LTL_f Synthesis Under Unreliable Input

Christian Hagemeier[1], **Giuseppe De Giacomo**[1], **Mose 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 — http://github.com/whitemech/ltlf-synth-unrel-input-aaai2025

Extended version — http://arxiv.org/abs/2412.14728

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, "software does not necessarily conform neatly to probabilistic distributions, making it difficult to apply statistical models or predictions commonly used in other scientific disciplines" (Office of the National Cyber Director 2024). 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, 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.

[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.

2 Preliminaries

Definition 1. The language of LTL_f is defined by

$$\varphi := A \mid \neg\varphi \mid \varphi \wedge \psi \mid \bigcirc\varphi \mid \phi \,\mathsf{U}\, \psi$$

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

Other modalities, like Weak Next ($\bullet\varphi := \neg\bigcirc\neg\varphi$), 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 $1 \leq i \leq \mathsf{length}(t) = last$:

- $t, i \vDash A$ 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 $t, i \vDash \psi$
- $t, i \vDash \bigcirc\varphi$ iff $i < last$ and $t, i+1 \vDash \varphi$
- $t, i \vDash \varphi \,\mathsf{U}\, \psi$ iff for some j with $i \leq j \leq last$ 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 2013). 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.

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 $\mathsf{exp}_{\mathcal{V}}(\widehat{t})$ takes a trace $\widehat{t}$ over variables $\widehat{\mathcal{P}}$ (with $\mathcal{V} \cap \widehat{\mathcal{P}} = \varnothing$) and returns all traces t by setting variables in $\mathcal{V}$ in every possible way at all instants of $\widehat{t}$. 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 2013), 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 $\varphi_m \coloneqq \top$), 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 $\mathsf{disallow}_{\mathsf{i,j}}$ 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 $\mathsf{disallow}_{\mathsf{i,j}}$ 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 $\varphi_{env} \to \bigvee_{v \in R} \Diamond(pos(v))$, where $\bigvee_{v \in R} \Diamond(pos(v))$ 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 $\varphi_e \supset \Box(\mathsf{berry} \wedge \neg\mathsf{poison} \supset \mathsf{eat}) \wedge \Diamond(\mathsf{eot})$. And the backup specification is $\varphi_e \supset \Diamond(\mathsf{eot} \wedge \neg\mathsf{sick})$. 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)):

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

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 $\overline{\mathcal{D}((\mathcal{A}_{\neg\varphi})_{\exists U_1,\dots,U_n})}$.

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 $N_{\exists U_1,\dots,U_n} = \left(2^{\mathcal{X}} \times 2^{\mathcal{Y}}, S, \delta', s_0, F\right)$ by setting

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

Theorem 9. *Solving synthesis for the synchronous product of the* LTL_f*-automaton for* φ_m*,* A_{φ_m}*, and the automaton* $\overline{\mathcal{D}((\mathcal{A}_{\neg\varphi})_{\exists U_1,\dots,U_n})}$ *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]

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

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 $\forall X.\varphi := \neg\exists X.\neg\varphi$; 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* $\mathcal{X}_{unr} = \{U_1,\dots,U_n\}$ *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.*

[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.

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 $\varphi_m \wedge \forall U_1 \dots \forall U_n . \varphi_b$ 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, 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 $x \leq last$):

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

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* $\psi = \varphi_m \wedge \forall U_1 \dots \forall U_n . \varphi_b$*, 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.

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

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.

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 syn-

thesis 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 investigated by De Giacomo and Vardi (2016). While related, our problem is novel: it requires 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

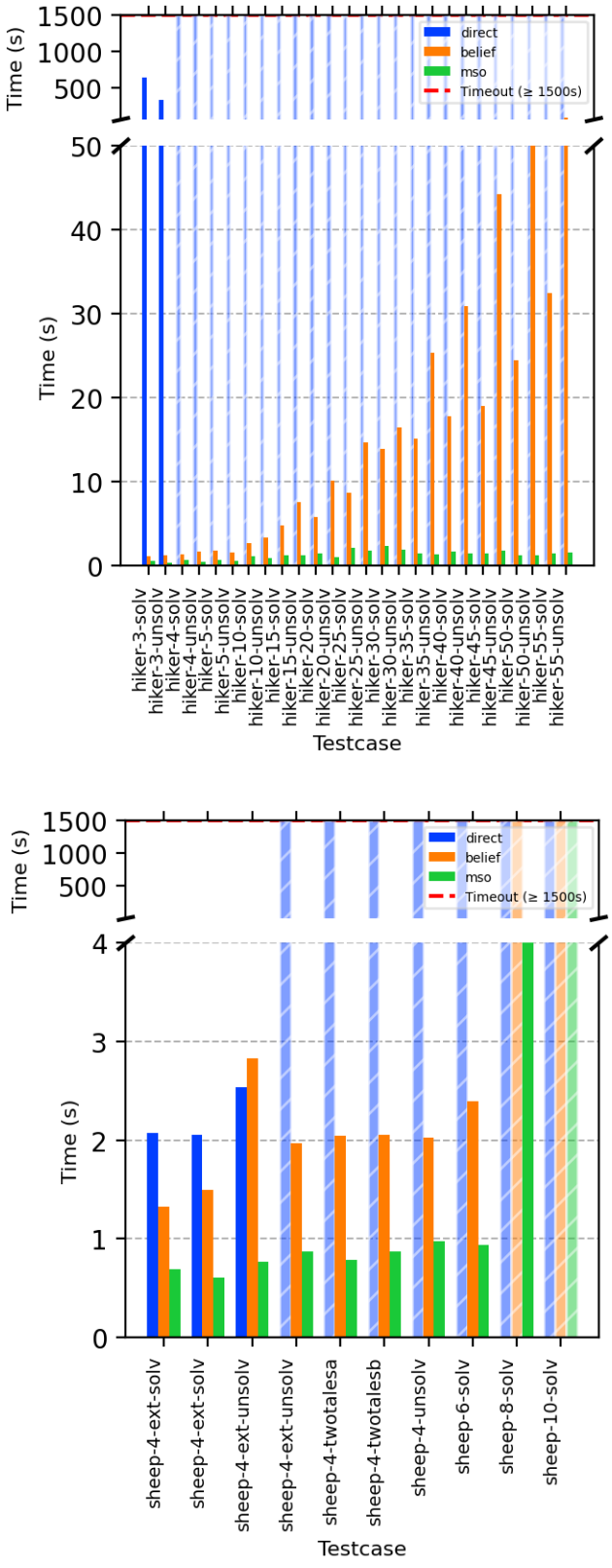

Figure 1: Runtime for the sheep and hiker examples.

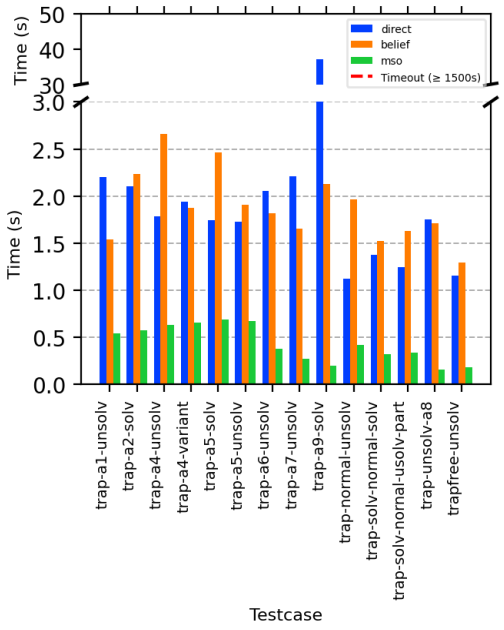

Figure 2: Runtime for the trap examples.

(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 thank Antonio Di Stasio, Shufang Zhu and Pian Yu for insightful discussions on the topic of this paper. We also thank Lucas M. Tabajara for providing the source code for LTL_f synthesis under partial observability. We would like to acknowledge the use of the University of Oxford Advanced Research Computing (ARC) facility in carrying out the experiments (http://dx.doi.org/10.5281/zenodo.22558). 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 *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, 788–795. AAAI Press. ISBN 9781577352815.

Bloem, R.; Chockler, H.; Ebrahimi, M.; and Strichman, O. 2019. Synthesizing Reactive Systems Using Robustness and Recovery Specifications. In *2019 Formal Methods in Computer Aided Design (FMCAD)*, 147–151.

Calvanese, D.; Giacomo, G. D.; and Vardi, M. Y. 2002. Reasoning about actions and planning in LTL action theories. In *Proceedings of the Eights International Conference on Principles of Knowledge Representation and Reasoning*, KR'02, 593–602. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 1558605541.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Strong planning in non-deterministic domains via model checking. In *Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems*, AIPS'98, 36–43. AAAI Press. ISBN 1577350529.

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 LTLf and LDLf goals. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, 4729–4735. AAAI Press. ISBN 9780999241127.

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. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, 854–860. AAAI Press. ISBN 9781577356332.

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.; Pnueli, A.; Shelah, S.; and Stavi, J. 1980. On the temporal analysis of fairness. In *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '80, 163–173. New York, NY, USA: Association for Computing Machinery. ISBN 0897910117.

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.

Office of the National Cyber Director. 2024. Back to the Building Blocks: A path towards secure and measurable software. Technical report. Available at: https://policycommons.net/artifacts/11346583/final-oncd-technical-report/.

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. In Raskin, J.; and Bresolin, D., eds., *Proceedings 11th International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2020, Brussels, Belgium, September 21-22, 2020*, volume 326 of *EPTCS*, 1–17.

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.