LTL$_f$ Synthesis under Unreliable Input

Christian Hagemeier[1], Giuseppe De Giacomo[1], Moshe Y. Vardi[2]

[1]University of Oxford, Oxford, UK
[2]Rice University, Houston, Texas, USA
christian@hagemeier.ch, giuseppe.degiacomo@cs.ox.ac.uk, vardi@cs.rice.edu

Abstract

We study the problem of realizing strategies for an LTL$_f$ goal specification while ensuring that at least an LTL$_f$ backup specification is satisfied in case of unreliability of certain input variables. We formally define the problem and characterize its worst-case complexity as 2EXPTIME-complete, like standard LTL$_f$ synthesis. Then we devise three different solution techniques: one based on direct automata manipulation, which is 2EXPTIME, one disregarding unreliable input variables by adopting a belief construction, which is 3EXPTIME, and one leveraging second-order quantified LTL$_f$ (QLTL$_f$), which is 2EXPTIME and allows for a direct encoding into monadic second-order logic, which in turn is worst-case nonelementary. We prove their correctness and evaluate them against each other empirically. Interestingly, theoretical worst-case bounds do not translate into observed performance; the MSO technique performs best, followed by belief construction and direct automata manipulation. As a byproduct of our study, we provide a general synthesis procedure for arbitrary QLTL$_f$ specifications.

Code — https://github.com/whitemech/ltlf-synth-unrel-input-aaai2025

Extended version — arxiv./example

1 Introduction

One of the key challenges in Artificial Intelligence is to equip intelligent agents with the autonomous capability to deliberate and execute complex courses of action to accomplish desired tasks, see, e.g. the work on reasoning about action (Reiter 2001) and on planning (Ghallab, Nau, and Traverso 2016).

The problem we study is related to *reactive synthesis* in Formal Methods (Pnueli and Rosner 1989; Finkbeiner 2016; Ehlers et al. 2017), which shares deep similarities with planning in fully observable nondeterministic domains (FOND, strong plans (Cimatti, Roveri, and Traverso 1998; Geffner and Bonet 2013)). A reactive synthesis setting is characterized by (boolean) variables, partitioned into input and output variables, changing over time. We have two entities acting in this setting: the environment that controls the input variables (or fluents, in planning terminology), and the agent who controls the output variables (or actions, in planning terminology). Given a specification, the agent has to find a strategy (or plan in planning terms) to choose its outputs to fulfil the specification by suitably reacting to the inputs chosen (possibly adversarially) by the environment.

In Formal Methods, the most common specification formalism is Linear Temporal Logic (LTL) (Pnueli 1977). In AI, a finite trace variant of LTL (LTL$_f$) is popular (Gabbay et al. 1980; Baier and McIlraith 2006; De Giacomo and Vardi 2013; De Giacomo and Rubin 2018). The interest in finite traces is due to the observation that, typically, intelligent agents are not dedicated to a single task (specification) all their lives but are supposed to accomplish one task after another. In this paper, we will focus on LTL$_f$ as a specification language.

Note that in reactive synthesis, at any point, the agent observes the current input and, based on its value (along with previous input values), decides how to react by choosing an appropriate output towards satisfying the specification. Interestingly, machine-learning techniques are bringing about notable advancements in sensing technologies (i.e., generation of input), as showcased by the success of vision and autonomous driving techniques. However, machine-learning techniques are typically data-oriented and hence have a statistical nature that may generate occasional unreliability of the input produced. Consider a surgical robot that uses machine-learning models to interpret sensory data and guide its actions with precision during complex medical procedures. Due to the inherent imprecision of the input models, it may misinterpret a critical piece of data regarding the patient's anatomy (for instance, the robot might incorrectly identify a blood vessel as a different tissue type due to subtle variances in the imaging data that were not accounted for in the training phase of the AI model). This misinterpretation can lead the robot to make an inaccurate incision, potentially causing unintended harm to the patient.

In this paper, we study how to address such potential unreliability in the values of input variables in reactive synthesis. One way to address this unreliability is to disregard the unreliable input completely and not consider it in choosing the next output (Bloem et al. 2019). This is related to planning/synthesis under partial observability (Rintanen 2004; De Giacomo and Vardi 2016; Kupferman and Vardi 2000; Ehlers and Topcu 2015). However, this might be too radical and could drastically reduce the agent's ability to operate, considering that the unreliability we are considering is only occasional. Our objective instead is to ensure that the system maintains functionality and adheres to critical specifications, despite uncertain and unreliable inputs. If the uncertainty is quantifiable, we could rely on probabilities turning to MDPs (Baier and Katoen 2008; Geffner and Bonet 2013) or Stochastic Games (Kwiatkowska 2016). Yet, as stated in a report by The White House (2024), "software does not necessarily conform neatly to probabilistic distributions, making it difficult to apply statistical models or predictions commonly used in other scientific disciplines". Here we aim at exploring a novel synthesis method to manage the potential unreliability of input variables obtained without relying on probabilities.

The crux of our approach is not to give up on using input variables that might be unreliable but to complement them with the guarantee that even when they behave badly, some safeguard conditions are maintained. Specifically, we consider two models simultaneously, a *brave model* where all input variables are considered reliable (as usual in synthesis), and a *cautious one* where unreliable input is projected out and discarded (De Giacomo and Vardi 2016). Using these two models, we devise a strategy that simultaneously fulfils the task objectives completely if the input variables behave correctly and maintains certain fallback conditions even if the unreliable input variables behave wrongly.

Our contributions are the following:

- A formalization of LTL$_f$ synthesis under unreliable input, which follows the framework described above.
- The computational characterization of the problem in terms of worst-case complexity as 2EXPTIME-complete, as standard LTL$_f$ synthesis.
- Three provably correct synthesis techniques, one based on a direct method, one based on a belief construction and one based on solving synthesis for QLTL$_f$ formulas, where QLTL$_f$, second-order quantified LTL$_f$, is the finite trace variant of QLTL (Sistla, Vardi, and Wolper 1985; Calvanese, De Giacomo, and Vardi 2002).
- The three techniques have different complexities: the direct one is 2EXPTIME, the one based on a belief construction is 3EXPTIME, and one based QLTL$_f$ is 2EXPTIME, but also admits a direct encoding in monadic second-order logic over finite sequences (MSO), which is non-elementary.
- An experimental assessment of the three synthesis techniques. Interestingly, the theoretical worst-case bounds do not translate into observed performance; the MSO technique performs best, followed by belief construction and direct automata manipulation.[1]

As a side result, we present a synthesis technique for arbitrary QLTL$_f$ specifications. For space reasons proofs are omitted; however, can be found in the extended version.

2 Preliminaries

Definition 1. The language of LTL$_f$ is defined by

$$\varphi \coloneqq A \mid \neg\varphi \mid \varphi \wedge \psi \mid \circ\varphi \mid \phi \,\mathsf{U}\, \psi$$

with $A \in \mathcal{P}$, where $\mathcal{P}$ is a set of propositional variables.

Other modalities, like Weak Next ($\bullet\varphi \coloneqq \neg\circ\neg\varphi$), Eventually ($\Diamond$), and Always ($\Box$), can be defined.

Definition 2 (LTL$_f$ semantics)**.** Given a trace $t \in (2^{\mathcal{P}})^+$, the satisfaction relation $t, i \vDash \varphi$ is defined inductively for $1 \leq i \leq \mathsf{length}(t) = \mathit{last}$:

- $t, i \vDash A$ iff $A \in t(i)$
- $t, i \vDash \neg\varphi$ iff $t, i \nvDash \varphi$
- $t, i \vDash \varphi \wedge \psi$ iff $t, i \vDash \varphi$ and $t, i \vDash \psi$
- $t, i \vDash \circ\varphi$ iff $i < \mathit{last}$ and $t, i+1 \vDash \varphi$
- $t, i \vDash \varphi \,\mathsf{U}\, \psi$ iff for some j with $i \leq j \leq \mathit{last}$ we have $t, j \vDash \psi$ and for all k with $i \leq k < j$ we have $t, k \vDash \varphi$.

We say that a trace satisfies an LTL$_f$-formula, written as $t \vDash \varphi$, iff $t, 1 \vDash \varphi$.

Synthesis under full observability. Classical LTL$_f$ **synthesis** refers to a game between an agent and the environment. Both control a subset of the variables of an LTL$_f$-formula φ, which the agent tries to satisfy.

An agent's strategy is a function $\sigma : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$. The strategy σ realizes a formula φ if, for any infinite sequence $\lambda = (X_0, X_1, \dots) \in (2^{\mathcal{X}})^\omega$, there exists a $k \in \mathbb{N}$ such that the finite trace $t = ((X_0, Y_0), \dots, (X_k, Y_k))$ satisfies φ (i.e. $t \vDash \varphi$), where $Y_i = \sigma(X_0, \dots, X_{i-1})$.

DFA Games. The standard technique for solving LTL$_f$ synthesis works by reducing it to solving a reachability game on a DFA (De Giacomo and Vardi 2015). A DFA game is also played between two players, the agent and the environment. They have corresponding sets of boolean variables $\mathcal{X}, \mathcal{Y}$. The specification of a game is given by a DFA $\mathcal{G} = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s_0, \delta, F)$ where $2^{\mathcal{X}\cup\mathcal{Y}}$ is the alphabet, S the set of states, the initial state s_0, total transition function $\delta : S \times 2^{\mathcal{X}\cup\mathcal{Y}} \to S$ and the set of final states $F \subseteq S$. A play on a DFA game is a sequence $\rho = ((s_{i,0}, X_0 \cup Y_0), (s_{i,1}, X_1 \cup Y_1), \dots) \in (S \times 2^{\mathcal{X}\cup\mathcal{Y}})^+$ with $s_{i,j+1} = \delta(s_{i,j}, X_i \cup Y_i)$. Such a play is winning if it ends in a final state. We say that a player has a winning strategy in a DFA game if they can choose the variables in a way that guarantees to end up in a final state regardless of how the other player responds.

[1]Observe that, following the state-of-the-art, the implementation of all the techniques makes use of the tool MONA (Henriksen et al. 1995), which is based on first-order/monadic second-order logic over finite sequences, both non-elementary in the worst case.

3 LTL_f Synthesis under Unreliable Input

Before formalizing the problem, we introduce some additional notation. The projection function $\mathsf{proj}_{\mathcal{V}}(t)$ removes all variables in $\mathcal{V}$ from a trace t over propositional variables $\mathcal{P}$. The expansion function $\mathsf{exp}_{\mathcal{V}}(\hat{t})$ takes a trace $\hat{t}$ over variables $\hat{\mathcal{P}}$ (with $\mathcal{V} \cap \hat{\mathcal{P}} = \varnothing$) and returns all traces t by setting variables in $\mathcal{V}$ in every possible way at all instants of $\hat{t}$. For traces t and t', $t \sim_{-\mathcal{V}} t'$ means $t \in \mathsf{exp}_{\mathcal{V}}(\mathsf{proj}_{\mathcal{V}}(t'))$. Slightly abusing notation, we use the same syntax to denote two subsets of $\mathcal{P}$ being equal up to elements of $\mathcal{V}$. We model uncertainty about the environment by assuming the agent cannot rely on the readings of certain environment variables; thus, we require that after any changes in these readings still satisfy a backup condition; this leads to the following formalization:

Definition 3 (LTL_f synthesis under unreliable input)**.** Given LTL_f-formulas φ_m, φ_b over variables $\mathcal{X} \uplus \mathcal{Y}$, called respectively the main and backup specification, and a partitioning $\mathcal{X} = \mathcal{X}_{rel} \uplus \mathcal{X}_{unr}$ of the input variables into reliable and unreliable ones respectively, solving LTL_f *synthesis under unreliable input* amounts to finding a strategy $\sigma : (2^{\mathcal{X}})^* \to 2^{\mathcal{Y}}$ such that for any infinite sequence of variables $\lambda = (X_0, X_1, \ldots) \in (2^{\mathcal{X}})^\omega$ there is an index $k \in \mathbb{N}$ such that

1. The finite trace $t = ((X_0, Y_0), \ldots, (X_k, Y_k))$ with $Y_i = \sigma(X_0, \ldots, X_{i-1})$ satisfies φ_m, i.e., $t \vDash \varphi_m$,
2. and every t' with $t' \sim_{-\mathcal{X}_{unr}} t$ satisfies φ_b, i.e., $t' \vDash \varphi_b$.

Our problem extends standard LTL_f synthesis by using $\top$ as a backup formula and $\mathcal{X}_{unr} = \varnothing$. Since LTL_f synthesis is 2EXPTIME-complete (De Giacomo and Vardi 2015), our problem is 2EXPTIME-hard. We later show a matching upper bound (see Theorem 10). The problem is also related, however distinct, to LTL_f synthesis under partial observability De Giacomo and Vardi (2016). If the main specification goal is trivial (i.e. setting $\varphi_m := \top$), our problem degenerates into LTL_f synthesis under partial observability.

Let us illustrate the problem with some examples, which are designed such that by suitably choosing parameters, realizability can be controlled.

Example 4 (Sheep)*.* A farmer wants to move her n sheep from the left to the right of a river, always moving two sheep simultaneously (it is inspired by well-known puzzles (Lampis and Mitsou 2007)). However, some pairs of sheep may not like each other. If she can correctly observe the animosities, she wants to move all sheep; however, when her intuition about which sheep are compatible is unreliable, only a subset (her favourite sheep) must be moved.

Here, the agent has n output variables, of which she can always activate two to request the environment to move the corresponding sheep. Additionally, if for any potentially forbidden pair of sheep i, j, an unreliable input variable $\mathsf{disallow_{i,j}}$ is activated, it indicates that i and j cannot be moved together. This leads to synthesis for the formula $\varphi_{ag} \wedge (\varphi_{env} \supset \varphi_{goal})$, with different φ_{goal} for main and backup. Here, the unreliable inputs are variables $\mathsf{disallow_{i,j}}$ for each pair of sheep that may not like each other.

Example 5 (Trap)*.* A robot is searching for a path from a vertex v of a graph with n vertices to a set of secure vertices. For simplicity of modelling, there are at most 2 outgoing edges from any vertex. However, the environment may control the state of a set of traps, allowing it to divert one of two edges to a different endpoint. If the robot can correctly sense the state of the traps, it should move to a secure vertex; however, if it cannot correctly observe the traps, the same strategy should guarantee that the (larger) backup safety region is reached.

We can model this with t unreliable input variables t_i that indicate the state of each trap. Additionally, the environment controls $\lceil \log_2(n) \rceil$ variables for denoting the current position. The main and backup specifications then both have the form $\varphi_{env} \to \bigvee_{v \in R} \lozenge(pos(v))$, where $\bigvee_{v \in R} \lozenge(pos(v))$ ensures that the agent ends up in the corresponding region.

Example 6 (Hiker)*.* We model a hiker on a trail of length n in the Alps who wants to eat berries, avoiding poisonous ones that could make her sick. Due to the similarity between poisonous and non-poisonous berries, she might consume poisonous ones if her senses are unreliable. Fortunately, natural medical herbs along the trail can cure sickness. Her main goal is to eat all non-poisonous berries, and even when her senses are unreliable, she wants to ensure that she is not sick by the end of the trail.

We can model this as an instance of synthesis under unreliable input by having $\mathcal{X} = \{berry, poison, sick, herb, eot\}$ and $\mathcal{Y} = \{eat, takeMedicine, collectMedicine\}$. We then have an environment specification φ_e and an agent main and backup goal. The main specification then is simply $\varphi_e \supset \Box(\mathsf{berry} \wedge \neg\mathsf{poison} \supset \mathsf{eat}) \wedge \lozenge(\mathsf{eot})$. And the backup specification is $\varphi_e \supset \lozenge(\mathsf{eot} \wedge \neg\mathsf{sick})$. The only unreliable input variable here is the variable that indicates whether the berry before the hiker (if existent) is poisonous, i.e. $\mathcal{X}_{unr} = \{\mathsf{poison}\}$. The environment constraints specify how each of the variables changes; for example, for sickness, we have the following environment constraint (in the style of successor state axioms (Reiter 1991)):

$$\circ sick \equiv \circ\top \wedge (\circ eat \wedge berry \wedge poison) \\ \vee (sick \wedge \neg(inbag \wedge takeMedication))$$

In general, the hiker does not have a strategy to solve this under unreliable input, as when there are no herbs along the path, she cannot guarantee fulfilling the backup specification under unreliable input. However, if we force there to be herbal medicine at least at one point of her trail, then the following strategy realises both goals: Eat all berries, collect the medicine when it is on the trail and finally, use it after eating all berries. The backup specification in this setting does not influence realisability but changes which strategies successfully realise the goal, as this blocks simple strategies such as eating all berries that appear non-poisonous and disregarding the medicine.

4 Technique 1: Direct Automata

Recall that we can essentially view the problem as the agent having to satisfy two goals in the brave and cautious arena simultaneously. This suggests combining an arena for the main goal under full observability with an arena for the backup goal under partial observability. We first describe the construction before showing correctness. It has three main ingredients: an automaton recognizing the main formula, one for the backup formula under partial observability and the correct combination of these, the synchronous product.

Definition 7 (Synchronous Product of DFAs)**.** Given two DFAs $\mathcal{G}_i = (2^{\mathcal{X}} \times 2^{\mathcal{Y}}, S_i, s_{0,i}, \delta_i, F_i)$ (for $i \in \{1, 2\}$, and defined over the same alphabet), we define their synchronous product as $G_1 \otimes G_2 = (2^{\mathcal{X}} \times 2^{\mathcal{Y}}, S_1 \times S_2, (s_{0,1}, s_{0,2}), \delta', F')$, where $\delta'((s_1, s_2), \sigma) = (\delta_1(s_1, \sigma), \delta_2(s_2, \sigma))$ and $F' = F_1 \times F_2$.

Let us now describe the construction of the automaton for the backup formula under partial observability:

- First, we create an NFA for the complement of the formula, i.e. $A_{\neg\varphi}$.
- Next, we existentially abstract over the unreliable inputs $U_1, \ldots, U_n$, yielding an NFA $(A_{\neg\varphi})_{\exists U_1,\ldots,U_n}$, which we formally define in Definition 8.
- Lastly, we determinize the NFA using subset construction. We then complement the final DFA obtaining our final automaton $\overline{\mathcal{D}((\mathcal{A}_{\neg\varphi})_{\exists U_1,\ldots,U_n})}$.

Formally, we can define the existential abstraction:

Definition 8. Given an NFA $N = (2^{\mathcal{X}} \times 2^{\mathcal{Y}}, S, \delta, s_0, F)$, we define the existentially abstracted NFA $N_{\exists U_1,\ldots,U_n} = \big(2^{\mathcal{X}} \times 2^{\mathcal{Y}}, S, \delta', s_0, F\big)$ by setting

$$\delta'(s, (X, Y)) = \{s' \mid \exists X'. X \sim_{-U_1,\ldots,U_n} X' \wedge s' \in \delta(s, (X', Y))\}.$$

Theorem 9. *Solving synthesis for the synchronous product of the* LTL$_f$*-automaton for* φ_m*,* A_{φ_m}*, and the automaton* $\overline{\mathcal{D}((\mathcal{A}_{\neg\varphi})_{\exists U_1,\ldots,U_n})}$ *solves synthesis under unreliable input.*

Theorem 10. *The outlined technique has worst-case complexity of 2EXPTIME.*

Combined with the 2EXPTIME-hardness of LTL$_f$ synthesis, this establishes 2EXPTIME-completeness for synthesis under unreliable input:

Theorem 11. LTL$_f$ *synthesis under unreliable input is 2EXPTIME-complete.*

This algorithm can be efficiently implemented in a symbolic synthesis framework. Here, given symbolic automata, the synchronous product simply corresponds to merging the sets of bits representing states and conjunction with the BDD representation for the final states (i.e. De Giacomo, Parretti, and Zhu (2023)). The construction of the NFA can be efficiently implemented symbolically by translating the negated formula into Pure-Past-LTLf, and thus constructing a DFA for the reverse language of $\neg\varphi_b$ (as described in Zhu, Pu, and Vardi (2019)). Then the subset construction (for determinization), reversal and existential abstraction can be carried out efficiently on the BDDs. For instance, the subset construction step can be carried out symbolically by introducing one variable for each state of the NFA.

5 Technique 2: Belief-States

De Giacomo and Vardi (2016) also investigate a second technique for generating an automaton to solve the game under partial observability, namely the belief-state construction, that we can basically use as an alternative to constructing an automaton for the backup formula under partial observability, keeping the other steps identical.

Definition 12 (Belief State DFA Game)**.** Given a DFA game $\mathcal{A} = (2^{\mathcal{X}\cup\mathcal{Y}}, S, s_0, F)$ with input variables partitioned into $\mathcal{X} = \mathcal{X}_{rel} \uplus \mathcal{X}_{unr}$, we define the belief-state game $\mathcal{G}^{rel}_{\mathcal{A}} = (2^{\mathcal{X}\cup\mathcal{Y}}, \mathcal{B}, B_0, \partial, \mathcal{F})$ as follows: $\mathcal{B} = 2^S$ (the power set of states), $B_0 = \{s_0\}$ (the initial state lifted to the power set), $\partial : \mathcal{B} \times 2^{\mathcal{X}\cup\mathcal{Y}} \to \mathcal{B}$ defined by

$$\partial(B, (X \cup Y)) = \{s' \mid \exists s \in B \exists X'. X \sim_{-\mathcal{X}_{unr}} X' \wedge \delta(s, (X' \cup Y)) = s'\},$$

and $\mathcal{F} = 2^F$ (the final states of the game).

With this definition, we can show correctness and characterize the complexity.

Theorem 13. *Solving synthesis for the synchronous product of the* LTL$_f$*-automaton for* φ_m*,* A_{φ_m}*, and the belief-state automaton* $G^{rel}_{\mathcal{A}_{\varphi_b}}$ *solves synthesis under unreliable input.*

Theorem 14. *Solving synthesis with backup using the belief-state construction yields a 3EXPTIME algorithm.*

Proof. This follows from the fact that constructing the belief-state automaton takes 3EXPTIME (construction of the DFA from the formula takes 2EXPTIME, then the belief-state construction costs another exponential); generating the DFA for the main formula takes 2EXPTIME, the other steps are polynomial. ☐

The belief state construction can also be implemented symbolically efficiently, similar to the subset construction for the direct approach (Tabajara and Vardi 2020).

6 Quantified LTL$_f$ Synthesis

Our third technique builds on translating the problem into synhesis for QLTL$_f$, which is a variant of QLTL, that similarly adds second-order quantification over propositional variables to LTL$_f$. In this section, we present a general algorithm for QLTL$_f$ synthesis. We then cast synthesis under unreliable input as a special case of this problem and show that for such formulas, the general algorithm matches the 2EXPTIME bound. A QLTL$_f$ formula is given by the following grammar (X denotes a second-order variable):[2]

$$\varphi \coloneqq X \mid \neg\varphi \mid \varphi \wedge \psi \mid \circ\varphi \mid \varphi\,\mathcal{U}\,\psi \mid \exists X.\varphi$$

The satisfaction relation for QLTL$_f$ is defined as for LTL$_f$, with an additional clause for the second-order quantifier:

$$t, i \vDash \exists X.\varphi :\Leftrightarrow \exists t'.t \sim_{-X} t' \wedge t', i \vDash \varphi$$

Universal quantification in QLTL$_f$ is defined as $\forall X.\varphi \coloneqq \neg\exists X.\neg\varphi$; other modalities are defined as in LTL$_f$. More informally, the existential quantifier '$\exists x.\varphi$' in QLTL$_f$ states that there is at least a way to modify where in the trace a variable x holds, thereby making the formula φ true. A formula is in prenex normal form (PNF) if it is of the form $Q_1X_1.\,Q_2X_2.\,\ldots Q_nX_n.\,\varphi$ where φ is an LTL$_f$-formula and $Q_i \in \{\forall, \exists\}$. Any formula can be polytime-transformed into PNF, similar to QLTL (Piribauer et al. 2021). The alternation count of a formula in PNF is the number of indices i s.t. $Q_i \neq Q_{i+1}$.

Theorem 15. *Synthesis for a* QLTL$_f$*-formula* ψ *with k alternations can be solved in* $(k+2)$*-EXPTIME.*

Theorem 16. *A strategy* σ *realizes the instance of* LTL$_f$ *synthesis under unreliable input with* $\mathcal{X}_{unr} = \{U_1, \ldots, U_n\}$ *iff it realizes synthesis for the* QLTL$_f$ *formula* $\varphi_m \wedge \forall U_1.\ldots\forall U_n.\varphi_b$.

From counting the alternations in the formula, we can deduce that this technique has optimal worst-case complexity.

Theorem 17. *Solving* LTL$_f$ *synthesis under unreliable input by translating into* QLTL$_f$ *has complexity 2EXPTIME.*

7 Technique 3: MSO Encoding

Exploiting Theorem 16 we now propose a third solution technique for LTL$_f$ synthesis under unreliable input. Specifically, we start from the QLTL$_f$ specification $\varphi_m \wedge \forall U_1.\ldots\forall U_n.\varphi_b$ translate it into monadic second-order logic (MSO), and then use MONA to obtain the DFA corresponding to the original specification (for synthesis under unreliable input). Then we can solve the DFA game, just like in standard LTL$_f$ synthesis. In fact, this approach also works for synthesis of arbitrary QLTL$_f$ formulas.

Formulas of MSO are given by the following grammar (x, y denote first-order variables, and X denotes a second-order variable):

$$\varphi \coloneqq X(x) \mid x < y \mid (\varphi \wedge \psi) \mid \neg\varphi \mid \exists x.\varphi \mid \exists X.\varphi.$$

We then consider *monadic structures* as interpretations that correspond to traces. We use the notation $t, [x/i] \vDash \varphi$ to denote that this interpretation of second-order variables (details in Appendix E), with assigning i to the FO-variable x satisfies φ. We can then give a translation and show its correctness:

Definition 18. We define a translation mso from QLTL$_f$ to MSO by setting (we use standard abbreviations for $succ(x,y)$, $x \leq y$, and $x \leq last$):

$$\begin{array}{rcl}
\mathsf{mso}(X, x) & \coloneqq & X(x) \\
\mathsf{mso}(\neg\varphi, x) & \coloneqq & \neg\mathsf{mso}(\varphi, x) \\
\mathsf{mso}(\varphi \wedge \psi, x) & \coloneqq & \mathsf{mso}(\varphi, x) \wedge \mathsf{mso}(\psi, x) \\
\mathsf{mso}(\circ\varphi, x) & \coloneqq & \exists y.succ(x,y) \wedge \mathsf{mso}(\varphi, y) \\
\mathsf{mso}(\varphi \,\mathsf{U}\, \psi, x) & \coloneqq & \exists y.(x \leq y \leq last) \wedge \mathsf{mso}(\psi, y) \\
 & & \quad \wedge \forall z.(x \leq z < y \rightarrow \mathsf{mso}(\varphi, z)) \\
\mathsf{mso}(\exists X.\varphi, x) & \coloneqq & \exists X.\mathsf{mso}(\varphi, x)
\end{array}$$

Theorem 19. *For any closed* QLTL$_f$ *formula* φ *and finite trace* t, $t, i \vDash \varphi$ *iff* $t, [x/i] \vDash \mathsf{mso}(\varphi, x)$.

This translation gives us the following technique to solve synthesis for a QLTL$_f$ formula. Once we have translated the formula to MSO, we can use MONA to obtain the DFA and then play the DFA game to solve synthesis.

Theorem 20. *The technique for* QLTL$_f$ *synthesis is correct.*

As a result, we can correctly solve our problem:

Theorem 21. *The technique to solve synthesis under unreliable input with* LTL$_f$ *main specification* φ_m *and* LTL$_f$ *backup specification* φ_b*, based on synthesis for the* QLTL$_f$ *formula* $\psi = \varphi_m \wedge \forall U_1.\ldots\forall U_n.\varphi_b$*, is correct.*

As we already discussed, we can only upper-bound the runtime using MONA as worst-case non-elementary because of the MSO-to-DFA translation. This technique can be easily implemented using Syft (Zhu et al. 2017), an open-source LTL$_f$ syntheziser.[3] The only modifications to Syft's pipeline happen before running synthesis; only the input we generate for MONA changes. It is interesting to notice that the implementations of the other techniques end up using MONA to create the DFA, too, since they use Syft.

8 Empirical Evaluation

We implemented the algorithms to empirically evaluate their performance. While the direct technique is optimal in the worst case, our empirical results show that other techniques perform comparably or even better.
Implementation. We built the techniques on top of the LTL$_f$-synthesis tool Syft, reusing existing implementations for belief-state and direct automata construction (Tabajara and Vardi 2020) with some adjustments.
Benchmarks. We used instances of the examples of varying sizes: hiker (trail length 10 to 50, increments of 5), sheep (even input sizes up to $n = 10$), and trap graphs (multiple sizes). For sheep, we restricted our attention to even input sizes as otherwise, trivially synthesis under unreliable input is impossible since standard synthesis already is. For $n > 10$, DFA-generation using MONA timed out.

[2]In QLTL$_f$ and MSO, with a little abuse of notation, we do not distinguish between second-order variables and propositions - open variables act as propositions.

[3]The original Syft source code is available at github.com/Shufang-Zhu/Syft. For our modifications, see the earlier linked repository.

Experimental Setup. Experiments were conducted on a high-performance compute cluster running Red Hat Enterprise Linux 8.10. Tests used an Intel Xeon Platinum 8268 processor (2.9 GHz) with 256 GB RAM per test, executed on a single CPU core.

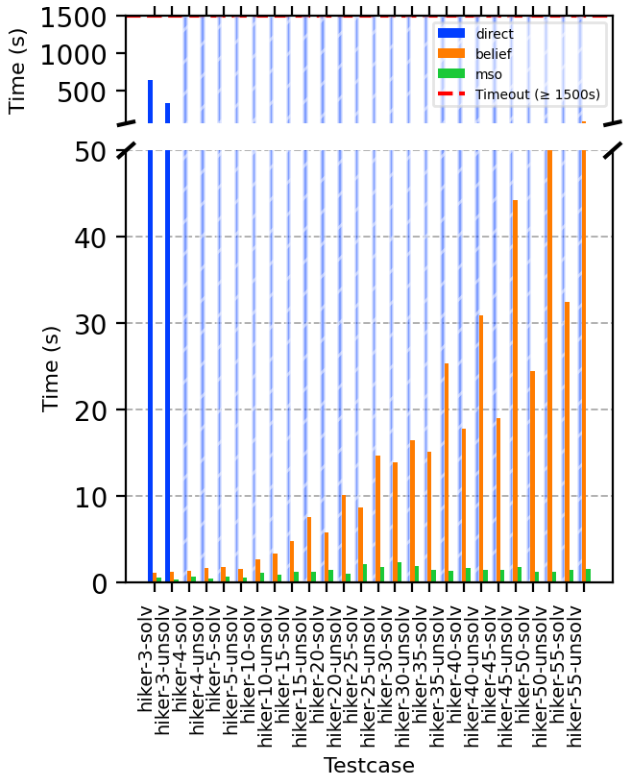
Experimental Results. We report the results for each example; all terminating runs produced correct results. Graphs of the runtime on different instances of the examples can be found in Figures 1 and 2. We used an average of 2 runs to produce the results. Given the deterministic nature of our algorithms, the additional run is only for double-checking. The y-axis shows the total runtime in seconds (DFA construction and synthesis combined) for our test suite, while the x-axis lists each test name. Each test has three bars representing runtimes for the direct automata (blue), belief state (orange), and MSO approach (green). Striped bars reaching the red timeout line indicate tests that ran out of memory or exceeded 30 minutes.

The main findings are this: Considering only the amount of test cases solved within the time limit, MSO performs best, with belief-state second and direct approach solving the least and noticeably smaller amount. We conjecture that this is partly due to MONA's efficient implementation and its already minimized DFAs, unlike the non-minimized product DFAs we compute, which worsen runtime for synthesis.

In general, the runtime of MSO is lower (and, for bigger examples, orders of magnitude lower) than that of the other approaches. For the hiker example, both MSO and belief state construction can solve all test cases up to 50 in length. In contrast, projection only manages to solve very small examples. Here, it is notable that many tests timed out during DFA construction, suggesting that constructing the automaton for the reversed language for these examples is hard. This pattern also shows up in the other examples, albeit less pronounced. As discussed before, in sheep and trap, as the number of sheep or traps increases, the number of variables also increases, while for hiker, this is constant (6 input variables, 3 output variables); hence, hiker examples are solvable for larger values. We performed a Wilcoxon-signed rank test with a significance level of $\alpha = .05$, both for each example individually and for the whole dataset. The runtimes are significantly different between approaches, except for the runtime difference between direct and belief approaches on the trap examples. We compared our MSO synthesis technique's performance with synthesis for the main and backup formula under full/partial observability, respectively. Our runtime is proportional to the sum of both, with a maximum 2.5x overhead.

9 Related Work

Our work is related to previous investigations on LTL$_f$ synthesis under partial observability, a problem that was originally introduced by De Giacomo and Vardi (2016). While related, our problem is novel: it requires



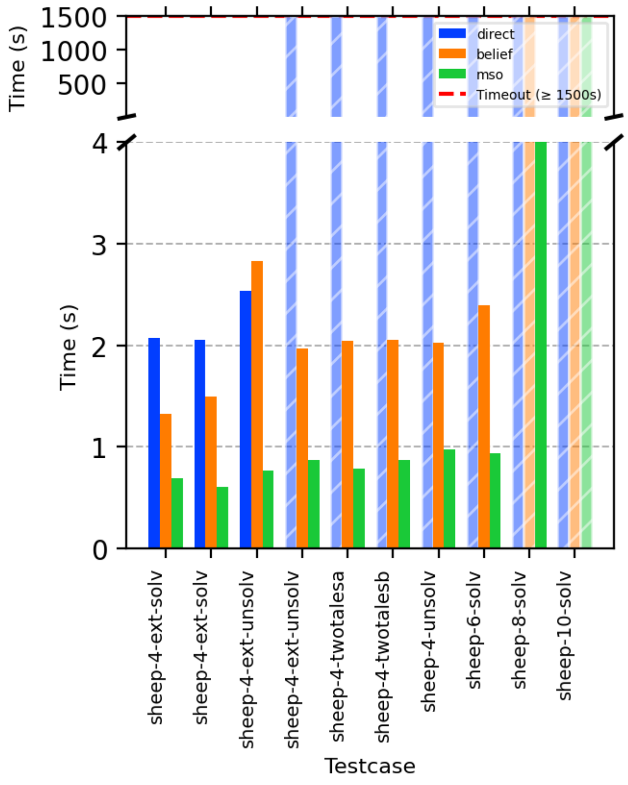


Figure 1: Runtime for the sheep and hiker examples.

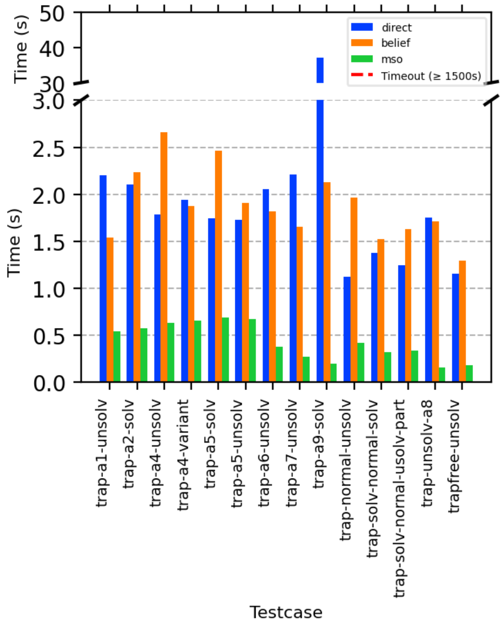


Figure 2: Runtime for the trap examples.

the agent to simultaneously play two games in different arenas with distinct objectives, one corresponding to standard synthesis and the other to synthesis with partial information, thereby generalizing the problem. Further, Tabajara and Vardi (2020) empirically investigate implementing algorithms for LTL$_f$ synthesis under partial observability, translating the results from De Giacomo and Vardi (2016) into practice. They also discuss an MSO approach (though they do not discuss QLTL$_f$ explicitly), the direct approach, and belief state construction. Our empirical results, however, are somewhat different; in our benchmarks, we never encountered situations where the direct approach or belief state construction solved instances that were not solvable using MSO, which they encounter in all their benchmarks. This may be because we have both a main and backup specification, and the larger main specification often dominates the runtime; additionally, the experimental setups (i.e., amount of memory) were different. Similarly related is work on LTL synthesis under partial information (Kupferman and Vardi 2000; Ehlers and Topcu 2015). However, we did not investigate embeddings into LTL in our setting because of the generally better scalability of LTL$_f$ synthesis (Zhu et al. 2020).

There are both stochastic and non-stochastic related approaches. De Giacomo, Parretti, and Zhu (2023) consider computing best-effort LTL$_f$-strategies. While in both their setting and ours multiple variants of the environment are considered, their best-effort synthesis assumes a fully reliable and observable environment, which does not apply to our framework, and rather models the agent being uncertain about the specific environment and not errors in the input. In the stochastic setting, Yu et al. (2024) studied the trembling hand problem, which refers to scenarios where the agent may instruct a wrong action (with a certain probability). In contrast, in our setting, the unreliability is on the environment. Multiple works use partially observable Markov decision processes to cope with uncertainty about the environment (Hibbard et al. 2019; Lauri, Hsu, and Pajarinen 2022). More closely related is planning under partial observability; for example, Aineto et al. (2023) study planning where the agent's actions may fail up to k times, which is similar to our framework and could be modeled in it. Similarly, Aminof et al. (2022) consider planning for multiple agents in a partially reliable environment simultaneously. Our work is also related to work on planning with soft and hard LTL/LDL$_f$ goals. For example, Rahmani and O'Kane (2020, 2019) consider the problem of satisfying an LTL specification while guaranteeing that a subset of some soft constraints (expressed in LDL$_f$ or LTL, respectively) is satisfied (where they are ordered by priority). Guo and Zavlanos (2018) address the synthesis of control policies for MDPs that ensure high-probability satisfaction of LTL tasks while optimizing total cost; their method too employs a product of automata, but additionally involves solving two linear programs.

Wu (2007) characterizes the expressive power of QLTL, observing that one alternation already suffices to express all ω-regular languages. Recently, there has been renewed interest in second-order quantification in infinite-trace variants of LTL (Piribauer et al. 2021); however, we are not aware of prior work on QLTL$_f$.

10 Conclusion

In this paper, we investigated reactive synthesis with backup goals for unreliable inputs using LTL$_f$ as our specification language. We presented three algorithms, two of which match the 2EXPTIME-hardness result, of which the MSO approach performs best. Moreover, we showed how our problem can be seen as an instance of QLTL$_f$ synthesis. While we investigated synthesis with unreliable input in the context of LTL$_f$, but it would be interesting to extend this study to other forms of specifications, possibly distinguishing the formalism used for the main goal and the backup one (i.e., using LTL safety specifications, Aminof et al. (2023)); furthermore one could here explore where the backup goal is satisfied earlier or later than the main goal. Additionally, it may be instructive to consider our problem in planning domains. Our techniques do not rely on probabilities and always ensure the backup condition is met, which is crucial for safety-critical scenarios. For less critical systems, this requirement may be relaxed by using quantitative techniques. In this paper, we have also introduced techniques for synthesis in QLTL$_f$. QLTL$_f$ deserves further study, including whether other problems can be cast naturally as QLTL$_f$ synthesis.

Acknowledgments

We would like to thank Antonio Di Stasio, Shufang Zhu and Pian Yu for insightful discussions on the topic of this paper. We would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out the experiments. This research has been supported by the ERC Advanced Grant WhiteMech (No. 834228).

References

Aineto, D.; Gaudenzi, A.; Gerevini, A.; Rovetta, A.; Scala, E.; and Serina, I. 2023. Action-failure resilient planning. In *ECAI 2023*, 44–51. IOS Press.

Aminof, B.; De Giacomo, G.; Di Stasio, A.; Francon, H.; Rubin, S.; and Zhu, S. 2023. LTLf Synthesis Under Environment Specifications for Reachability and Safety Properties. In Malvone, V.; and Murano, A., eds., *Multi-Agent Systems - 20th European Conference, EUMAS 2023, Naples, Italy, September 14-15, 2023, Proceedings*, volume 14282 of *Lecture Notes in Computer Science*, 263–279. Springer.

Aminof, B.; Murano, A.; Rubin, S.; and Zuleger, F. 2022. Verification of agent navigation in partially-known environments. *Artificial Intelligence*, 308: 103724.

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

Baier, J. A.; and McIlraith, S. A. 2006. Planning with First-Order Temporally Extended Goals using Heuristic Search. In *AAAI*, 788–795. AAAI.

Bloem, R.; Chockler, H.; Ebrahimi, M.; and Strichman, O. 2019. Synthesizing Reactive Systems Using Robustness and Recovery Specifications. In *FMCAD*, 147–151. IEEE.

Calvanese, D.; De Giacomo, G.; and Vardi, M. Y. 2002. Reasoning about Actions and Planning in LTL Action Theories. In *KR*, 593–602. Morgan Kaufmann.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Strong Planning in Non-Deterministic Domains Via Model Checking. In *AIPS*, 36–43. AAAI.

De Giacomo, G.; Parretti, G.; and Zhu, S. 2023. Symbolic LTLf best-effort synthesis. In *20th European Conference on Multi-Agent Systems (EUMAS 2023)*, volume 14282, 228–243. Springer Nature.

De Giacomo, G.; and Rubin, S. 2018. Automata-Theoretic Foundations of FOND Planning for LTL$_f$/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*, 854–860. IJCAI/AAAI.

De Giacomo, G.; and Vardi, M. Y. 2015. Synthesis for LTL and LDL on finite traces. IJCAI'15, 1558–1564. AAAI Press. ISBN 9781577357384.

De Giacomo, G.; and Vardi, M. Y. 2016. LTLf and LDLf synthesis under partial observability. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, 1044–1050. IJCAI/AAAI Press.

Ehlers, R.; Lafortune, S.; Tripakis, S.; and Vardi, M. Y. 2017. Supervisory control and reactive synthesis: a comparative introduction. *Discrete Event Dynamic Systems*, 27(2).

Ehlers, R.; and Topcu, U. 2015. Estimator-based reactive synthesis under incomplete information. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, HSCC '15, 249–258. New York, NY, USA: Association for Computing Machinery. ISBN 9781450334334.

Finkbeiner, B. 2016. Synthesis of Reactive Systems. In *Dependable Software Systems Engineering*. IOS Press.

Gabbay, D. M.; Pnueli, A.; Shelah, S.; and Stavi, J. 1980. On the Temporal Analysis of Fairness. In *POPL*, 163–173. ACM Press.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

Guo, M.; and Zavlanos, M. M. 2018. Probabilistic motion planning under temporal tasks and soft constraints. *IEEE Transactions on Automatic Control*, 63(12): 4051–4066.

Henriksen, J.; Jensen, J.; Jørgensen, M.; Klarlund, N.; Paige, B.; Rauhe, T.; and Sandholm, A. 1995. Mona: Monadic Second-order logic in practice. In *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS '95, LNCS 1019*.

Hibbard, M.; Savas, Y.; Wu, B.; Tanaka, T.; and Topcu, U. 2019. Unpredictable planning under partial observability. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2271–2277. IEEE.

Kupferman, O.; and Vardi, M. Y. 2000. *Synthesis with Incomplete Informatio*, 109–127. Dordrecht: Springer Netherlands. ISBN 978-94-015-9586-5.

Kwiatkowska, M. Z. 2016. Model Checking and Strategy Synthesis for Stochastic Games: From Theory to Practice. In *ICALP*, volume 55 of *LIPIcs*, 4:1–4:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Lampis, M.; and Mitsou, V. 2007. The ferry cover problem. In *Proceedings of the 4th International Conference on Fun with Algorithms*, FUN'07, 227–239. Berlin, Heidelberg: Springer-Verlag. ISBN 9783540729136.

Lauri, M.; Hsu, D.; and Pajarinen, J. 2022. Partially observable markov decision processes in robotics: A survey. *IEEE Transactions on Robotics*, 39(1): 21–40.

Piribauer, J.; Baier, C.; Bertrand, N.; and Sankur, O. 2021. Quantified linear temporal logic over probabilistic systems with an application to vacuity checking. In *CONCUR 2021-32nd International Conference on Concurrency Theory*, 1–18.

Pnueli, A. 1977. The Temporal Logic of Programs. In *FOCS*.

Pnueli, A.; and Rosner, R. 1989. On the Synthesis of a Reactive Module. In *POPL*.

Rahmani, H.; and O'Kane, J. M. 2019. Optimal temporal logic planning with cascading soft constraints. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2524–2531. IEEE.

Rahmani, H.; and O'Kane, J. M. 2020. What to do when you can't do it all: Temporal logic planning with soft temporal logic constraints. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6619–6626. IEEE.

Reiter, R. 1991. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. *Artificial and Mathematical Theory of Computation*, 3.

Reiter, R. 2001. *Knowledge in Action*. MIT Press.

Rintanen, J. 2004. Complexity of Planning with Partial Observability. In Zilberstein, S.; Koehler, J.; and Koenig, S., eds., *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004), June 3-7 2004, Whistler, British Columbia, Canada*, 345–354. AAAI.

Sistla, A. P.; Vardi, M. Y.; and Wolper, P. 1985. The Complementation Problem for Büchi Automata with Applications to Temporal Logic (Extended Abstract). In *ICALP*, volume 194 of *Lecture Notes in Computer Science*, 465–474. Springer.

Tabajara, L. M.; and Vardi, M. Y. 2020. LTLf synthesis under partial observability: From theory to practice. *arXiv preprint arXiv:2009.10875*.

The White House. 2024. Back to the Building Blocks: A Path Toward Secure and Measurable Software. Technical report, White House Report, Feb. 2024.

Wu, Z. 2007. On the Expressive Power of QLTL. In *Theoretical Aspects of Computing – ICTAC 2007*, 467–481. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 9783540752905.

Yu, P.; Zhu, S.; De Giacomo, G.; Kwiatkowska, M.; and Vardi, M. Y. 2024. The trembling-hand problem for LTLf planning. In *33rd International Joint Conference on Artificial Intelligence (IJCAI 2024)*. International Joint Conference on Artificial Intelligence.

Zhu, S.; De Giacomo, G.; Pu, G.; and Vardi, M. Y. 2020. LTLf Synthesis with Fairness and Stability Assumptions. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(03): 3088–3095.

Zhu, S.; Pu, G.; and Vardi, M. Y. 2019. First-Order vs. Second-Order Encodings for LTLf-to-Automata Translation. In *Theory and Applications of Models of Computation*, Lecture Notes in Computer Science, 684–705. Switzerland: Springer International Publishing. ISBN 9783030148119.

Zhu, S.; Tabajara, L. M.; Li, J.; Pu, G.; and Vardi, M. Y. 2017. Symbolic LTLf Synthesis. In Sierra, C., ed., *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 1362–1369. ijcai.org.

Reproducibility Checklist

This paper:

- Includes a conceptual outline and/or pseudocode description of AI methods introduced: **YES**
- Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results: **YES**
- Provides well-marked pedagogical references for less-familiar readers to gain the background necessary to replicate the paper: **YES**

Does this paper make theoretical contributions? **YES**

If yes, please complete the list below.

- All assumptions and restrictions are stated clearly and formally. **YES.**
- All novel claims are stated formally (e.g., in theorem statements). **YES.**
- Proofs of all novel claims are included.
  YES. It is in the supplementary material.
- Proof sketches or intuitions are given for complex and/or novel results. **YES.**
- Appropriate citations to theoretical tools used are given. **YES.**
- All theoretical claims are demonstrated empirically to hold. **YES.**
- All experimental code used to eliminate or disprove claims is included. **YES.**

Does this paper rely on one or more datasets? **NO.**

Does this paper include computational experiments? **YES.**

If yes, please complete the list below.

- Any code required for pre-processing data is included in the appendix.
  NA. Our experiments do not use datasets. There is no need to pre-process data.
- All source code required for conducting and analyzing the experiments is included in a code appendix.
  YES.
- All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes. **YES.**
- All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from.
  YES.
- If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results.
  NA. The algorithms are completely deterministic.
- This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks.
  YES. It is in the paper (specs of the computing infrastructure) and supplementary material (versions of libraries).
- This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics. **YES.**
- This paper states the number of algorithm runs used to compute each reported result.
  YES
- Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information.
  No. Our experiments report answers to yes/no queries and computation times for these queries; moreover our algorithms are deterministic; thus we do not believe that we need more sophisticated measures of performance.
- The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank).
  YES
- This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments.
  NA. There are no (hyper-)parameters.
- This paper states the number and range of values tried per (hyper-)parameter during development of the paper, along with the criterion used for selecting the final parameter setting.
  NA. There are no (hyper-)parameters.

A Sheep Example

Consider a farmer who has n sheep $S = \{1, \ldots, n\}$ that are standing on the left side of a river. She only has a boat to cross the river. However, the boat can only move exactly two sheep at one point in time; additionally, some sheep $\mathcal{D} \subseteq S \times S$ do not like each other and cannot be moved across in the same time instance. Additionally, the farmer believes some sheep currently like each other $L \subseteq \{1, \ldots, n\} \times \{1, \ldots, n\}$, but she may be mistaken about this. The main goal is to move all sheep across; however, if the farmer is mistaken about some of the sheep pairs belonging to set D or L, then she is fine with moving a special subset $S' \subseteq S$ across.

Let us now describe how we translate this problem into an instance of LTL$_f$ synthesis under unreliable input. We introduce n atomic variables left_i that are true if the i-th sheep is currently on the left side. Additionally, we introduce output variables $\mathsf{move_i}$ that allow the agent to ask the environment to move a sheep. Lastly, for each pair $(i, j) \in \mathcal{D} \cup \mathcal{L}$, we introduce a variable $\mathsf{disallow_{i,j}}$ that indicates whether the sheep are allowed to move together.

Environment Description: We first describe the environment constraints:

- Initially, all sheep are left i.e. $\bigwedge_{s \in \mathcal{S}} \mathsf{left}_s$ needs to be true.
- If a sheep is not requested to move, its position stays the same:
$$\bigwedge_{s \in \mathcal{S}} \Box \left(\bigcirc (\neg \mathsf{move_i}) \land \neg \mathsf{left_i} \supset \bigcirc (\neg \mathsf{left_i}) \right)$$
and
$$\bigwedge_{s \in \mathcal{S}} \Box \left(\circ (\neg \mathsf{move_i}) \land \mathsf{left_i} \supset \circ (\mathsf{left_i}) \right).$$
- We also need to specify when the sheep actually move. We split this into generation for pairs (i, j) such that sheep i and sheep j could be blocked and pairs that are always unblocked (as for efficiency, the variables about blocking only exist for potentially blocked pairs).
For pairs that will never be blocked, we introduce
$$\Box \left(\left(\bigcirc (\mathsf{move_i} \land \mathsf{move_j}) \land \mathsf{left_i} \land \mathsf{left_j} \right) \supset \bigcirc (\neg \mathsf{left_i} \land \neg \mathsf{left_j}) \right).$$
For possibly blocked pairs, we introduce the following two constraints:
$$\Box((\bigcirc (\mathsf{move_i} \land \mathsf{move_j} \land \neg \mathsf{disallow_{i,j}}) \land \mathsf{left_i} \land \mathsf{left_j})$$
$$\supset \bigcirc (\neg \mathsf{left_i} \land \neg \mathsf{left_j})),$$
- We do not need to specify other movement constraints, as when we do not specify whether an action must lead to the sheep being moved, the environment can adversarially choose not to move the sheep.

$$\Box \left(\bigcirc (\mathsf{move_i} \land \mathsf{move_j} \land \mathsf{disallow_{i,j}}) \land \mathsf{left_i} \land \mathsf{left_j} \right)$$
$$\supset \bigcirc (\mathsf{left_i} \land \mathsf{left_j}).$$

- Lastly, we need to specify the initial situation description, namely forcing the values of the disallow variables to never change, thus for the main specification (but not for the backup one), we force that for $(i, j) \in L$ we have $\Box(\neg \mathsf{disallow}_{i,j})$ and for $(i, j) \in D$ we have $\Box(\mathsf{disallow}_{i,j})$.

With φ_e (and $\varphi_{e'}$ for the backup specification) , we denote the conjunction of the above formulas.

Agent Constraints: Secondly, we have to specify the agent constraints φ_{ag} as a conjunction of the following formulas:

- The agent has to ensure that exactly two move variables are active. Thus we set the agent formula to be
$$\Box(\mathsf{exactly} - 2 - \mathsf{of}(\mathsf{move_1}, \ldots, \mathsf{move_n})).$$
To encode the exactly-2-constraint, we use a simple quadratic encoding.

Lastly, we need to specify the goal; here it is that eventually, all of the sheep are moved, i.e. $\varphi_{goal} := \Diamond \left(\bigwedge_{i \in \mathcal{S}} \neg \mathsf{left_i} \right)$; and for the backup specification we have $\varphi_{goal'} := \Diamond \left(\bigwedge_{i \in \mathcal{S}'} \neg \mathsf{left_i} \right)$.

Partitioning: As a partitoning, we set all the disallow variables as unobservables.

In summary, we are trying to synthesize a strategy realising $\varphi_{ag} \land (\varphi_e \supset \varphi_{goal})$, with backup-goal $\varphi_{ag} \land (\varphi'_e \supset \varphi'_{goal})$, where the goal is to only have $\Diamond \bigwedge_{i \in \mathcal{S}'} \neg \mathsf{left_i}$.

B Graph Example

Let us first describe the variables. We have $\lceil \log_2 n \rceil$ variables pos_i, which represent the current position of the robot in the graph. Additionally, for each trap, we introduce a variable t_i that denotes whether the trap is on or off. These are all of the input variables. The robot has a single output variable left – indicating whether to go left or right.

Environment Description: Since the agent can only move left or right, we do not need any constraints. The constraints we generate for the environment are based on having conjuncts of the form $\Box(p(e_s) \land \bullet(left) \supset \bullet(p(e_t)))$ for any edge $e = (e_s, e_t)$, where $p(n)$ is the propositional formula corresponding to being in state n (i.e. using the unique binary encoding of the state). For edges that are used in traps, we add t_i or $\neg t_i$ as a conjunct to the left-hand side and, of course, adapt the goal state analogously. If a vertex has only 0 or 1 edge, we specify that the agent remains at that position.

Additionally, we have a condition that the traps cannot change, as this would make it impossible . This is just by having formulas of the form $t_i \supset \Box(t_i)$ and $\neg t_i \supset \Box(\neg t_i)$.

Goal: For the normal description, the goal is just to eventually reach one of the goal states; for the backup version, we allow the safety states as well. Thus, both are of the form $\varphi_{env} \supset \varphi_{goal}$.

Partitioning: The partitioning has all trap variables t_i as unobservable.

C Hiker Example

We can model this as a problem of synthesis under unreliable input in the following way: The environment controls the following variables:

- **berry**: If true, signifies that at the current position on the hiking trail, there is a berry that the agent can eat.
- **poison**: If true and berry is true, it indicates that the berry at the current position on the trail is poisonous.
- **herbs**: If true, indicates that at the current position of the hiking path, there are medical herbs that the agent could take to possibly alleviate sickness now or later.
- **sick**: If true, indicates that the hiker is currently sick.
- **eot**: If true, signifies that the end of the hiking trail was reached.
- **inbag**: If true, the herbs are currently in the hiker's bag and could be used.

The agent has control of the following variables:

- **eat**: If true, it signifies that the agent wants to eat the berry (if existent) at the current position of the hiking path.
- **takeMedication**: Take the herbal medication; if the hiker was sick before, they will no longer be.
- **collectMedication**: If there is herbal medication, the hiker can collect it (notice that their bag only allows them to store one piece of herbs).

Our formula, as in the other examples, will be of the shape $\varphi_{env} \supset \varphi_{ag}$.

Environment Description: We can describe the parts of the environment formula similar to SSA axioms in Reiter's Basic Action Theory. Thus, we basically have one successor state axiom for each of the environment variables.

We do not have this for berry, poison and herbs, as those are randomized by the environment; however, we here include the constraint that poison is only true when berry also is true and that berries and herbs cannot be there at the same time.

$$\Box\,(berry \supset \neg herbs)$$

$$\Box\,(poison \supset berry)$$

For sickness, we can notice that the hiker is sick when either they were sick before and have not both had medication available and taken medication or when they ate a poisonous berry. This leads to the following successor state axioms:

$$\circ sick \equiv \circ\top \wedge (\circ eat \wedge berry \wedge poison)$$
$$\vee\,(sick \wedge \neg(inbag \wedge takeMedication))$$

For the end of the trail, we only have to specify that it is reached after a certain number of steps (here k).

$$\bullet^{k}(eot) \wedge \bigwedge_{1<k'<k} \bullet^{k'}(\neg eot)$$

Additionally, we describe that once eot is true, it will stay true and no more berries appear before the hiker.

$$\Box(eot \supset \bullet(eot)) \wedge \Box(eot \supset \neg\mathsf{berry})$$

To control realizability, we can influence whether there is medicine along the trail. For the realizable variant, we force that medication is available somewhere along the trace before the end:

$$\bullet^{k-3}(medication)$$

For inbag, we have that it is true if either we have collected herbs in the last step (and there were herbs) or if we already had herbs and have not used them.

$$\circ inbag \equiv \circ\top \wedge (herbs \wedge \circ collectMedication)$$
$$\wedge(inbag \wedge \neg(takeMedication))$$

Goal. The agent's main goal consists of three parts: The hiker wants to eat all non-poisonous berries that they cross during their hiking. Additionally, they want to reach the end of the trail.

$$\varphi_{ag} := \Diamond(eot) \wedge \Box(berry \wedge \neg poison \supset \bullet(eat))$$

Lastly, for the initial state, we specify that the hiker is not sick and, of course, that it is not the end of the trace.

$$\neg sick \wedge \neg eot$$

Partitioning: For the partial observability variant, we set as unobservables $\mathcal{X}_{unr} = \{poison\}$.

D Implementation Details

The source code of the modified version of Syft that we use for our experiments is included in the supplementary material.

Each program can be run using the command `./Syft input.ltlf paritioning.part 0 <mode>`. With `mode`, it is possible to select the approach by setting it to either `direct, belief, mso`, for the direct approach, the belief-state construction or the encoding into MSO, respectively.

Input file format: It is important to notice that since our problem is different from standard LTL$_f$-synthesis, the layout of the input files is slightly changed. Here, `input.ltlf` is a file containing two lines defining the main and backup LTL$_f$-formula, respectively.

The formula itself is specified in Syft's LTLf syntax using `N` for next, `X` for Weak Next, `G` for always, `F` for eventually and usual symbols for propositional connectives.

Partitioning file: The partition file contains the necessary information. It has the following format:

```
.inputs: a b c
.outputs: x y z
.unobservables: b c
```

This would correspond to a partitioning with $\mathcal{X} = \{a, b, c\}, \mathcal{Y} = \{x, y, z\}$ where $\mathcal{X}_{rel} = \{a\}$ and $\mathcal{X}_{unr} = \{b, c\}$. Notice that we **require** each unreliable input variable to be listed **twice**: Once in the inputs and in the list of unobservables.

Tests. The tests can be found in the `tests/` directory. Each test is in its own subdirectory and consists of a partitioning file `test.part`, a corresponding formula file `test.ltlf` and a file `expected` that either contains a 0 or 1, indicating whether the test is unrealizable or realizable, respectively.

The tests can be run using the python script `runTests.py` in the directory `scripts/runTests`. As an argument it takes the implementation type (i.e., mso, direct or belief), the other arguments can be viewed by calling it with `-h` to display the help.

Test Generation. The tests (for sheep, hiker and graph) can be generated using the Python scripts in the `generators/` folder. The other test cases (i.e. `simpl-unrel`) are not auto-generated.

Compilation.

Syft requires some libraries to be installed, this includes CUDD (vesion 3.0.0), Boost (v. 1.82) and MONA (used version MONA v1.4-18). The code was compiled using g++ (version 13.2.0). We have included the version of CUDD we used in the supplementary material (`extern/` folder), as different versions may lead to incompatibilities.

There are detailed installation instructions for Ubuntu-based systems in the `Syft/INSTALL` path of the archive.

E Proofs

Direct Approach

Theorem 22. *Let A_1, A_2 be DFAs over the same alphabet, i.e., $\mathcal{G}_i = (2^{\mathcal{X}} \times 2^{\mathcal{Y}}, S_i, s_{0,i}, \delta_i, F_i)$ (with $i \in \{1, 2\}$, then a trace $t \in L(G_1 \otimes G_2)$ if and only if $t \in L(G_1) \wedge t \in L(G_2)$.*

Proof. From left-to-right assume that $t \in L(G_1 \oplus G_2)$. This is the case if and only if running t on the product automaton produces a sequence of states $((s_{0,1}, s_{0,2}, \ldots, (s_{i,1}, s_{j,2}))$ with $s_{i,1} \in F_1$ and $s_{i,2} \in F_2$. By construction of the transitions, this is the case if and only if running t on $\mathcal{G}_1$ produces the state sequence $(s_{0,1}, \ldots, s_{i,1})$ and running t on $\mathcal{G}_2$, but this is the case iff $t_1 \in L(\mathcal{G}_1)$ and $t_2 \in L(\mathcal{G}_2)$. $\square$

Lemma 23. *For any trace t, we have $t \in L(N_{\exists U_1,\ldots,U_n})$ if and only if there is a trace $t' \sim_{-U_1,\ldots,U_n} t$ such that $t' \in L(N)$.*

Proof. Assume that $t \in L(N_{\exists U_1,\ldots,U_n})$. This implies that for running t on $\mathcal{N}$ there is a possible sequence of states $s_0, \ldots, s_k$ with $s_k \in F$. By definition, each transition $(s_i, t(i), s_{i+1})$ was possible since there was a way to modify $t(i)$ on the existentially abstracted variables such that the original automaton transitioned from s_i to s_{i+1}. Putting these together gives a trace t' that satisfies the definition.

The other direction is similar, using that we can arbitrarily modify the variables by existentially abstracting.

For the other direction, let t be a trace s.t. there is a trace t' with $t \sim_{-U_1,\ldots,U_n} t'$ and $t' \in L(N)$. We need to show that $t \in L(N_{\exists U_1,\ldots,U_n})$. Since $t' \in L(N)$, there is a sequence of states $(s_0, \ldots, s_k)$ with $s_k \in F$ and $(s_i, t'(i), s_{i+1}) \in \delta$. Now, since $t \sim_{-U_1,\ldots,U_n} t'$, we can use the same transition as we did for the modified trace, and thus to reach s_k and therefore $t \in L(N_{\exists U_1,\ldots,U_n})$. $\square$

Lemma 24. *We have that $t \in \overline{\mathcal{D}(N_{\exists U_1,\ldots,U_n})}$, if and only if for any trace $t' \sim_{-U_1,\ldots,U_n} t$,we have $t' \notin L(N)$.*

Proof. Assume that $t \in L(\overline{\mathcal{D}(N_{\exists U_1,\ldots,U_n})})$. This is the case if and only if $t \notin L((N_{\exists U_1,\ldots,U_n}))$. By Lemma 23 this is the case if and only if there is no trace $t' \sim_{-U_1,\ldots,U_n} t$ s.t. $t' \in L(N)$, which is equivalent to claiming that for any trace $t' \sim_{-U_1,\ldots,U_n}$ we have $t' \notin L(N)$. $\square$

Corollary 25. *We have that $t \in \overline{\mathcal{D}((\mathcal{A}_{\neg\varphi_b})_{\exists U_1,\ldots,U_n})}$, if and only if for any trace $t' \sim_{-U_1,\ldots,U_n} t$,we have $t' \vDash \varphi_b$.*

Proof. Let $t \in L(\overline{\mathcal{D}((\mathcal{A}_{\neg\varphi_b})_{\exists U_1,\ldots,U_n})})$. Let $t' \in \mathsf{exp}_{\mathcal{X}_{unr}}(\mathsf{proj}_{X_{unr}}(t))$. By Lemma 24, we then have that $t' \notin L(\mathcal{A}_{\neg\varphi_b})$. But this implies that $t' \vDash \varphi_b$, just as we needed.

For the other direction, assume that for any trace $t' \sim_{-U_1,\ldots,U_n} t$,we have $t' \vDash \varphi_b$. Assume for the sake of deriving a contradiction that $t \notin \overline{\mathcal{D}((\mathcal{A}_{\neg\varphi_b})_{\exists U_1,\ldots,U_n})}$. This would imply that $t \in \mathcal{D}((\mathcal{A}_{\neg\varphi_b})_{\exists U_1,\ldots,U_n})$, which would imply (by Lemma 24) that there is a trace $t'' \sim_{-U_1,\ldots,U_n} t$ s.t. $t'' \vDash \neg\varphi$. This would contradict our initial assumption. $\square$

Theorem 9. *Solving synthesis for the synchronous product of the* LTL$_f$*-automaton for φ_m, A_{φ_m}, and the automaton $\overline{\mathcal{D}((\mathcal{A}_{\neg\varphi})_{\exists U_1,\ldots,U_n})}$ solves synthesis under unreliable input.*

Proof. We have to argue that any strategy that guarantees reaching a final state in the DFA game, also ensures that synthesis under unreliable input is solved. Thus let σ be a strategy that wins in the synchronized product, thus there is an index k, where the transitions induced by the strategy of the DFA end in an accepting state. We have to show that σ satisfies the conditions for synthesis under unreliable input. Therefore, let $\lambda = (X_0, \dots,) \in (2^{\mathcal{X}})^\omega$ be arbitrary. Since σ is a winning strategy in the DFA game, there is an index k s.t., the state sequence induced by $t = ((Y_0, \sigma(Y_0)), \dots,)$ is in a final state. We show that the same k also satisfies the conditions for synthesis under unreliable input.

First, since σ wins in the DFA-game, we know that $t \in L(\mathcal{A}_{\varphi_m} \otimes D((\mathcal{A}_{\neg\varphi})_{\exists U_1,\dots,U_n}))$ we have (by Theorem 22) that $t \in L(\mathcal{A}_{\varphi_m})$, which is the case if and only if $t \vDash \varphi_m$. The other requirement to solve synthesis under unreliable input, follows using Corollary 25 and theorem 22.

For the other direction, assume that σ solves synthesis under unreliable input. We have to show that σ is also a winning strategy in the DFA game. Therefore, let $\lambda = (X_0, \dots,) \in (2^{\mathcal{X}})^\omega$ be arbitrary. Since σ is strategy realizing synthesis under unreliable input, there is an index k s.t., $t \vDash \varphi_m$ and for any $t' \sim_{-U_1,\dots,U_n} t$ we have that $t' \vDash \varphi_b$. We need to argue that the DFA ends up in an accepting state.

Again, it suffices to show both $t \in L(\mathcal{A}_{\varphi_m})$ and $t \in L(\mathcal{D}((\mathcal{A}_{\neg\varphi})_{\exists U_1,\dots,U_n}))$. Again, the first is immediate from the correctness of the automaton. The second follows directly from Corollary 25. □

Belief State Approach

Lemma 26. *For any two traces t, t' with $t' \in \mathsf{exp}_{\mathcal{X}_{unr}}(\mathsf{proj}_{\mathcal{X}_{unr}}(t))$ we have that $t \in L(\mathcal{G}^{rel}_{\mathcal{A}}) \iff t' \in L(\mathcal{G}^{rel}_{\mathcal{A}})$*

Proof. Follows by a simple induction, from the fact that the transitions are invariant under changing the unobservable variables. □

Lemma 27. *If a trace t induces a play $(S_0, S_1, \dots, S_{t+1})$ in G^{rel}_{A} and for any $s_{t+1} \in S_{t+1}$, then there is a trace $t' \sim_{-\mathsf{U}_1,\dots,U_n} t$ that induces a play $\rho' = (s_0, \dots, s_{t+1})$ on $\mathcal{A}$.*

Proof. The proof is by induction on the length of the trace. The base case for traces of length 0 is trivial, as only $s_0 \in S_0$. For the inductive case, we consider the subtrace $(S_0, \dots, S_{i-1})$.

We know that since $s_i \in S_i$, there is a transition $(s, X \cup Y, s_i)$. Moreover, using the inductive hypothesis, there is a trace t' of length i that leads to s in the original automaton. If we consider $t = t'.(X, Y)$, this trace will lead to the state s_i. □

Lemma 28. *If $t \in L(\mathcal{G}^{Obs}_{A})$, then $t \in L(\mathcal{A})$.*

Proof. Use the fact that each step of the original automaton is also *tracked* in the power sets, i.e. for the sequence of states $(s_0, \dots, s_t, s_{t+1})$ generated by running t on $\mathcal{A}$, we will always have that $s_i \in S_i$ where S_i are the states of the belief state automaton (Lemma 27).

But this implies that the last state of running t on the original automaton is included in the last set S_n we transition into, and since this is a final state of the belief state automaton, we know that $s_{t+1} \in F$ and thus $t \in L(A)$. □

Theorem 29. *For any trace t, $t \in \mathcal{L}(G^{rel}_{\mathcal{A}_{\varphi_b}})$ if and only if for any t' with $t \sim_{-U_1,\dots,U_n} t'$ we have $t' \vDash \varphi_b$.*

Proof. Assume $t \in \mathcal{L}(G^{rel}_{\mathcal{A}_{\varphi_b}})$. Assume for the sake of deriving a contradiction that there is a trace $t' \in \mathsf{exp}_{\mathcal{X}_{unr}}(\mathsf{proj}_{\mathcal{X}_{unr}}(t')$, such that $t' \nvDash \varphi_b$. But since $t \in L(G^{rel}_{\mathcal{A}_{\varphi_b}})$, by Lemma 26, we would also have that $t' \in L(G^{rel}_{\mathcal{A}_{\varphi_b}})$ and thus (by Lemma 28) that $t' \in L(\mathcal{A}_{\varphi_b})$ and therefore $t' \vDash \varphi_b$ – a contradiction.

For the other direction, assume for any t' with $t' \sim_{-U_1,\dots,U_n} t$ we have $t' \vDash \varphi_b$ but assume for the sake of contradiction, that $t \notin L(G^{rel}_{\mathcal{A}_{\varphi_b}})$. This can only be the case if a non-accepting state is among the state-set the belief state automaton is in after running t. This would imply (by Lemma 27) that there is a trace $t' \in \mathsf{exp}_{\mathcal{X}_{unr}}(\mathsf{proj}_{\mathcal{X}_{unr}}(t))$ s.t. $t' \nvDash \varphi_b$ (as it does not end in an accepting trace of the deterministic automaton). But this contradicts our initial assumption. □

Theorem 13. *Solving synthesis for the synchronous product of the* LTL$_f$*-automaton for φ_m, A_{φ_m}, and the belief-state automaton $G^{rel}_{\mathcal{A}_{\varphi_b}}$ solves synthesis under unreliable input.*

Proof. The proof is almost identical to the one for Theorem 9. To see this, notice that Theorem 29 and corollary 25 together imply that the belief state and projection automaton both recognize the same traces, thus the correctness argument is identical. □

Translation of Quantified LTL$_f$ to MSO

Definition 30 (MSO Semantics)**.** Given a domain $\mathcal{U} = \{1, \dots, n\}$, and two functions $v : \mathcal{V}_1 \to \mathcal{U}$ assigning first-order variables to members of U and a $w : \mathcal{V}_2 \to \mathcal{U}$ assigning second-order variables to subsets of U, we define entailment for MSO:

$$\begin{aligned}
u, w \vDash X(x) &\Leftrightarrow u(x) \in w(X)\\
u, w \vDash x < y &\Leftrightarrow u(x) < u(w)\\
u, w \vDash \varphi \wedge \psi &\Leftrightarrow (u, w \vDash \varphi) \wedge (u, w \vDash \psi)\\
u, w \vDash \neg\varphi &\Leftrightarrow \neg(u, w \vDash \varphi)\\
u, w \vDash \exists x.\varphi &\Leftrightarrow \exists \hat{x} \in \mathcal{U} \text{ s.t. } u[x/\hat{x}], w \vDash \varphi\\
u, w \vDash \exists X.\varphi &\Leftrightarrow \exists \hat{X} \subseteq \mathcal{U} \text{ s.t. } u, w[X/\hat{X}] \vDash \varphi
\end{aligned}$$

Definition 31. Given a trace t, we define the corresponding assignment of second order-variables by setting $w_t(A) = \{i \mid A \in t(i)\}$ for any $A \in \mathcal{P}$. Then we define that $t, [i/x] \vDash \varphi$ is a shorthand for $[i/x], w_t \vDash \varphi$.

Lemma 32. *Let φ be a* QLTL$_f$ *formula and t be a finite trace. Then $t, i \vDash \varphi$ iff $[x/i], w_t \vDash \varphi$.*

Proof. The proof is by induction on the formula with i, t, x quantified.

- Assume that $\varphi = A$. Then $t, i \vDash A$ if and only if $A \in t(i)$, which is the case iff $A \in t(i)$, this is the case iff $i \in w_t(A)$ which again is the case iff $[x/i], w_t \vDash A(x)$. This concludes this step since $A(x) = \mathsf{mso}(A, x)$.
- Assume that $\varphi = \neg\psi$. Then we know that $t \vDash \neg\psi$ if and only if it is not the case that $t, i \vDash \psi$, which by the IH, is equiavlent to it not being the case that $[x/i], w_t \vDash \mathsf{mso}(\psi, x)$. By semantics this is the case iff $[x/i], w_t \vDash \neg\mathsf{mso}(\psi, x)$ and this is the case if and only if $[x/i], w_t \vDash \mathsf{mso}(\varphi, x)$.
- Assume that $\varphi = \circ\psi$. We prove both directions separately. For the forward direction, assume that $t, i \vDash \circ\psi$. This is the case if and only if $t, i+1 \vDash \psi$. By the inductive hypothesis, this is equivalent to $[y/(i+1)], w_t \vDash \mathsf{mso}_{\mathcal{B}}(\psi, y)$. Since $succ(x, y)$ is interpreted as the successor relation, we have $\sigma[x/i], w_t \vDash \exists y.succ(x, y) \wedge \mathsf{mso}(\psi, y)$. By the definition of the translation function, this is equivalent to $[x/i], w_t \vDash \mathsf{mso}(\circ\psi, x)$.
  For the backward direction, assume that $[x/i], w_t \vDash \mathsf{mso}(\circ\psi, x)$. By definition, this means $[x/i], w_t \vDash \exists y.succ(x, y) \wedge \mathsf{mso}(\psi, y)$. Thus there is a $\hat{y} \in \mathcal{U}$ such that $[x/i][y/\hat{y}], w_t \vDash succ(x, y) \wedge \mathsf{mso}(\psi, y)$. This is the case if both $[x/i][y/\hat{y}], w_t \vDash succ(x, y)$ and $[x/i][y/\hat{y}], w_t \vDash \mathsf{mso}(\psi, y)$. The first implies that $(i, \hat{y})$ are successors, thus $\hat{y} = i + 1$. The second (noticing that x does not occur free in $\mathsf{mso}(\psi, y)$) implies that $[y/i+1], w_t \vDash \mathsf{mso}(\psi, y)$, which by the inductive hypothesis implies that $t, i+1 \vDash \psi$. This directly implies that $t, i \vDash \circ\psi$.
- Assume that $\varphi = \psi \mathsf{U} \chi$. Assume that $t, i \vDash \psi \mathsf{U} \chi$. This is the case if there exists j, with $i \leq j \leq last$, such that $t, j \vDash \chi$ and for all k with $i \leq k < j$, $t, k \vDash \psi$. By the inductive hypothesis, this is equivalent to there existing $j \geq i$ such that $[x/j], w_t \vDash \mathsf{mso}(\chi, x)$ and for all k with $i \leq k < j$, $[x/k], w_t \vDash \mathsf{mso}(\psi, x)$. By MSO semantics and the definition of our model (importantly, since $<$ and S are interpreted as they are on $\mathbb{N}$), this is the case iff $[x/i], w_t \vDash \exists y.(x \leq y \leq \text{last}) \wedge \mathsf{mso}(\chi, y) \wedge \forall z.(x \leq z < y \rightarrow \mathsf{mso}(\psi, z))$. This is, by the definition of the translation function, equivalent to $\mathcal{I}_t, \sigma[x/i] \vDash \mathsf{mso}_{\mathcal{B}}(\psi \mathsf{U} \chi, x)$.
- Assume that $\varphi = \exists X.\, \psi$. We know that $t, i \vDash \exists X.\varphi$ if and only if there is a t' s.t. $t' \sim_{-X} t$ and $t', i \vDash \varphi$, by the IH this is the case iff $[x/i], w_{t'} \vDash \mathsf{mso}(\varphi, x)$; since the two variable assignments w_t and $w_{t'}$ (as can be easily checked) at most differ in their X-assignments, this is the case iff $[x/i], w_t \vDash \exists X.\mathsf{mso}(\varphi, x)$, this is the case iff $[x/i], w_t \vDash \mathsf{mso}(\exists X.\varphi, x)$.

□

As a direct corollary, we have a proof of the correctness statement from the main text:

Theorem 19. *For any closed* QLTL$_f$ *formula φ and finite trace t, $t, i \vDash \varphi$ iff $t, [x/i] \vDash \mathsf{mso}(\varphi, x)$.*

Proof. Directly follows from Lemma 32. □

Quantified LTLf Synthesis

Theorem 16. *A strategy σ realizes the instance of* LTL$_f$ synthesis under unreliable input *with $\mathcal{X}_{unr} = \{U_1, \ldots, U_n\}$ iff it realizes synthesis for the* QLTL$_f$ *formula $\varphi_m \wedge \forall U_1 \ldots \forall U_n.\varphi_b$.*

Proof. From left to right, assume that σ solves synthesis under unreliable input and let $t = ((X_0, Y_0), \ldots, (X_i, Y_i))$ be a trace with $Y_i = \sigma(X_0, \ldots, X_{i-1})$. Then, trivially, we have that $t \vDash \varphi_m$. Thus, to establish that $t \vDash \varphi_m \wedge \forall U_1 \ldots \forall U_n.\varphi_b$ we only need to show $t \vDash \forall U_1 \ldots \forall U_n.\varphi_b$. This can be shown considering that the semantics of $t \vDash \forall U_1 \ldots . \forall U_n.\varphi_b$ unfolds into $\forall t' \in \mathsf{exp}_{\mathcal{X}_{unr}}(\mathsf{proj}_{\mathcal{X}_{unr}}(t)).\, t' \vDash \varphi_b$.

Conversely, assume σ synthesizes the QLTL$_f$ formula. Thus, for any trace $t = ((X_0, Y_0), \ldots, (X_i, Y_i))$ with $Y_i = \sigma(X_0, \ldots, X_{i-1})$, it holds that $t \vDash \varphi \wedge \forall U_1 \ldots . U_n.\varphi_b$. Then trivially (since QLTL$_f$ has the same semantics as LTL$_f$ for its LTL$_f$-fragment), we have $t \vDash \varphi_m$. Additionally, we also have that $t' \vDash \varphi_b$ for any t' with $t' \in \mathsf{exp}_{\mathcal{X}_{unr}}(\mathsf{proj}_{\mathcal{X}_{unr}}(t))$, as essentially any trace in the expansion can be restored by using the universal quantification in QLTL$_f$. □

Definition 33. Given a QLTL$_f$ formula φ in PNF with k-alternations, we can inductively define a deterministic automaton A_φ that recognizes it:

- If φ has one alternation, we either have a bare LTL$_f$-formula, or a a chain of one or multiple quantifiers of the same type:
  - For bare formulas, we just use the LTL$_f$-automaton corresponding to φ.
  - For a single existential alternation i.e. $\exists X_1, \ldots, \exists X_n.\varphi$, we first construct NFA for φ and then existentially abstract over the rest, i.e. $(A_\varphi)_{\exists X_1, \ldots, X_n}$, and determinize it.
  - For a single universal alternation, i.e. $\forall X_1, \ldots, X_n.\psi$, we know that this is the same as $\neg\exists X_1 \ldots \exists X_n.\neg\psi$. We can construct a DFA for ψ, then negate that, obtaining a DFA $A_{\neg\psi}$, we then existentially abstract $(A_{\neg\psi})_{\exists X_1, \ldots, X_n}$ determinize and lastly negate once more, yielding $A_\varphi = \overline{D((A_{\neg\psi})_{\exists X_1, \ldots, X_n})}$.
- If the formula has a higher alternation count it is of the form $\varphi = (\exists/\forall)^+\phi_k$ where ϕ_k has k alternations. We thus, assume (for the inductive definition) to have an automaton A_ψ.
  - If $\varphi = \exists X_1 \ldots \exists X_n.\phi_k$. We build the existentially-abstracted NFA $N_{\exists X_1, \ldots, X_n}(A_{\phi_k})$, and determinise it; i.e. $A_\psi \coloneqq \mathcal{D}(N_{\exists X_1, \ldots, X_n}(A_{\phi_k}))$.
  - If $\varphi = \forall X_1, \ldots X_n.\psi$, then we built the DFA $A_{\neg\phi_k}$ (by negating the DFA provided by the inductive hypothesis). Then we build the NFA $N_{\exists X_1, \ldots, X_n}$, and determinise and negate it; i.e. $A_\varphi = \overline{D(N_{X_1, \ldots, X_n}(A_{\neg\psi}))}$.

Theorem 34. *The automaton A_φ we compute correctly recognises φ (for any* QLTL$_f$ *formula φ in PNF).*

Proof. We have to show that for any formula φ and trace t, $t \vDash \varphi$ iff $t \in L(A_\varphi)$. The proof is by induction on the alternation count.

- Base Case:
  - If φ is a bare LTL$_f$-formula, then the correctness follows from the correctness of translating LTL$_f$ into DFAs.
  - Assume that $\varphi = \exists U_1.\dots\exists U_n.\psi$. Then, $t \in L(\exists X_1.\dots\exists X_n.\psi)$ iff there is a t' with $t' \sim_{-X_1,\dots,X_n} t$ and $t' \vDash \psi$, this is by correctness of LTL$_f$ translation the case if and only if $t' \in L(A_\psi)$. But this is by Corollary 25 the case if and only if $t \in L((A_\psi)_{\exists X_1,\dots,X_n})$; which is the case iff $t \in L(A_\varphi)$ (since determinization does not affect the language).
  - Assume that $\varphi = \forall X_1.\dots\forall X_n.\psi$. Then $t \vDash \varphi$ if and only if for all traces t' with $t \sim_{-X_1,\dots,X_n} t$ we have that $t' \vDash \psi$. This is equivalent to there being no trace t' with $t' \sim_{-X_1,\dots,X_n} t$ with $t' \nvDash \psi$. This is equivalent , by the IH, to there being no trace t' with $t' \sim_{-X_1,\dots,X_n}$ having $t' \notin L(A_\psi)$.
  This is equivalent to to there being no trace t' with $t' \sim_{-X_1,\dots,X_n} t$, we have that $t' \in L(A_{\neg\psi})$. But, this is equivalent it is not the case that there is a trace t' with $t' \sim_{-X_1,\dots,X_n}$ and $t' \in L((A_{\neg\psi})$. But this is the case iff it is not the case that $t \in L((A_{\neg\psi})_{\exists X_1,\dots,X_m})$, this is the case iff $t \in L(\overline{D((A_{\neg\psi})_{\exists X_1,\dots,X_n})})$.
- Inductive case:
  - Assume that $\varphi = \exists X_1 \dots \exists X_n \varphi_{k-1}$. By the inductive hypothesis, there is a DFA $A_{\varphi_{k-1}}$ that recognises φ_{k-1}. Now $t \vDash \exists X_1 \dots \exists X_n.\varphi_{k-1}$ if and only if there is a trace t' with $t' \sim_{-X_1,\dots,X_n} t$ and $t' \vDash \varphi_{k-1}$. This is the case iff there is a trace t' with $t' \sim_{-X_1,\dots,X_n} t$ and $t' \in L(A_{\varphi_{k-1}})$. But this is by Corollary 25 the case if and only if $t \in L((A_{\varphi_{k-1}})_{\exists X_1,\dots,X_n})$. This is the case if and only if $t \in L(A_\varphi)$, since the determinisation does not affect the recognised language.
  - Assume that $\varphi = \forall X_1 \dots \forall X_n \varphi_{k-1}$. Then $t \vDash \varphi$ if and only if for all traces t' with $t \sim_{-X_1,\dots,X_n} t$ we have that $t' \vDash \varphi_{k-1}$. This is equivalent to there being no trace t' with $t' \sim_{-X_1,\dots,X_n} t$ with $t' \nvDash \varphi_{k-1}$. This is equivalent, by the IH, to there being no trace t' with $t' \sim_{-X_1,\dots,X_n}$ having $t' \notin L(A_{\varphi_{k-1}})$.
  This is equivalent to to there being no trace t' with $t' \sim_{-X_1,\dots,X_n} t$, we have that $t' \in L(A_{\neg\varphi_{k-1}})$. But, this is equivalent it is not the case that there is a trace t' with $t' \sim_{-X_1,\dots,X_n}$ and $t' \in L((A_{\neg\varphi_{k-1}})$. But this is the case iff it is not the case that $t \in L((A_{\neg\psi})_{\exists X_1,\dots,X_m})$, this is the case iff $t \in L(\overline{D((A_{\neg\varphi_{k-1}})_{\exists X_1,\dots,X_n})})$.

□

Theorem 35. *Computing the A_φ for a QLTL$_f$ formula φ takes $(k+2)$-EXPTIME.*

Proof. The proof is by induction on the alternation count.

- If the alternation count is 0, we have three base cases:
  - If the matrix is a LTL$_f$ formula, then trivially it takes 2EXPTIME, as we need one exponential for creating an NFA and one for determinisation.
  - If the formula is of the kind $\exists^*\varphi$, then we need one exponential for the NFA for φ, polynomial for the existential abstraction, and another exponential for the determinzation.
  - If the formula is of the kind $\forall^*\varphi$, then we need one exponential to create the NFA for $\neg\varphi$. Then, we need polynomial time for the existential abstraction, one exponential for determinizing and afterwards complementing is linear time. Thus, we overall have 2EXPTime.
- For the inductive step, we have to consider two cases:
  - If the formula is of the form $\exists^*\varphi_{k-1}$ then we need (k+1)-EXPTIME to create a DFA recognising φ_{k-1}. The existential abstraction NFA is polytime, however determinising it costs another exponential, yielding $(k+2)$-EXPTIME.
  - If the formula is of the form $\forall^*\varphi_{k-1}$ then we need (k+1)-EXPTIME to create a DFA recognising φ_{k-1}. Negating a DFA is polynomial, then existential abstraction produces an NFA and is too polynomial. Determinization takes another exponential, negation is polynomial, thus we overall take $(k+2)$-EXPTIME.

□

These allow us to show that we can do synthesis for arbitrary QLTL$_f$ specifications.

Theorem 15. *Synthesis for a QLTL$_f$-formula ψ with k alternations can be solved in $(k+2)$-EXPTIME.*

Proof. Assume w.l.o.g. that φ is in PNF; then we can compute an automaton that recognises φ in (k+2)-EXPTIME, by Theorems 34 and 35. We can then solve the game in polynomial time. □

F Supplementary figures

Overhead

To evaluate the overhead introduced by synthesis under unreliable inputs, we compared the runtime of our approach with the runtime required for separately synthesizing strategies for the main formula under full observability and the backup formula under partial observability.[4]

In Figure 3, we present the ratio of our algorithm's performance relative to synthesizing only the main formula, only the backup formula under partial observability, or the sum of these two. A ratio of 1 indicates

[4]This can be triggered using the command line option `-disregard=main/backup` in the testing script.

identical performance, while a ratio of 2 means our approach takes twice as long, and so forth. As shown in the figure, when compared with the sum of individual runtimes, our approach is at most 2.3 times slower, indicating a linear overhead. Furthermore, the mean and median ratios are both less than 1, with a standard deviation of 0.3.

We also provide a plot of the actual runtimes in Figure 4, using the same color scheme for direct, belief-state, and MSO techniques as in the other figures.

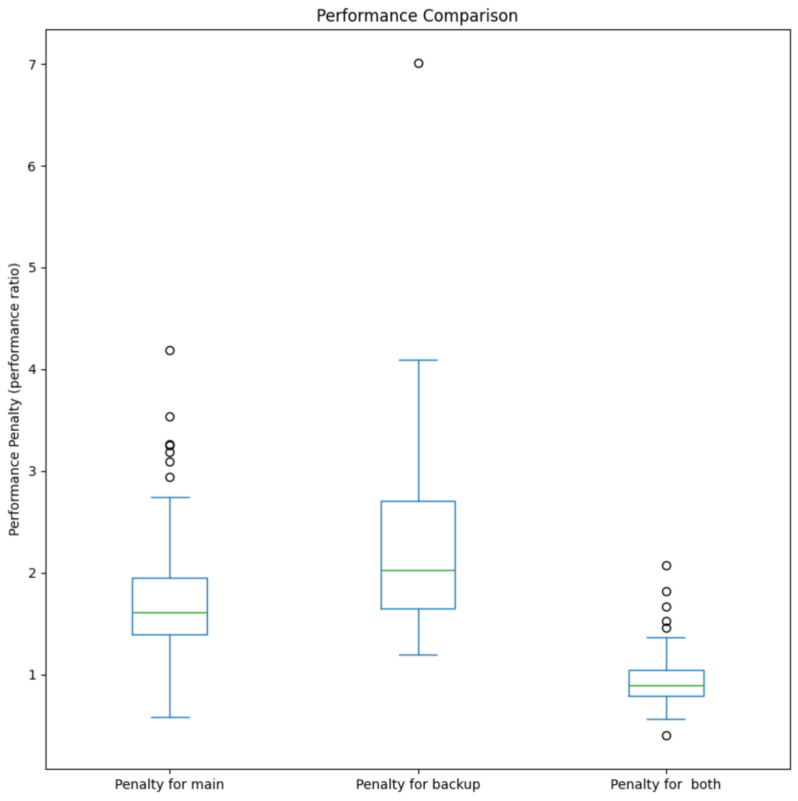


Figure 3: Boxplot of ratio of MSO solution for synthesis under unreliable input

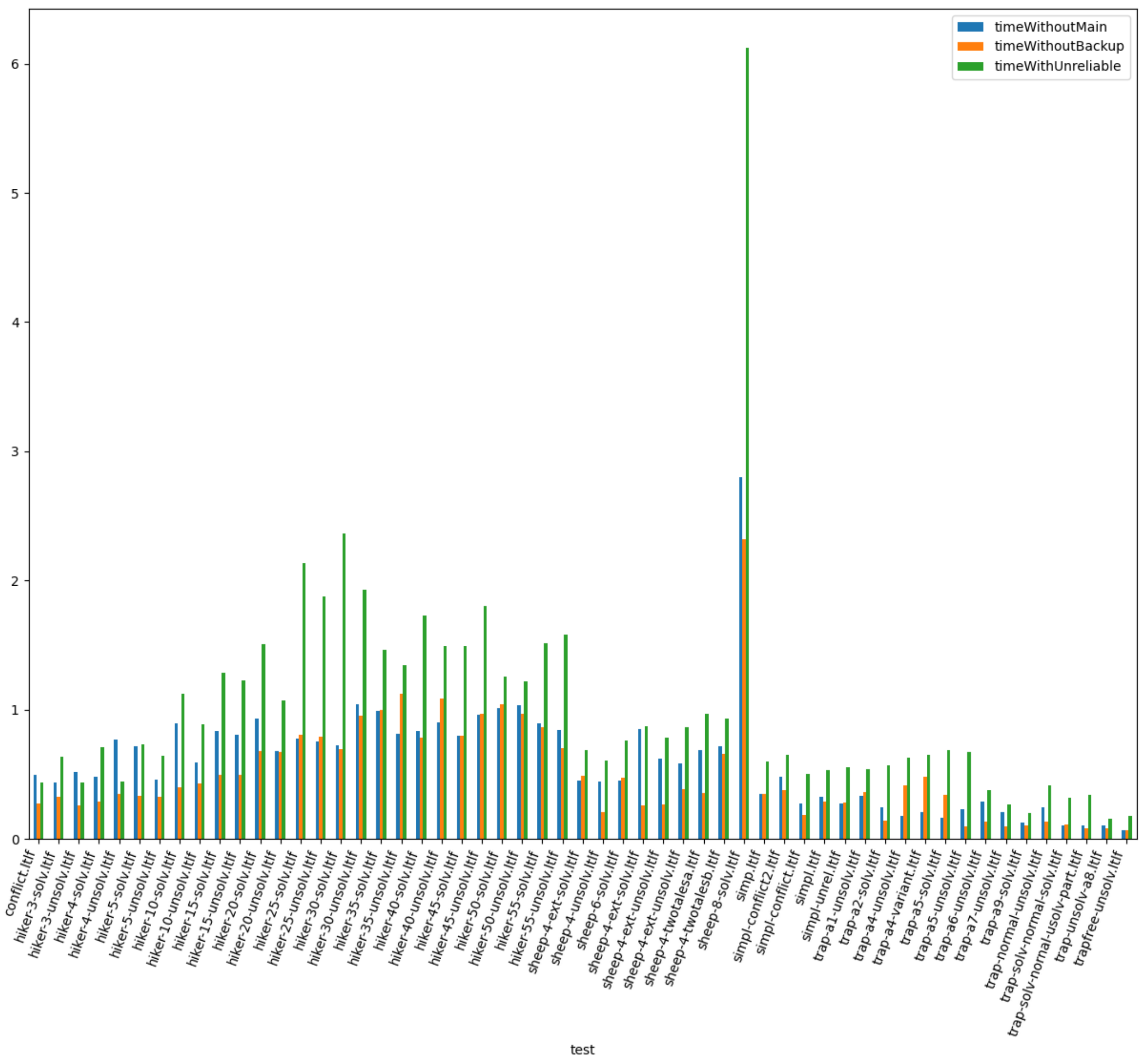


Figure 4: Overhead on tests