions $Prop$, the formulas of LTL 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 $p \in Prop$. We use common abbreviations for *eventually* $\Diamond\varphi \equiv true\,\mathcal{U}\,\varphi$ and *always* as $\Box\varphi \equiv \neg\Diamond\neg\varphi$.

LTL formulas are interpreted over infinite traces $\pi \in (2^{Prop})^\omega$. A *trace* $\pi = \pi_0\pi_1\ldots$ is a sequence of propositional interpretations (sets), where for every $i \geq 0$, $\pi_i \in 2^{Prop}$ is the i-th interpretation of π. Intuitively, π_i is interpreted as the set of propositions that are $true$ at instant i. Given π, we define when an LTL formula φ *holds* at position i, written as $\pi, i \models \varphi$, inductively on the structure of φ, 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 $\pi, i+1 \models \varphi$;
- $\pi, i \models \varphi_1 \,\mathcal{U}\, \varphi_2$ iff there exists $i \leq j$ such that $\pi, j \models \varphi_2$, and for all $k, i \leq k < j$ we have that $\pi, k \models \varphi_1$.

We say π *satisfies* φ, written as $\pi \models \varphi$, if $\pi, 0 \models \varphi$.

LTL_f is a variant of LTL interpreted over *finite traces* instead of infinite traces (De Giacomo and Vardi 2013). The syntax of LTL_f is the same as the syntax of LTL. We define $\pi, i \models \varphi$, stating that φ holds at position i, as for LTL, except that for the temporal operators we have:

- $\pi, i \models \bigcirc\varphi$ iff $i < \mathsf{last}(\pi)$ and $\pi, i+1 \models \varphi$;
- $\pi, i \models \varphi_1 \,\mathcal{U}\, \varphi_2$ iff there exists j such that $i \leq j \leq \mathsf{last}(\pi)$ and $\pi, j \models \varphi_2$, and for all $k, i \leq k < j$ we have that $\pi, k \models \varphi_1$.

where $\mathsf{last}(\pi)$ denotes the last position (i.e., index) in the finite trace π. In addition, we define the *weak next* operator $\bullet$ as abbreviation of $\bullet\varphi \equiv \neg\bigcirc\neg\varphi$. Note that, over finite traces, it does not holds that $\neg\bigcirc\varphi \not\equiv \bigcirc\neg\varphi$, but instead $\neg\bigcirc\varphi \equiv \bullet\neg\varphi$. We say that a trace *satisfies* an LTL_f formula φ, written as $\pi \models \varphi$, if $\pi, 0 \models \varphi$.

Automata. A *nondeterministic automaton* (NA, for short) is a tuple $\mathcal{N} = (\Sigma, Q, q_0, \delta, \alpha)$, where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $\alpha \subseteq Q^\omega$ is an acceptance condition. Given an infinite word $w = a_0a_1a_2\ldots \in \Sigma^\omega$, a *run* of $\mathcal{A}$ on w is a sequence $r = q_0q_1q_2\ldots \in Q^\omega$ starting at the initial state q_0 and $q_{i+1} \in \delta(q_i, a_i)$ for $i \geq 0$. An automaton $\mathcal{N}$ is *deterministic* (DA, for short), if $|\delta(q,a)| = 1$ for every $(q,a) \in Q \times \Sigma$, and we denote it by $\mathcal{D}$. A run r is *accepting* if $r \in \alpha$. The *language* of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of words

accepted by $\mathcal{A}$, that is, for which there exists an accepting run. For a given infinite sequence $r \in Q^\omega$, by $\inf(r) \subseteq Q$ we denote the set of symbols that occur infinitely often in r. In this work we specifically consider Büchi, reachability, safety, and reachability-safety conditions:

- *Büchi.* Given a set $T \subseteq Q$, $\text{Büchi}(T) = \{r \in Q^\omega \mid \inf(r) \cap T \neq \emptyset\}$.
- *Reachability.* Given a set $T \subseteq Q$, $\text{Reach}(T) = \{q_0 q_1 q_2 \ldots \in Q^\omega \mid \exists k \geq 0 : q_k \in T\}$.
- *Safety.* Given a set $T \subseteq Q$, $\text{Safe}(T) = \{q_0 q_1 q_2 \ldots \in Q^\omega \mid \forall k \geq 0 : q_k \in T\}$.
- *Reachability-Safety.* Given two sets $T_1, T_2 \subseteq Q$ corresponding to reachability and safety conditions, respectively, $\text{Reach–Safe}(T_1, T_2) = \{q_0 q_1 q_2 \ldots \in Q^\omega \mid \exists i \geq 0 : q_i \in T_1 \text{ and } \forall j, 0 \leq j \leq i : q_j \in T_2\}$ requires that a state in T_1 is visited at least once, and until then only states in T_2 are visited.

Notably, a DA with reachability condition defines a deterministic finite automaton (DFA), and a NA with Büchi condition defines a nondeterministic Büchi automaton (NBA).

We define the *complement* of a DA $\mathcal{D} = (\Sigma, Q, q_0, \delta, \alpha)$ as $\overline{\mathcal{D}} = (\Sigma, Q, q_0, \delta, Q^\omega \setminus \alpha)$. Note that $\mathcal{L}(\overline{\mathcal{D}}) = \Sigma^\omega \setminus \mathcal{L}(\mathcal{D})$. Note also that $Q^\omega \setminus \text{Reach}(T) = \text{Safe}(Q \setminus T)$ and $Q^\omega \setminus \text{Safe}(T) = \text{Reach}(Q \setminus T)$. Therefore, the complement of a DA with a reachability acceptance condition is a DA with a safety acceptance condition, and vice-versa. We also define the *bounded intersection* of two DAs $\mathcal{D} = (\Sigma, Q, q_0, \delta, \text{Reach}(R))$ and $\hat{\mathcal{D}} = (\Sigma, \hat{Q}, \hat{q}_0, \hat{\delta}, \text{Safe}(S))$ as $\mathcal{D} \cap \hat{\mathcal{D}} = (\Sigma, Q \times \hat{Q}, (q_0, \hat{q}_0), \delta', \alpha')$, where $\delta'((q, \hat{q}), a) = (\delta(q, a), \hat{\delta}(\hat{q}, a))$ and $\alpha' = \text{Reach–Safe}(R', S')$ such that $R' = \{(q, \hat{q}) \mid q \in R\}$ and $S' = \{(q, \hat{q}) \mid \hat{q} \in S\}$.

Games over DAs. Two-player games over DA are games consisting of two players, the *environment* and the *agent*. $\mathcal{X}$ and $\mathcal{Y}$ are disjoint sets of environment Boolean variables and agent Boolean variables, respectively. The *specification* of the game arena is given by a DA $\mathcal{D} = (2^{\mathcal{X} \cup \mathcal{Y}}, Q, q_0, \delta, \alpha)$, where $\delta : Q \times \Sigma \rightharpoonup Q$ is a *partial transition function*.

A position in the game is a state $q \in Q$. At first, the environment moves by setting $X \in 2^{\mathcal{X}}$, then the agent moves by setting $Y \in 2^{\mathcal{Y}}$, and the next position is updated to the state $\delta(q, X \cup Y)$. An *agent strategy* is a function $\sigma_{\texttt{ag}} : (2^{\mathcal{X}})^+ \to 2^{\mathcal{Y}}$, and an environment strategy is a function $\sigma_{\texttt{env}} : (2^{\mathcal{Y}})^* \to 2^{\mathcal{X}}$. A *trace* is a sequence $\pi = (X_0 \cup Y_0)(X_1 \cup Y_1) \cdots \in (2^{\mathcal{X} \cup \mathcal{Y}})^\omega$ over the alphabet $2^{\mathcal{X} \cup \mathcal{Y}}$. A trace π is *compatible* with an agent strategy $\sigma_{\texttt{ag}}$ if $\sigma_{\texttt{ag}}(X_0 X_1 \cdots X_i) = Y_i$ for every $i \geq 0$. A trace π is *compatible* with an environment strategy $\sigma_{\texttt{env}}$ if $\sigma_{\texttt{env}}(\epsilon) = X_0$ and $\sigma_{\texttt{env}}(Y_0 Y_1 \cdots Y_i) = X_{i+1}$ for every $i \geq 0$. The *unique* trace compatible with strategies $\sigma_{\texttt{ag}}$ and $\sigma_{\texttt{env}}$ is a *play* and denoted as $\text{play}(\sigma_{\texttt{ag}}, \sigma_{\texttt{env}})$. For a given play π, by $\pi^k = (X_0 \cup Y_0) \cdots (X_k \cup Y_k)$ we denote the *prefix* of π up to the k-th iteration. Sometimes, for simplicity, we write $\sigma_{\texttt{ag}}(\pi^k)$ in place of $\sigma_{\texttt{ag}}(X_0 X_1 \cdots X_k)$. An agent strategy $\sigma_{\texttt{ag}}$ is *winning* if for every environment strategy $\sigma_{\texttt{env}}$, we have that $\text{play}(\sigma_{\texttt{ag}}, \sigma_{\texttt{env}}) \in \mathcal{L}(\mathcal{D})$. By Win_{ag} (resp., Win_{env}) we denote the set of states from which the agent (resp., the environment) has a winning strategy. Depending on the actual winning condition α we get Büchi, reachability, safety, or reachability-safety games.

Safety and Liveness. A *safety* property excludes traces whose "badness" follows from a finite prefix. To define *safety* properties, we need to introduce the concept of *bad prefix*. Consider a property $P \subseteq \Sigma^\omega$ over an alphabet Σ. A finite word $x \in \Sigma^*$ is a *bad prefix* for P if and only if for all infinite words $y \in \Sigma^\omega$, we have $x \cdot y \notin P$. A property P is a *safety* property iff every $w \notin P$ has a bad prefix. A formula φ is a *safety formula* iff φ specifies a safety property.

A property P is a *liveness* property if for every word $x \in \Sigma^*$ there exists a word $y \in \Sigma^\omega$ such that $x \cdot y \in P$. A formula φ is a *liveness formula* iff φ specifies a *liveness* property.

Partitioning into Safety and Liveness. As shown in (Alpern and Schneider 1987; Kupferman and Vardi 2001), an NA in which every state has been made into a safe state, i.e., $\mathcal{N} = (\Sigma, Q, q_0, \delta, \text{Safe}(Q))$, is able to capture a safety property. This is because it only rejects an input by attempting an undefined transition, which terminates the run. Indeed, the following theorem holds:

Theorem 1 ((Alpern and Schneider 1987))**.** *Let $\mathcal{N} = (\Sigma, Q, q_0, \delta, \text{Büchi}(T))$ be an NA with Büchi condition and $\mathcal{N}' = (\Sigma, Q, q_0, \delta, \text{Safe}(Q))$. Then, $\mathcal{N}$ specifies a safety property if and only if $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{N}')$.*

Note that an NA with safety condition can be determinized by applying the standard *subset construction* (Rabin and Scott 1959; Kupferman and Vardi 2001), thus a safety formula can be recognized by a DA with safety condition.

In (Alpern and Schneider 1987), it is also shown that every formula specified by an NA with Büchi condition is equivalent to the conjunction of two automata specifying a safety property and a liveness property. Formally, we have:

Theorem 2 ((Alpern and Schneider 1987))**.** *Let $\mathcal{N} = (\Sigma, Q, q_0, \delta, \text{Büchi}(T))$, then $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{N}_{safe}) \cap \mathcal{L}(\mathcal{N}_{live})$ where $\mathcal{N}_{safe}$ specifies a safety property and $\mathcal{N}_{live}$ specifies a liveness property.*

Specifically, by Theorem 2, we can extract from an NA $\mathcal{N} = (\Sigma, Q, q_0, \delta, \text{Büchi}(T))$ the NA $\mathcal{N}_{safe} = (\Sigma, Q, q_0, \delta, \text{Safe}(Q))$ that recognizes the safety part of $\mathcal{N}$. In particular, for every prefix $x \in \Sigma^*$ that is not a bad prefix of $\mathcal{L}(\mathcal{N}_{safe})$, there always exists $y \in \Sigma^\omega$, such that $x \cdot y \in \mathcal{L}(\mathcal{N})$. This is because by definition, $\mathcal{N}$ and $\mathcal{N}_{safe}$ share the same automaton structure. Moreover, $x \cdot y \in \mathcal{L}(\mathcal{N}_{live})$, since $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{N}_{safe}) \cap \mathcal{L}(\mathcal{N}_{live})$.

Synthesis under Environment Specifications. In standard synthesis the environment is free to choose an arbitrary move at each step, but in AI typically the agent has some knowledge of how the environment works, which it can exploit in order to enforce the goal, specified as an LTL_f formula $\varphi^a_{\texttt{task}}$. Here, we specify the environment behaviour by an LTL formula φ^e and call it *environment specification*. In particular, φ^e specifies the set of environment strategies that ensure φ^e. Moreover, we require that φ^e must be *environment realizable*, i.e., the set of environment strategies that enforce φ^e is nonempty.

Given an LTL formula φ, we say that an agent strategy (*resp.,* environment strategy) *enforces* φ, written $\sigma_{\mathtt{ag}} \rhd \varphi$ (*resp.,* $\sigma_{\mathtt{env}} \rhd \varphi$), if for every environment strategy $\sigma_{\mathtt{env}}$ (*resp.,* agent strategy $\sigma_{\mathtt{ag}}$), we have $\mathsf{play}(\sigma_{\mathtt{ag}}, \sigma_{\mathtt{env}}) \models \varphi$.

The problem of *synthesis under environment specifications* is to find an agent strategy $\sigma_{\mathtt{ag}}$ such that

$$\forall \sigma_{\mathtt{env}} \rhd \varphi^e, \mathsf{play}(\sigma_{\mathtt{ag}}, \sigma_{\mathtt{env}})^k \models \varphi^a_{\mathsf{task}} \text{ for some } k \in \mathbb{N}.$$

As shown in (Aminof et al. 2019), this can be reduced to solving the synthesis problem for the implication $\varphi^e \rightarrow ltl(\varphi^a_{\mathsf{task}})$, with $ltl(\varphi^a_{\mathsf{task}})$ being a suitable LTL_f-to-LTL transformation (De Giacomo and Vardi 2013), which is 2EXPTIME-complete (Pnueli and Rosner 1989).

3 Synthesis with Mandatory Stop Actions

In this paper we study synthesis under environment specifications in which the agent at a certain point of the trace must stop performing any action. Hence, based on this condition, the agent must achieve her goal and then stop the execution of any action from that point on. However, this mandatory choice to stop leads the agent to not consider the part of the environment specifications that require some progress, i.e., specified as liveness formulas, as they do not constraint the finite behaviour but require a certain condition on the infinite behaviour, and the stop can occur, in a finite number of steps, before this constraint is satisfied.

In order to define the synthesis problem with mandatory stop actions, we first need to include into the agent strategy this stop condition. More specifically, we assume that every action of the agent is an assignment over $\mathcal{Y}$, and stop is one of them. In particular, wlog, stop is encoded as an assignment where all variable in $\mathcal{Y}$ are set to $false$, i.e., $\mathsf{stop} = \bigwedge_{y \in \mathcal{Y}} \neg y$.

We redefine agent strategies as follows. An agent strategy $\sigma_{\mathtt{ag}}$ is *stopping* iff, for every play $\pi \in (2^{\mathcal{X} \cup \mathcal{Y}})^\omega$ there exists $k \in \mathbb{N}$ such that $\sigma_{\mathtt{ag}}(\pi^j) = \mathsf{stop}$ for every $j \geq k$ and $\sigma_{\mathtt{ag}}(\pi^h) \neq \mathsf{stop}$ for every $h < k$. That is, for each play compatible with $\sigma_{\mathtt{ag}}$, the action stop is eventually executed and held forever. From now on, we restrict agent strategies to be stopping strategies and therefore reconsider the synthesis problem for the agent goal $\varphi^a_{\mathsf{task}}$ under the environment specification φ^e accordingly.

Definition 1 (Synthesis with mandatory stop actions)**.**

1. *The problem is described as a tuple* $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi^e, \varphi^a_{\mathsf{task}} \rangle$*, where* $\mathcal{X}$ *and* $\mathcal{Y}$ *are two disjoint sets of Boolean variables, controlled respectively by the environment and the agent,* $\varphi^a_{\mathsf{task}}$ *is an* LTL_f *formula, and* φ^e *is an* LTL *formula.*
2. *An agent strategy* $\sigma_{\mathtt{ag}}$ *realizes* $\varphi^a_{\mathsf{task}}$ *under environment specification* φ^e *if for all environment strategies* $\sigma_{\mathtt{env}} \rhd \varphi^e$*,* $\pi = \mathsf{play}(\sigma_{\mathtt{ag}}, \sigma_{\mathtt{env}})$ *has a prefix* π^k *such that* $\sigma_{\mathtt{ag}}(\pi^k) = \mathsf{stop}$ *and* $\pi^k \models \varphi^a_{\mathsf{task}}$*, and for all* $h < k$ *we have* $\sigma_{\mathtt{ag}}(\pi^h) \neq \mathsf{stop}$*.*
3. *Solving* $\mathcal{P}$ *consists in finding an agent strategy that realizes* $\varphi^a_{\mathsf{task}}$ *under specification* φ^e*.*

This class of synthesis problem naturally reflects the structure of many reactive systems in practice. We illustrate this with an example.

Example 1. *A cleaning robot is deployed in a shelter on Mars. The robot is sent to clean the dust in lab whenever it is needed. The agent's goal is to clean the lab and leave. This scenario can be represented by the following specification.*

The robot controls the actions: robot_wait, get_out, clean_dust. The environment controls the fluents: RobotOut and Dust.

The environment specification can be written as:

$$\begin{aligned}\varphi^e = & \neg \mathit{RobotOut} \wedge \mathit{Dust} \\ & \wedge \Box((\neg \mathit{RobotOut} \wedge \mathit{clean\_dust}) \\ & \quad \rightarrow (\bigcirc(\neg \mathit{RobotOut} \wedge \neg \mathit{Dust}))) \\ & \wedge (\Box\Diamond \mathit{robot\_wait} \rightarrow \Box\Diamond \neg \mathit{Dust}) \\ & \wedge \Box(\mathit{robot\_wait} \rightarrow (\mathit{RobotOut} \equiv \bigcirc \mathit{RobotOut})) \\ & \wedge \Box(\mathit{get\_out} \rightarrow (\mathit{Dust} \equiv \bigcirc \mathit{Dust})) \\ & \wedge \Box((\neg \mathit{RobotOut} \wedge \mathit{get\_out}) \rightarrow \bigcirc \mathit{RobotOut})\end{aligned}$$

The agent goal is:

$$\varphi^a_{\mathsf{task}} = \Diamond(\mathit{RobotOut} \wedge \neg \mathit{Dust}).$$

A possible plan for the robot is given by:

$$(\mathit{robot\_wait})^\star; \mathit{clean\_dust}; \mathit{get\_out}.$$

However, in order to achieve its goal, the robot cannot wait forever such that relying on the environment to clean the lab, as the environment can make this happen after the robot performs its mandatory stop action.

Reduction to LTL Synthesis. A naive approach to solve $\mathcal{P}$ is to reduce it to standard LTL synthesis, see e.g. (Camacho, Bienvenu, and McIlraith 2018; Zhu et al. 2020). This is because LTL_f can be viewed as a fragment of LTL, as shown in (De Giacomo and Vardi 2013), and thus the agent goal $\varphi^a_{\mathsf{task}}$ can be translated to LTL formula $ltl(\varphi^a_{\mathsf{task}})$ by introducing a new agent variable alive. Intuitively, this variable indicates the satisfaction of $\varphi^a_{\mathsf{task}}$, hence it stays $true$ until $\varphi^a_{\mathsf{task}}$ is satisfied, and stays $false$ since then.

It should be noted that the reduction technique introduced in (Camacho, Bienvenu, and McIlraith 2018; Zhu et al. 2020) can not be directly applied here, since we restrict the agent strategy to be *stopping*. In addition to the translation of $ltl(\varphi^a_{\mathsf{task}})$, we need to restrict the agent behaviour to consider the mandatory stop action. To do so, we need to add the LTL formula $(\neg\mathsf{stop}\,\mathcal{U}\,\Box\mathsf{stop})$. Moreover, we also need to relate alive to stop by having $\Box(\mathsf{alive} \equiv \neg\mathsf{stop})$, since both indicate the satisfaction of $\varphi^a_{\mathsf{task}}$. In particular, expressing the mandatory stop action should be independent from synthesizing agent goal under environment specifications. Therefore, $((\neg\mathsf{stop}\,\mathcal{U}\,\Box\mathsf{stop}) \wedge \Box(\mathsf{alive} \equiv \neg\mathsf{stop}))$ is applied to restrict the complete implication of $(\varphi^e \rightarrow ltl(\varphi^a_{\mathsf{task}}))$, rather than $ltl(\varphi^a_{\mathsf{task}})$ itself. As a consequence, the reduced LTL formula is $\psi = (\varphi^e \rightarrow ltl(\varphi^a_{\mathsf{task}})) \wedge (\neg\mathsf{stop}\,\mathcal{U}\,\Box\mathsf{stop}) \wedge \Box(\mathsf{alive} \equiv \neg\mathsf{stop}))$.

Theorem 3. *Let* $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi^e, \varphi^a_{\mathsf{task}} \rangle$ *be a synthesis problem with mandatory stop actions and* $\mathcal{P}^{LTL} = \langle \mathcal{X}, \mathcal{Y} \cup \{\mathsf{alive}\}, \psi \rangle$ *be the reduced* LTL *synthesis problem, where* $\psi = (\varphi^e \rightarrow ltl(\varphi^a_{\mathsf{task}})) \wedge ((\neg\mathsf{stop}\,\mathcal{U}\,\Box\mathsf{stop}) \wedge \Box(\mathsf{alive} \equiv \neg\mathsf{stop}))$*. We have* $\mathcal{P}$ *is realizable if and only if* $\mathcal{P}^{LTL}$ *is realizable.*

Following (Camacho, Bienvenu, and McIlraith 2018; Zhu et al. 2020; De Giacomo et al. 2020b), it is trivial to show that every agent winning strategy of $\mathcal{P}$ is also an agent winning strategy of $\mathcal{P}^{LTL}$, and vice-versa.

However reducing to LTL synthesis has not shown promising results. Hence specific techniques have been proposed that try to avoid, if possible, the Büchi determinization and the solution of parity games, see e.g., (Camacho, Bienvenu, and McIlraith 2018; De Giacomo et al. 2020b). This is what we will do for our case as well.

An important implication of dealing with the need of the agent to stop is that, when solving the synthesis problem $\mathcal{P}$, we can reduce the environment LTL specification φ^e to its safety part only, due to the fact that the agent cannot exploit the liveness part to achieve her goal, since satisfying the liveness part can happen after the agent has executed the action stop. Indeed, we know by Theorem 2 that every $\mathcal{N}$ with Büchi condition recognizing an LTL formula can be decomposed into a safety part $\mathcal{N}_{safe}$ and a liveness part $\mathcal{N}_{live}$, i.e., $\varphi^e = \varphi^e_{\mathsf{safe}} \wedge \varphi^e_{\mathsf{live}}$, where $\mathcal{L}(\varphi^e_{\mathsf{safe}}) = \mathcal{L}(\mathcal{N}_{safe})$, and $\mathcal{L}(\varphi^e_{\mathsf{live}}) = \mathcal{L}(\mathcal{N}_{live})$. It is important to note that, $\varphi^e_{\mathsf{safe}}$ and $\varphi^e_{\mathsf{live}}$ are not independent from each other. Instead, both are related to the original LTL formula φ^e. More specifically, for every prefix $x \in \Sigma^*$ that is not a bad prefix of $\mathcal{L}(\varphi^e_{\mathsf{safe}})$, there always exists $y \in \Sigma^\omega$, such that $x \cdot y \in \mathcal{L}(\varphi^e_{\mathsf{live}} \wedge \varphi^e_{\mathsf{safe}})$.

We show next that in the case of LTL_f goals, we can disregard $\varphi^e_{\mathsf{live}}$ from φ^e and use only $\varphi^e_{\mathsf{safe}}$ as environment specification.

Theorem 4. *Let $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi^e, \varphi^a_{\mathsf{task}} \rangle$ be the synthesis problem with mandatory stop actions and σ_{ag} an agent strategy. Then, σ_{ag} realizes $\mathcal{P}$ if and only if σ_{ag} realizes $\hat{\mathcal{P}} = \langle \mathcal{X}, \mathcal{Y}, \varphi^e_{\mathsf{safe}}, \varphi^a_{\mathsf{task}} \rangle$.*

Proof. We prove the two implications separately.

($\Rightarrow$) We prove this direction by contradiction. Assume that there exists an agent strategy σ_{ag} that realizes $\mathcal{P}$ but does not realize $\hat{\mathcal{P}}$. Therefore, there exists an environment strategy $\sigma'_{\mathsf{env}} \rhd \varphi^e_{\mathsf{safe}}$ such that $\pi' = \mathsf{play}(\sigma_{\mathsf{ag}}, \sigma'_{\mathsf{env}})^k \not\models \varphi^a_{\mathsf{task}}$ for k being the first iteration on which σ_{ag} plays the action stop. Now, since $\sigma'_{\mathsf{env}} \rhd \varphi^e_{\mathsf{safe}}$, it holds that $\mathsf{play}(\sigma_{\mathsf{ag}}, \sigma'_{\mathsf{env}})^k$ is not a bad prefix for $\varphi^e_{\mathsf{safe}}$ and thus there exists a strategy $\sigma_{\mathsf{env}} \rhd \varphi^e = \varphi^e_{\mathsf{safe}} \wedge \varphi^e_{\mathsf{live}}$ such that $\pi^k = \mathsf{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}})^k = \mathsf{play}(\sigma_{\mathsf{ag}}, \sigma'_{\mathsf{env}})^k = \pi'^k$. Therefore, it holds that $\pi^k \not\models \varphi^a_{\mathsf{task}}$, which is a contradiction as $\sigma_{\mathsf{env}} \rhd \varphi^e$ and σ_{ag} realizes $\mathcal{P}$, which implies $\pi^k \models \varphi^a_{\mathsf{task}}$.

($\Leftarrow$) This direction is a direct consequence of the fact that φ^e implies $\varphi^e_{\mathsf{live}}$. Indeed, assume that σ_{ag} realizes $\hat{\mathcal{P}}$. Consider now an environment strategy σ_{env} such that $\sigma_{\mathsf{env}} \rhd \varphi^e$. In particular, it holds that $\sigma_{\mathsf{env}} \rhd \varphi^e_{\mathsf{safe}}$ and so, since σ_{ag} realizes $\hat{\mathcal{P}}$, that $\mathsf{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathsf{env}})^k \models \varphi^a_{\mathsf{task}}$, with k being the last iteration in which the agent does not execute the action stop, which proves the statement. □

As shown in Theorem 2, given the NA $\mathcal{N}^e = (\Sigma, Q, q_0, \delta, \text{Büchi}(T))$ of φ^e we can extract from $\mathcal{N}^e$ the NA with safety condition $\mathcal{N}^e_s = (\Sigma, Q, q_0, \delta, \text{Safe}(Q))$ that recognizes $\varphi^e_{\mathsf{safe}}$, and $\mathcal{N}^e_s$ can be determinized using the subset construction instead of the more complex Büchi determinization. Hence, Theorem 4, together with Theorem 2, can be exploited to define a specific synthesis technique with a deep impact on the efficiency of solving the problem $\mathcal{P}$.

4 Synthesis Technique

To solve the problem $\hat{\mathcal{P}}$ we first observe that the agent's goal is to satisfy $\neg\varphi^e_{\mathsf{safe}} \vee \varphi^a_{\mathsf{task}}$. However, being $\varphi^e_{\mathsf{safe}}$ environment realizable, i.e., the set ξ of environment strategies that realize $\varphi^e_{\mathsf{safe}}$ is nonempty, we can narrow down to ξ when solving $\hat{\mathcal{P}}$. Moreover, we know that $\varphi^e_{\mathsf{safe}}$ can be represented by a DA with safety condition (see Section 3), and $\varphi^a_{\mathsf{task}}$ can be represented by a DA with reachability condition (De Giacomo and Vardi 2015). Therefore, we devise a synthesis technique for $\mathcal{P}$ which (*i*) restricts to considering only the set ξ of environment strategies by computing the safe region for the environment, (*ii*) builds the DA of $\varphi^a_{\mathsf{task}}$, and (*iii*) combines the two DAs and solves the reachability game over the resulting automaton.

Hence, given synthesis problem $\hat{\mathcal{P}} = \langle \mathcal{X}, \mathcal{Y}, \varphi^e_{\mathsf{safe}}, \varphi^a_{\mathsf{task}} \rangle$, we can solve $\hat{\mathcal{P}}$ by taking the following stages.

Stage 1: Compute the safe region for the environment. Given the environment specification φ^e, we first build the NA with Büchi condition $\mathcal{N}^e = (\Sigma, Q^e, q^e_0, \delta^e, \text{Büchi}(T))$ of φ^e, such that $\mathcal{L}(\mathcal{N}^e) = \mathcal{L}(\varphi^e)$. Then, by Theorem 2, we can extract from $\mathcal{N}^e$ the NA $\mathcal{N}^e_s = (\Sigma, Q^e, q^e_0, \delta^e, \text{Safe}(Q^e))$ that recognizes the safety part of φ^e. At the end, we determinize $\mathcal{N}^e_s$, by using the subset construction, into $\mathcal{D}^e_s = (\Sigma, Q^e_s \cup \{q_{sink}\}, q_{0^e_s}, \delta^e_s, \text{Safe}(Q^e_s))$ that accepts a trace π if and only if $\pi \models \varphi^e_{\mathsf{safe}}$. Now, we solve a safety game for the environment over $\mathcal{D}^e_s$ by computing the environment winning states Win_{env}, and then restrict $\mathcal{D}^e_s$ with Win_{env}, obtaining $\mathcal{D}'^e_s = (\Sigma, Win_{env}, \delta'^e_s, \text{Safe}(Win_{env}))$, where for $(q, a) \in Win_{env} \times \Sigma$, $\delta'^e_s(q, X \cup Y)$ is undefined if there exists $Y' \in 2^{\mathcal{Y}}$ such that $\delta^e_s(q, X \cup Y') \notin Win_{env}$, $\delta'^e_s(q, X \cup Y) = \delta^e_s(q, X \cup Y)$ otherwise.

Stage 2: Compute DA for the goal. Here we build the DA of $\varphi^a_{\mathsf{task}}$ that is with reachability condition $\mathcal{D}^a_t = (\Sigma, Q^a_t, q_{0^a_t}, \delta^a_t, \text{Reach}(R^a_t))$, we have that $\pi \in \mathcal{L}(\mathcal{D}^a_t)$ if and only if $\pi \models \varphi^a_{\mathsf{task}}$.

Stage 3: Solve the synthesis for $\varphi^a_{\mathsf{task}}$ under $\varphi^e_{\mathsf{safe}}$. Having $\mathcal{D}'^e_s$ and $\mathcal{D}^a_t$, we do the intersection between $\mathcal{D}'^e_s$ and $\mathcal{D}^a_t$ thus obtaining $\mathcal{D} = (\Sigma, Win_{env} \times Q^a_t, (q_{0^e_s}, q_{0^a_t}), \delta, \text{Reach}(R))$, where for $q \in Win_{env}$, $\hat{q} \in Q^a_t$, and $a \in \Sigma$, $\delta((q, \hat{q}), a) = (\delta'^e_s(q, a), \delta^a_t(\hat{q}, a))$, and $\text{Reach}(R) = \{(q_{0^e_s}, q_{0^a_t})(q_{1^e_s}, q_{1^a_t}) \ldots \in (Win_{env} \times Q^a_t)^\omega \mid q_{0^a_t}, q_{1^a_t}, \ldots \in R^a_t\}$. Finally, we solve a reachability game for the agent over $\mathcal{D}$ which returns an agent winning strategy, if one exists.

Correctness. Now we prove the correctness of the algorithm above. By Theorem 4 it suffices to show that, following the stages above, we are able to obtain an agent winning strategy for the problem $\hat{\mathcal{P}}$, if one exists.

Theorem 5. *Let $\hat{\mathcal{P}} = (\mathcal{X}, \mathcal{Y}, \varphi^e_{\mathsf{safe}}, \varphi^a_{\mathsf{task}})$ be a synthesis problem with mandatory stop actions. Then, $\hat{\mathcal{P}}$ can be reduced to solving the reachability game for the agent over $\mathcal{D}$.*

Proof. Observe that $\mathcal{D}$ is obtained as the intersection of $\mathcal{D}'^{e}_{s}$ which is in turn obtained by keeping from $\mathcal{D}^{e}_{s}$ only the states from which the environment can ensure $\varphi^{e}_{\mathsf{safe}}$. This means that $\mathcal{D}$ can only accept plays that are induced by environment strategies $\sigma_{\mathtt{env}}$ such that $\sigma_{\mathtt{env}} \rhd \varphi^{e}_{\mathsf{safe}}$.

Now, consider agent winning strategy $\sigma_{\mathtt{ag}}$ in $\mathcal{D}$. It holds that $\mathsf{play}(\sigma_{\mathtt{ag}}, \sigma_{\mathtt{env}})$ is fully contained in $\mathcal{D}$. Moreover, being $\sigma_{\mathtt{ag}}$ winning, it holds that $\mathsf{play}(\sigma_{\mathtt{ag}}, \sigma_{\mathtt{env}})^k \models \varphi^{a}_{\mathsf{task}}$, for some $k \in \mathbb{N}$. As this holds for every environment strategy $\sigma_{\mathtt{env}} \rhd \varphi^{e}_{\mathsf{safe}}$, we obtain that $\sigma_{\mathtt{ag}}$ realizes $\hat{\mathcal{P}}$. □

Computational properties. We now discuss the computational properties of our synthesis algorithm. Specifically, Stage 1 builds the corresponding DA with safety condition for $\varphi^{e}_{\mathsf{safe}}$ in double exponential time in the size of φ^{e}. This allows us to exploit subset construction and minimization rather than Büchi determinization, making it more scalable in practice, as for LTL$_f$ synthesis. Moreover, it solves a safety game that takes polynomial time in the size of the game. Stage 2 builds the DA with reachability condition in double exponential time in the size of $\varphi^{a}_{\mathsf{task}}$, again, going through subset construction and minimization. Finally, Stage 3 just requires solving adversarial reachability, compared to solving parity games in the case of general LTL environment specifications. Then, the overall complexity of the algorithm is given by the following.

Theorem 6. *The above algorithm solves synthesis with mandatory stop actions problem $\mathcal{P} = (\mathcal{X}, \mathcal{Y}, \varphi^{e}, \varphi^{a}_{\mathsf{task}})$ in* 2EXPTIME *(the problem is indeed* 2EXPTIME*-complete).*

Note that LTL$_f$ synthesis (without environment specifications) is a special case of our problem. Indeed in this case the mandatory stop action requirement is irrelevant. This provides a matching lower-bound to the complexity specified in the theorem above.

5 Adding Agent Safety Goals

We now introduce agent safety goals into the framework. Safety goals are constraints that can be applied to the agent such that its behaviour always remains in the desired boundaries. We enrich our synthesis framework by adding safety goals for the agent. We use $\varphi^{a}_{\mathsf{safe}}$ to denote the safety properties specified in LTL that the agent is required to follow. Therefore, we are interested in solving the synthesis for $\langle \varphi^{a}_{\mathsf{task}}, \varphi^{a}_{\mathsf{safe}} \rangle$ under environment specification φ^{e}, where $\varphi^{a}_{\mathsf{task}}$ is expressed as an LTL$_f$ formula, while $\varphi^{a}_{\mathsf{safe}}$ and φ^{e} are LTL formulas.

Agent Safety Goals. The agent goal $\langle \varphi^{a}_{\mathsf{task}}, \varphi^{a}_{\mathsf{safe}} \rangle$ requires the agent to satisfy $\varphi^{a}_{\mathsf{task}}$ over a finite prefix and $\varphi^{a}_{\mathsf{safe}}$ over the entire (infinite) play. If we consider them separately, $\varphi^{a}_{\mathsf{task}}$ specifies the goal that the agent needs to reach in a *finite* number of steps, and $\varphi^{a}_{\mathsf{safe}}$, nevertheless, requires that "bad things" never happens in an *infinite* number of steps. First of all, $\varphi^{a}_{\mathsf{safe}}$ should not be violated until $\varphi^{a}_{\mathsf{task}}$ gets accomplished. What about after that? We illustrate this concern with an example.

Example 2. *The cleaning robot clearly has a peaceful life in the shelter. Being secured but also trapped in such a shelter, the robot wants to escape to explore the universe. However, once the airlock is open with emergency core off, the shelter will be air outage in 2 time steps. The robot is also able to manually activate the emergency core to keep it running. For security, the shelter itself also activates the emergency core from time to time to pump some air. This scenario can be represented by the following specification.*

The robot can take actions: robot_wait, open_airlock, act_emergency_core, and robot_escape. The environment controls the following set of fluents: $\mathcal{F} = \{AirLockIsOpen, AirOutage, EmergencyCoreOn, RobotFree\}$.

The environment specification φ^{e} can be written as:

$$
\begin{aligned}
\varphi^{e} = {} & \neg \mathit{AirLockIsOpen} \wedge \neg \mathit{AirOutage} \\
& \wedge \neg \mathit{RobotFree} \wedge \neg \mathit{EmergencyCoreOn} \\
& \wedge \Box(\mathit{robot\_wait} \rightarrow \bigwedge_{f \in \mathcal{F}} f \equiv \bigcirc f) \\
& \wedge \Box(\mathit{open\_airlock} \rightarrow (\bigcirc \mathit{AirLockIsOpen} \wedge \\
& \qquad \bigwedge_{f \neq \mathit{AirLockIsOpen}} f \equiv \bigcirc f)) \\
& \wedge \Box((\mathit{AirLockIsOpen} \wedge \neg \mathit{EmergencyCoreOn}) \equiv \\
& \qquad \bigcirc\bigcirc \mathit{AirOutage}) \\
& \wedge \Box(\mathit{act\_emergency\_core} \rightarrow (\bigcirc \mathit{EmergencyCoreOn} \wedge \\
& \qquad \bigwedge_{f \neq \mathit{EmergencyCoreOn}} f \equiv \bigcirc f)) \\
& \wedge \Box(\mathit{robot\_escape} \rightarrow \bigcirc \mathit{RobotFree}) \\
& \wedge \Box\Diamond \mathit{EmergencyCoreOn}
\end{aligned}
$$

The robot task goal is

$$\varphi^{a}_{\mathsf{task}} = \Diamond \mathit{RobotFree}$$

The robot safety goal is

$$\varphi^{a}_{\mathsf{safe}} = \Box\neg \mathit{AirOutage} \wedge \Box(\mathit{robot\_escape} \rightarrow \mathit{AirLockIsOpen})$$

If the robot only cares about AirOutage before it gets free, it can just open the airlock without activating the emergency core, and then get out, which specifies the plan:

$$(\mathit{robot\_wait})^{\star}; \mathit{open\_airlock}; \mathit{robot\_escape}.$$

But this might lead to an inevitable damage to all the humans in the shelter, due to the air outage in 2 time steps. Moreover, it cannot rely on the shelter to activate the emergency core for pumping air, since this can happen after it leaves. Being a responsible robot, it better takes the plan:

$$(\mathit{robot\_wait})^{\star}; \mathit{open\_airlock}; \mathit{act\_emergency\_core}; \mathit{robot\_escape}.$$

Note that once the robot leaves the shelter, it is not able to do anything to prevent the air outage anymore.

As this example illustrates, it is not enough for an agent to only care about the satisfaction of its safety goal until accomplishing its task. Instead, the agent still needs to maintain it forever. Nevertheless it can not perform any other actions rather than stop.

Adding Agent Safety Goals. We formulate the problem setting described above as follows.

Definition 2 (Synthesis with mandatory stop actions and agent safety goals)**.**

1. *The problem is described as a tuple* $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi^e, Goal \rangle$*, where* $\mathcal{X}$ *and* $\mathcal{Y}$ *are two disjoint sets of Boolean variables, controlled respectively by the environment and the agent,* φ^e *in an* LTL *formula, and* $Goal = \langle \varphi^a_{\mathsf{task}}, \varphi^a_{\mathsf{safe}} \rangle$ *is defined as a pair where* $\varphi^a_{\mathsf{task}}$ *is an* LTL_f *formula and* $\varphi^a_{\mathsf{safe}}$ *is a safety* LTL *formula.*
2. *An agent strategy* σ_{ag} *realizes* $Goal$ *under environment specification* φ^e *if for all environment strategies* $\sigma_{\mathtt{env}} \rhd \varphi^e$*,* $\pi = \mathsf{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathtt{env}})$ *has a prefix* π^k *such that* $\sigma_{\mathsf{ag}}(\pi^k) = \mathsf{stop}$ *and* $\pi^k \models \varphi^a_{\mathsf{task}}$*, and for all* $h < k$ *we have* $\sigma_{\mathsf{ag}}(\pi^h) \neq \mathsf{stop}$*. Moreover,* $\pi \models \varphi^a_{\mathsf{safe}}$*.*
3. *Solving* $\mathcal{P}$ *consists in finding an agent strategy that realizes* $\langle \varphi^a_{\mathsf{task}}, \varphi^a_{\mathsf{safe}} \rangle$ *under specification* φ^e*.*

Reduction to LTL Synthesis. Despite that the enriched synthesis problem includes agent safety goals, the alternative approach of reducing to LTL synthesis still works. Based on the reduction technique presented in Section 3, we can reduce the synthesis problem $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi^e, Goal \rangle$, where $Goal = \langle \varphi^a_{\mathsf{task}}, \varphi^a_{\mathsf{safe}} \rangle$, to LTL synthesis as well.

Theorem 7. *Let* $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi^e, Goal \rangle$ *be a synthesis problem with mandatory stop actions, where* $Goal = \langle \varphi^a_{\mathsf{task}}, \varphi^a_{\mathsf{safe}} \rangle$ *and* $\mathcal{P}^{LTL} = \langle \mathcal{X}, \mathcal{Y} \cup \{\mathsf{alive}\}, \psi \rangle$ *be the reduced* LTL *synthesis problem, where* $\psi = (\varphi^e \rightarrow ltl(\varphi^a_{\mathsf{task}}) \wedge \varphi^a_{\mathsf{safe}}) \wedge (\neg \mathsf{stop}\, \mathcal{U}\, \Box \mathsf{stop}) \wedge \Box(\mathsf{alive} \equiv \neg \mathsf{stop}))$*. We have* $\mathcal{P}$ *is realizable if and only if* $\mathcal{P}^{LTL}$ *is realizable.*

In Section 3, we pointed out how to translate $\varphi^a_{\mathsf{task}}$ to LTL. Regarding the agent safety goal $\varphi^a_{\mathsf{safe}}$, since it is specified in LTL, it can be directly conjuncted with $ltl(\varphi^a_{\mathsf{task}})$ thus having $ltl(\varphi^a_{\mathsf{task}}) \wedge \varphi^a_{\mathsf{safe}}$ on the rightside of the implication. Then we can show that every agent winning strategy of $\mathcal{P}$ is also an agent winning strategy of $\mathcal{P}^{LTL}$, and vice-versa, which is analogous to Theorem 3.

6 Synthesis Technique

Due to the unsatisfying scalability of its algorithms, the reduction to LTL synthesis is not a promising direction for solving synthesis with agent's safety goals. In this section, we propose a different approach that again circumvents the difficulties of Büchi automata determinization. We first give an overview of our proposed technique, and then present it in full details.

Given a synthesis problem $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi^e, Goal \rangle$, where $Goal = \langle \varphi^a_{\mathsf{task}}, \varphi^a_{\mathsf{safe}} \rangle$, as defined in Section 5. A strategy σ_{ag} for the agent is winning if, for every environment strategy $\sigma_{\mathtt{env}}$ such that $\sigma_{\mathtt{env}} \rhd \varphi^e$, it holds that the play $\pi = \mathsf{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathtt{env}})$ satisfies $\varphi^a_{\mathsf{safe}}$ and has a finite prefix π^k that satisfies $\varphi^a_{\mathsf{task}}$. To synthesize such a strategy, besides the finite prefix until $\varphi^a_{\mathsf{task}}$ holds, we should also consider the infinite suffix since $\varphi^a_{\mathsf{task}}$ gets satisfied, such that $\varphi^a_{\mathsf{safe}}$ never gets violated. Therefore, we split such synthesis problem into two phases. In phase I we focus on the satisfaction of $\varphi^a_{\mathsf{task}}$ that does not violate $\varphi^a_{\mathsf{safe}}$, that is, no bad prefixes of the safety condition are reached. In phase II we concentrate on the remainder of the execution, that is, after $\varphi^a_{\mathsf{task}}$ has been accomplished and the agent worries only about the non-violation of $\varphi^a_{\mathsf{safe}}$. Note that in phase II the agent is only allowed to play the stop action, and so it needs to enter such phase after making sure that such obligation will be enough to prevent the environment to violate $\varphi^a_{\mathsf{safe}}$.

We devise a synthesis technique that addresses the two phases in reverse order. Specifically, we first compute a set of so-called "Non-stop" states, from which if the agent plays action stop forever, the environment is able to break $\varphi^a_{\mathsf{safe}}$ in the future. Then we make use of these "Non-stop" states to solve a suitable reachability game, which allows the agent to satisfy $\varphi^a_{\mathsf{task}}$ while not violating $\varphi^a_{\mathsf{safe}}$ thus addressing phase I, and switching to phase II avoiding these "Non-stop" states.

In particular, we perform the following stages: (*i*) compute the safe region for the environment, where the environment is guaranteed to satisfy $\varphi^e_{\mathsf{safe}}$; (*ii*) collect "Non-stop" states for the agent, from which if the agent plays action stop, the environment is able to ultimately break $\varphi^a_{\mathsf{safe}}$; (*iii*) compute the DA for the agent goal, which expresses the satisfaction of $\varphi^a_{\mathsf{task}}$ and $\varphi^a_{\mathsf{safe}}$, and considers the "Non-stop" states for the agent to avoid; (*iv*) solve the synthesis for $Goal$ under φ^e, which can be done by solving a reachability game for the agent. We now present each stage in detail.

Stage (*i*): Compute the safe region for the environment. Consider the environment specification φ^e. Following the approach of the first stage described in Section 3, we obtain the NA $\mathcal{N}^e$ such that $\mathcal{L}(\mathcal{N}^e) = \mathcal{L}(\varphi^e)$, the DA $\mathcal{D}^e_s$ such that $\mathcal{L}(\mathcal{D}^e_s) = \mathcal{L}(\varphi^e_{\mathsf{safe}})$, and the DA $\mathcal{D}'^e_s$ consisting of states where the environment is guaranteed to satisfy $\varphi^e_{\mathsf{safe}}$.

Stage (*ii*): Collect "Non-stop" states for the agent. The set of "Non-stop" states of the agent are those from which the environment can satisfy φ^e, but violate $\varphi^a_{\mathsf{safe}}$, assuming that the agent plays stop forever. Being a safety property, the violation of $\varphi^a_{\mathsf{safe}}$ must happen in a finite number of steps, that is to say, violated by a finite bad prefix. Therefore, we can build a DA that captures all the bad prefixes of $\varphi^a_{\mathsf{safe}}$, with a pure reachability condition. To do so, we first obtain the DA $\mathcal{D}^a_s = (\Sigma, Q^a_s \cup \{q_{sink}\}, q_0{}^a_s, \delta^a_s, \mathrm{Safe}(Q^a_s))$ of $\varphi^a_{\mathsf{safe}}$. Intuitively, $\mathcal{D}^a_s$ rejects a trace π if it has a bad prefix, i.e., whose run leads to q_{sink}. We thus complement $\mathcal{D}^a_s$ and obtain an automaton defined as $\mathcal{D}_{bad} = \overline{\mathcal{D}^a_s} = (\Sigma, Q^a_s \cup \{q_{sink}\}, q_0{}^a_s, \delta^a_s, \mathrm{Reach}(\{q_{sink}\}))$.

Lemma 1. *Let* $\mathcal{D} = (\Sigma, Q \cup \{q_{sink}\}, q_0, \delta, \mathrm{Safe}(Q))$ *be a* DA *of a safety property* φ*. We have that* $\pi \in \mathcal{L}(\overline{\mathcal{D}})$ *iff* π *is a bad prefix of* φ*.*

Proof. Let r be the corresponding run of π on $\mathcal{L}(\overline{\mathcal{D}})$. By definition, $\pi \in \mathcal{L}(\overline{\mathcal{D}})$ such that there exists $i \geq 0$, $r_i = q_{sink}$. Due to the fact that the automata structure of $\mathcal{D}$ and $\overline{\mathcal{D}}$ is the same. Therefore, r corresponds to the run of π on $\mathcal{D}$ as well, such that $r \notin \mathrm{Safe}(Q)$. That is to say, π is a bad prefix of φ. □

Consider an agent strategy σ_{ag} and an environment strategy $\sigma_{\mathtt{env}}$ such that $\sigma_{\mathtt{env}} \rhd \varphi^e$, and let $\pi = \mathsf{play}(\sigma_{\mathsf{ag}}, \sigma_{\mathtt{env}})$. Suppose there exists $k \geq 0$, such that $i > k$, $Y_i = \mathsf{stop}$, $\pi^k \models \varphi^a_{\mathsf{task}}$ and no prefix π^h, with, $h < k$, is a bad prefix of $\varphi^a_{\mathsf{safe}}$. Thus, as described in above, π is split in two

parts: prefix π^k with non-stop actions that satisfies $\varphi^a_{\mathsf{task}}$ while not violating $\varphi^a_{\mathsf{safe}}$ and the remaining suffix in which the action stop is executed forever after. Regarding the suffix, the agent wins on this play only if a bad prefix of $\varphi^a_{\mathsf{safe}}$ never occurs. The environment, in contrast, aims at obtaining a bad prefix of $\varphi^a_{\mathsf{safe}}$, while ultimately satisfying φ^e, that is $\pi \models \varphi^e \wedge \neg\varphi^a_{\mathsf{safe}}$.

Observe that the formula $\varphi^e \wedge \neg\varphi^a_{\mathsf{safe}}$ mentions variables both in $\mathcal{X} \cup \mathcal{Y}$, and so its synthesis is as usual a two-player game between the environment and the agent, controlling $\mathcal{X}$ and $\mathcal{Y}$, respectively. However, in this phase, the agent is obliged to play the action stop, thus having only one assignment over $\mathcal{Y}$ at every iteration. This means that only one strategy is available to the agent, and the two-player game problem reduces to model-checking the subgame where only such strategy is available to the agent. Clearly, this has a significant impact on the complexity of the problem, as we need to solve model-checking instead of synthesis. As a matter of fact, although dealing with Büchi automata, we are reducing to their non-emptiness, which does not require their determinization and sidesteps the consequent difficulties.

Consequently, we first restrict both automata $\mathcal{N}^e = (\Sigma, Q^e, q^e_0, \delta^e, \text{Büchi}(T))$ and $\mathcal{D}_{bad} = (\Sigma, Q_{bad}, q_{0bad}, \delta_{bad}, \text{Reach}(\{q_{sink}\}))$, namely, $\mathcal{N}^e{}_{|\mathsf{stop}} = (\Sigma, Q^e, q^e_0, \delta^e_{|\mathsf{stop}}, \text{Büchi}(T))$ and $\mathcal{D}_{bad|\mathsf{stop}} = (\Sigma, Q_{bad}, q_0, \delta_{bad|\mathsf{stop}}, \text{Reach}(\{q_{sink}\}))$, in a way that they reject whenever a non-stop action is performed by the agent. This is obtained by removing from them all transitions of the form (q, a, d) where $a \not\models \neg\mathsf{stop}$.

We then compute their product NA defined as $\mathcal{N}' = (\Sigma, Q', q'_0, \delta', \text{Büchi}(T'))$ where

- $q'_0 = (q^e_0, q_{0bad})$;
- $(d, \hat{d}) \in \delta'((q, \hat{q}), a)$ if $d \in \delta_e|_{\mathsf{stop}}(q, a)$ and $\hat{d} = \delta_{bad}|_{\mathsf{stop}}(\hat{q}, a)$;
- $(q, q_{sink}) \in T'$ if $q \in T$.

The following lemma holds.

Lemma 2. *Consider* NA $\mathcal{N} = (\Sigma, Q, q_0, \delta, \text{Büchi}(T))$, DA $\mathcal{D}_{bad} = (\Sigma, Q_{bad}, q_{0bad}, \delta_{bad}, \text{Reach}(\{q_{sink}\}))$, *and the product* NA $\mathcal{N}'$ *constructed as above. Then, it holds that* $\mathcal{L}(\mathcal{N}') \subseteq \mathcal{L}(\mathcal{N}) \cup \mathcal{L}(\mathcal{D}_{bad})$.

Proof. Consider a $\pi \in \mathcal{L}(\mathcal{N}')$ and let $r' = (r, r_{bad}) = (q^e_0, q_{0bad}), (q^e_1, q_{1bad}), \ldots$ be an accepting run in $\mathcal{N}'$, with $r = q^e_0, q^e_1, \ldots$, and $r_{bad} = q_{0bad}, q_{1bad}, \ldots$ be the first and second projections of r', respectively. Note that r and r_{bad} are runs of π over $\mathcal{N}$ and $\mathcal{D}_{bad}$, respectively.

Therefore, from r' being accepting in $\mathcal{N}'$, it holds that $\inf(r) \cap T \neq \emptyset$ and so that $\pi \in \mathcal{L}(\mathcal{N})$. In addition, it holds that q_{sink} occurs at least once in r_{bad}. From the definition of $\mathcal{D}_{bad}$ it follows that q_{sink} holds indefinitely over r_{bad} and therefore it is accepting in $\mathcal{D}_{bad}$, from which we conclude that $\pi \in \mathcal{L}(\mathcal{D}_{bad})$. □

Clearly, the automaton $\mathcal{N}'$ accepts all and only those plays π that satisfy $\varphi^e \wedge \neg\varphi^a_{\mathsf{safe}}$ while the agent only plays stop. By employing the nonemptiness checking over $\mathcal{N}'$ we then compute the set $Q_{acc} \subseteq Q'$ of accepting states. We denote by $N_{\mathsf{stop}} = \{\hat{q} \in Q_{bad} \mid \exists q, (q, \hat{q}) \in Q_{acc}\}$ the set of states in $\mathcal{D}_{bad}$ that are part of some accepting state in $\mathcal{N}'$.

The set of "Non-stop" states essentially tells that if a play π enters phase II through them, then π is an environment winning play, regardless of the agent accomplishing the $\varphi^a_{\mathsf{task}}$ in phase I. Therefore, it is important for the agent to avoid "Non-stop" states when accomplishing $\varphi^a_{\mathsf{task}}$ while not violating $\varphi^a_{\mathsf{safe}}$, which can be considered as a reachability-safety condition.

Stage (*iii*): Compute the DA for the agent goal. We start with constructing a DA that only encodes accomplishing $\varphi^a_{\mathsf{task}}$ while not violating $\varphi^a_{\mathsf{safe}}$ until then, which intuitively corresponds to the reachability-safety condition defined in Section 2. On the one hand, the language of $\varphi^a_{\mathsf{task}}$ can be recognized by a DA $\mathcal{D}^a_t = (\Sigma, Q^a_t, q_{0^a_t}, \delta^a_t, \text{Reach}(R))$ with a reachability condition that accepts all and only the executions with a finite prefix satisfying $\varphi^a_{\mathsf{task}}$. On the other hand, the language of $\varphi^a_{\mathsf{safe}}$ can be recognized by a DA $\mathcal{D}^a_s = (\Sigma, Q^a_s \cup \{q_{sink}\}, q_{0^a_s}, \delta^a_s, \text{Safe}(Q^a_s))$ with a safety condition. Therefore, we can take the bounded intersection of these automata, thus obtaining $\mathcal{D}^a_{t,s} = (\Sigma, Q^a_{t,s}, q_{0^a_{t,s}}, \delta^a_{t,s}, \text{Reach–Safe}(R^a_{t,s}, S^a_{t,s}))$ with a reachability-safety condition.

Then, the language of such intersection is given by those infinite plays π for which there exists $k \geq 0$ such that $\pi^k \models \varphi^a_{\mathsf{task}}$, and all prefixes of π^k are not bad prefixes of $\varphi^a_{\mathsf{safe}}$.

Next, we show that such automaton can be equivalently turned into an DA with a pure reachability condition.

Reachability-Safety to Reachability. Consider a DA $\mathcal{D} = (\Sigma, Q, q_0, \delta, \text{Reach–Safe}(R, S))$ with a reachability-safety condition. We describe a reduction to a $\mathcal{D}' = (\Sigma, Q, q_0, \delta', \text{Reach}(T))$ with a pure reachability condition such that $\mathcal{L}(\mathcal{D}') = \mathcal{L}(\mathcal{D})$. The new automaton has the same set of states but a different transition relation, which is defined as follows:

$$\delta'(q, a) = \begin{cases} \delta(q, a) & \text{if } q \in S \\ q & \text{if } q \notin S. \end{cases}$$

Intuitively, the only change we make is to turn all non-safe states (states not in S) into sink states. We then define the reachability condition as $T = R \cap S$. Intuitively, we want to reach a goal state (a state in R) that is also safe (i.e., it is in S). The two automata are indeed equivalent.

Lemma 3. *Let $\mathcal{D}$ and $\mathcal{D}'$ be as above, then $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{D}')$.*

Proof. Consider an infinite trace $\pi \in \Sigma^\omega$ and the corresponding runs r and r' in $\mathcal{D}$ and $\mathcal{D}'$, respectively. Two cases are possible. If r is fully contained in S, then by the definition of $\mathcal{D}$ and $\mathcal{D}'$, it holds that $r = r'$ and so clearly π is accepted by $\mathcal{D}$ if and only if it is accepted by $\mathcal{D}$. The other case is that r is not fully contained in S and let $k+1$ be the minimum index for which $r_{k+1} \notin S$. Note that π then is accepted by $\mathcal{D}$ iff there is $h \leq k$ such that $r_h \in T$. Moreover, again by the definition of $\mathcal{D}'$, it holds that the prefix r^k up to k of r equals the prefix r'^k up to k. In particular, if a state $r_h \in R \cap S$ is reached in r^k, then the same state is reached in $r'^k \in R \cap S = T$, and therefore iff π is accepted by $\mathcal{D}'$. □

By applying Lemma 3 to $\mathcal{D}^a_{t,s}$, we can turn it into a DA $\mathcal{D}_{\mathsf{ag}} = (\Sigma, Q^a_{t,s}, q_{0^a_{t,s}}, \delta_{\mathsf{ag}}, \mathrm{Reach}(T_{\mathsf{ag}}))$ with a simple reachability condition that accepts those traces π for which there exists a prefix π^k that satisfies $\varphi^a_{\mathsf{task}}$ while not violating $\varphi^a_{\mathsf{safe}}$ up to k. In order to keep $\varphi^a_{\mathsf{safe}}$ hold even after accomplishing $\varphi^a_{\mathsf{task}}$, the agent needs to make sure that the second phase of the execution, i.e., the one starting with the action stop executed forever after. In other words, following the discussion earlier, the agent should avoid landing in those accepting states in $\mathcal{D}_{\mathsf{ag}}$ that do not ensure $\varphi^a_{\mathsf{safe}}$ when executing the action stop forever. As noted above, this piece of information is carried in the set N_{stop} precomputed as the emptiness of $\mathcal{N}'$.

We define $\mathcal{D}'_{\mathsf{ag}} = (\Sigma, Q^a_{t,s}, q_{0^a_{t,s}}, \delta'_{\mathsf{ag}}, \mathrm{Reach}(T'_{\mathsf{ag}}))$, where $T'_{\mathsf{ag}} = \{(q, \hat{q}) \mid (q, \hat{q}) \in T_{\mathsf{ag}} \text{ and } \hat{q} \notin N_{\mathsf{stop}}\}$.

Lemma 4. *Let $\mathcal{D}'_{\mathsf{ag}} = (\Sigma, Q^a_{t,s}, q_{0^a_{t,s}}, \delta_{\mathsf{ag}}, \mathrm{Reach}(T'_{\mathsf{ag}}))$ be as above. Then $\pi \in \mathcal{L}(\mathcal{D}'_{\mathsf{ag}})$ iff there exists k such that $\pi_k \models \varphi^a_{\mathsf{task}}$ and for all $f \in (2^\Sigma)^\omega$, where $f[i] \models \mathsf{stop}, \forall i \geq 0$, $\pi_k \cdot f \models (\varphi^e \rightarrow \varphi^a_{\mathsf{safe}})$.*

Proof. The proof proceeds by double implication. First assume that $\pi \in \mathcal{L}(\mathcal{D}'_{\mathsf{ag}})$ and let r be its accepting run. From the acceptance condition of the automaton, this holds if and only if there exists $k \in \mathbb{N}$ such that $r_k = (q, \hat{q}) \in T'_{\mathsf{ag}}$, which in turns holds iff $(q, \hat{q}) \in T_{\mathsf{ag}}$ and $\hat{q} \notin N_{\mathsf{stop}}$. Now, following the definition of $\mathcal{D}_{\mathsf{ag}}$ this holds if and only if q is an accepting state for $\mathcal{D}^a_t$ and $\mathcal{D}^a_s$, respectively. This means, on the one hand, that $\pi^k \models \varphi^a_{\mathsf{task}}$ and, on the other hand, that π^k does not contain bad prefixes for $\varphi^a_{\mathsf{safe}}$. Moreover, from the fact that $\hat{q} \notin N_{\mathsf{stop}}$ it follows that any sequence $f \in \Sigma^\omega$ with $f[i] \models \mathsf{stop}$ is such that $f \in \mathcal{L}(\mathcal{N}')$ and so that $f \models (\varphi^e \rightarrow \varphi^a_{\mathsf{safe}})$. By combining this with the fact that π^k does not contain bad prefixes for $\varphi^a_{\mathsf{safe}}$, it easily follows that $\pi_k \cdot f \models (\varphi^e \rightarrow \varphi^a_{\mathsf{safe}})$. This concludes the proof, as all the steps presented hold in double implication. □

Stage (*iv*): Solve the synthesis for $Goal$ under φ^e. Following the stages detailed in Section 3, we now construct the product automaton $\mathcal{D}$ between $\mathcal{D}'^e_s$ and $\mathcal{D}'_{\mathsf{ag}}$. Solving the reachability game for the agent over $\mathcal{D}$ returns us an agent winning strategy if one exists.

Theorem 8. *The problem of realizing $\mathcal{P} = \langle \mathcal{X}, \mathcal{Y}, \varphi^e, Goal \rangle$, with $Goal = \langle \varphi^a_{\mathsf{task}}, \varphi^a_{\mathsf{safe}} \rangle$ can be reduced to solving the reachability game in $\mathcal{D}'^e_s \cap \mathcal{D}'_{\mathsf{ag}}$.*

Proof. First, note that from Theorem 4, the problem of realizing $\mathcal{P}$ is equivalent to the one of realizing $\hat{\mathcal{P}} = \langle \mathcal{X}, \mathcal{Y}, \varphi^e_{\mathsf{safe}}, Goal \rangle$, with $\varphi^e = \varphi^e_{\mathsf{safe}} \wedge \varphi^e_{\mathsf{live}}$. Therefore, we can focus on the latter instead.

Now, consider a strategy $\hat{\sigma_{\mathsf{ag}}}$ that is winning in $\mathcal{D}'^e_s \cap \mathcal{D}'_{\mathsf{ag}}$. Moreover, let σ_{env} be an environment strategy such that $\sigma_{\mathsf{env}} \rhd \varphi^e_{\mathsf{safe}}$ and $\pi = \mathsf{play}(\hat{\sigma_{\mathsf{ag}}}, \sigma_{\mathsf{env}})$. Being σ_{env} satisfying the environment specification and $\hat{\sigma_{\mathsf{ag}}}$ winning, it holds that $\pi \in \mathcal{L}(\mathcal{D}'^e_s \cap \mathcal{D}'_{\mathsf{ag}})$ and so its run $r(\pi)$ is such that $r(\pi)^k \in T'_{\mathsf{ag}}$ for some $k \in \mathbb{N}$. Observe that, from Lemma 4, it holds that $\pi^k \models \varphi^a_{\mathsf{task}}$ and $\pi^k \cdot f \models (\varphi^e \rightarrow \varphi^a_{\mathsf{task}})$. Therefore, we need to adjust $\hat{\sigma_{\mathsf{ag}}}$ in a way that it produces plays of that form. This can be done by considering the strategy σ_{ag} defined as

$$\sigma_{\mathsf{ag}}(x) = \begin{cases} \mathsf{stop}, & \text{if } r(x \cup \hat{\sigma_{\mathsf{ag}}}(x)) \text{ is accepting in } \mathcal{D}'_{\mathsf{ag}} \\ \hat{\sigma_{\mathsf{ag}}}(x), & \text{otherwise.} \end{cases}$$

Intuitively, σ_{ag} imitates $\hat{\sigma_{\mathsf{ag}}}$ until it wins the reachability game in $\mathcal{D}'_{\mathsf{ag}}$ and then switches to executing stop forever. Then, we have that σ_{ag} realizes $\hat{\mathcal{P}}$ and so does with $\mathcal{P}$. □

Referring to the agent safety goal, one might question whether it is natural to always interpret it on infinite traces, due to the fact that the task specified in LTL$_f$ is achieved in a finite number of steps. In fact, our solution is also able to manage the case where only ensuring the safety goal until task is accomplished. In that case, there is no "Non-stop" states for the agent, we can just skip Stage (*ii*) and proceed with Stage (*iii*), considering the set of "Non-stop" states as empty.

Computational properties. We study the computational properties of the synthesis technique presented in this section. We first consider the overall complexity, as stated by the following theorem.

Theorem 9. *The above algorithm solves synthesis with mandatory stop actions problem $\mathcal{P} = (\mathcal{X}, \mathcal{Y}, \varphi^e, Goal)$ with safety goals in* 2EXPTIME *(the problem is indeed* 2EXPTIME*-complete).*

We now analyze the operations required in our synthesis technique in detail. As stated in Section 4, reducing φ^e to only considering the safety part $\varphi^e_{\mathsf{safe}}$ allows us to perform standard subset construction and minimization instead of Büchi automata determinization to obtain DA $\mathcal{D}^e_s$. Moreover, reducing non-violation of $\varphi^a_{\mathsf{safe}}$ after $\varphi^a_{\mathsf{task}}$ to non-emptiness checking on nondeterministic Büchi automata brings us an exponential advantage, which allows us, again sidestep the Büchi automata determinization. Later in Stage (*iii*), being a safety formula, the DA $\mathcal{D}^a_s$ can be obtained by using the subset construction and minimization to maximally shrink the state space. At the end, we only need to solve an adversarial reachability game instead of a parity game. We thus believe that our technique is intrinsically simpler and easier to implement than standard LTL synthesis.

7 Conclusion

In this paper we have studied mandatory stop actions in agent strategies. We have seen this has a deep impact on the efficiency of solving the synthesis under LTL environment specifications, since the liveness part of the LTL environment specifications becomes irrelevant, and hence we can adopt a synthesis technique based on finite-state automata manipulation as in LTL$_f$ synthesis. We have also studied adding safety goals for the agent. Safety goals must hold forever. Hence, before stopping, the agent must ensure her safety goal will be maintained anyway. While we have studied synthesis in this case under the requirement of mandatory stop actions, it is of interest to study these kinds of goals independently of such a requirement. Possibly the insight gained here can be of help also when lifting such a requirement.

Acknowledgements

Work supported by the ERC Advanced Grant WhiteMech (No. 834228) and by the EU ICT-48 2020 project TAILOR (No. 952215).

References

Alpern, B., and Schneider, F. B. 1987. Recognizing safety and liveness. *Distributed Comput.* 2(3):117–126.

Aminof, B.; De Giacomo, G.; Murano, A.; and Rubin, S. 2018. Planning and synthesis under assumptions. *CoRR* abs/1807.06777.

Aminof, B.; De Giacomo, G.; Murano, A.; and Rubin, S. 2019. Planning under LTL environment specifications. In *ICAPS*, 31–39.

Baier, J. A., and McIlraith, S. A. 2006. Planning with temporally extended goals using heuristic search. In *ICAPS*, 342–345. AAAI.

Camacho, A.; Triantafillou, E.; Muise, C. J.; Baier, J. A.; and McIlraith, S. A. 2017. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In Singh, S. P., and Markovitch, S., eds., *AAAI*, 3716–3724.

Camacho, A.; Bienvenu, M.; and McIlraith, S. A. 2018. Finite LTL synthesis with environment assumptions and quality measures. In *KR*, 454–463.

Camacho, A.; Bienvenu, M.; and McIlraith, S. A. 2019. Towards a unified view of AI planning and reactive synthesis. In *ICAPS*, 58–67. AAAI Press.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, strong, and strong cyclic planning via symbolic model checking. 1–2(147).

De Giacomo, G., and Rubin, S. 2018. Automata-theoretic foundations of FOND planning for LTL_f and LDL_f goals. In *IJCAI*, 4729–4735.

De Giacomo, G., and Vardi, M. Y. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *IJCAI*.

De Giacomo, G., and Vardi, M. Y. 2015. Synthesis for LTL and LDL on finite traces. In *IJCAI*.

De Giacomo, G.; Di Stasio, A.; Vardi, M. Y.; and Zhu, S. 2020a. Two-stage technique for ltlf synthesis under LTL assumptions. In *KR*, 304–314.

De Giacomo, G.; Di Stasio, A.; Vardi, M. Y.; and Zhu, S. 2020b. Two-stage technique for ltlf synthesis under LTL assumptions. In *KR 2020*, 304–314.

Finkbeiner, B. 2016. Synthesis of Reactive Systems. *Dependable Software Systems Eng.* 45:72–98.

He, K.; Wells, A. M.; Kavraki, L. E.; and Vardi, M. Y. 2019. Efficient symbolic reactive synthesis for finite-horizon tasks. In *ICRA*, 8993–8999. IEEE.

Kupferman, O., and Vardi, M. Y. 2001. Model checking of safety properties. *Formal Methods Syst. Des.* 19(3):291–314.

Pnueli, A., and Rosner, R. 1989. On the synthesis of a reactive module. In *POPL*.

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

Rabin, M. O., and Scott, D. S. 1959. Finite automata and their decision problems. *IBM J. Res. Dev.* 3(2):114–125.

Rodriguez, I. D.; Bonet, B.; Sardiña, S.; and Geffner, H. 2021. Flexible FOND planning with explicit fairness assumptions. *CoRR* abs/2103.08391.

Sardiña, S., and D'Ippolito, N. 2015. Towards fully observable non-deterministic planning as assumption-based automatic synthesis. In *IJCAI*.

Tabakov, D., and Vardi, M. Y. 2005. Experimental evaluation of classical automata constructions. In *LPAR*, volume 3835 of *Lecture Notes in Computer Science*, 396–411. Springer.

Tabakov, D.; Rozier, K. Y.; and Vardi, M. Y. 2012. Optimized temporal monitors for systemc. *Formal Methods Syst. Des.* 41(3):236–268.

Zhu, S.; Tabajara, L. M.; Li, J.; Pu, G.; and Vardi, M. Y. 2017. Symbolic LTL_f synthesis. In *IJCAI*, 1362–1369.

Zhu, S.; De Giacomo, G.; Pu, G.; and Vardi, M. 2020. LTL_f synthesis with fairness and stability assumptions. In *AAAI*.