TOWARDS EXPLAINABLE AI: DIRECTED INFERENCE OF LINEAR TEMPORAL LOGIC CONSTRAINTS

BY

NOEL BRINDISE

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Aerospace Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

Adviser:

Professor Cédric Langbort

ABSTRACT

Many systems in robotics and beyond may be classified as mixed logical-dynamical (MLD) systems. These systems are subject to both logical constraints, which govern their safe operation and goals; and dynamical constraints, which describe their physical behavior. These time-dependent constraints can be described with linear temporal logic (LTL). In the case where the constraints are not known, their inference offers a type of explanation for their behavior.

Previous work has attempted to infer constraints for MLD systems by Bayesian methods, searching for optimally contrastive rules between "good" and "bad" system runs. However, due to a reliance on an unknown prior distribution, as well as a limited search space, these efforts are unable to recover all desired constraints.

We propose an alternative inference method called directed hypothesis space generation (DHSG). DHSG compares each system run and constructs a full hypothesis space of all conjunctions and disjunctions of the desired LTL formula types. In simulation, DHSG recovered a full hypothesis space for each test case. However, due to a comparatively high computational demand, it also exhibited run times which increased significantly with state space complexity. The computational load was lightened by limiting the length of inferred formulas, at the cost of hypothesis space completeness. However, the adjustable computation time of the Bayesian approach means that it retains an advantage under some use cases.

Finally, for scenarios in which neither the LTL rules are known, nor the state-space regions they govern, DHSG has potential to construct the unknown regions. This approach would give a basis on which to perform further inference. Region construction would apply to lesser-understood systems and presents a topic for future work.

For my brothers, Dave and Scott

ACKNOWLEDGMENTS

I would first like to thank Prof. Langbort for his guidance on this thesis and much more. Many thanks are also due to my parents, for their encouragement; Tove Kopperstad, for her open ears; and Markus Pfahler, for all his technical and moral support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DHSG	Directed Hypothesis Space Generation
LTL	Linear Temporal Logic
MILP	Mixed Integer Linear Programming
MLD	Mixed Logical Dynamical
xAI	Explainable Artificial Intelligence

CHAPTER 1

PRELIMINARIES

1.1 Motivation: Explainable artificial intelligence

1.1.1 Introduction to explainable AI

Artificial intelligence is becoming a shaping force in the human experience, and countless applications reach into Aerospace. AI route planners provide military pilots with flight plans and reroutes based on real-time threat, weather, and efficiency data [3], while AI situational awareness software filters situational data for space and air military applications [4]. AI shows significant promise in aircraft adaptive control, able to reassess a damaged aircraft during flight and adjust control inputs accordingly [5]. However, as AI becomes increasingly prevalent, the potential for human-machine disconnect grows.

Artificial intelligence demands explanation in a wide variety of aerospace applications. A pilot following a rerouted flight plan, or trusting adaptive control to take the yoke after aircraft damage, will want to understand the "what"s and "why"s of the decisions a computer is making on his behalf. That need for explanation is not only a practical one, but also a moral one—when a person entrusts their life to a system, they should be informed about the choices that system makes for them. The looming question is then: what constitutes a useful explanation, and how can we construct it?

The emerging field of explainable AI (xAI) aims to answer that question, developing explanations for AI decisions which would otherwise be unintelligible to a person. However, while xAI has made large strides in AI interpretation, there remains significant work to be done to make drones, robots, military aircraft, and countless other systems intelligible to their human users.

It is natural to identify three major stages in the explanation process (Figure 1.1). The first stage is model construction. Since humans interpret cause and effect based on a personal context of assumptions and background knowledge [6], a good explanation must provide an accurate, optimally relevant context. To provide context on an AI, though, an explainer must learn the context in the first place. The explainer needs to build its own model of the AI, from which it can mine the relevant details. This stage is where most xAI research is now focused.

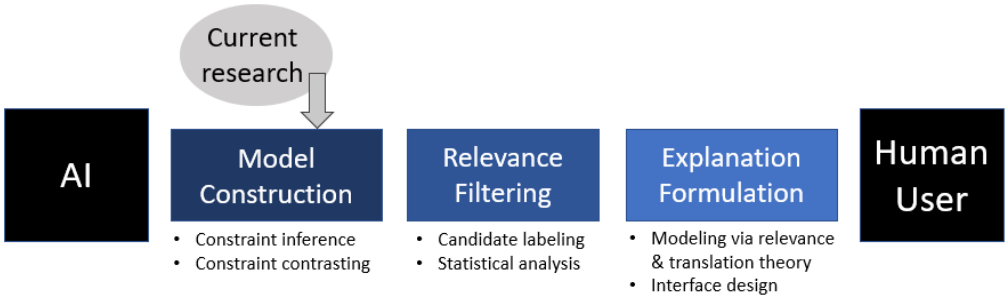

Figure 1.1: 3-stage explanation and the research focus of three fields. Current xAI research has an overwhelming emphasis on model construction.

Building on current work, the inference method described in the following chapters aims to make a contribution to that first explanatory step, model construction. We aim to improve model building via constraint inference, specifically the inference of linear temporal logic (LTL) constraints on mixed logical-dynamical (MLD) systems.

1.1.2 MLD system LTL constraint inference as explanation

Constraints on a system are useful for explanation because they are essentially the rules that a system must follow. If we know the constraints, we can describe the system's decisions in terms of which rules they are following, providing a backbone for explanation. In our case, we observe runs of an MLD system and attempt to recover a set of LTL formulas that those runs are consistently satisfying. If a set of system runs, or "traces", is known to be acceptable—i.e., if the runs are labeled as "good" traces—it makes sense to characterize their goodness based on which LTL formulas appear to describe all of them.

MLD systems operate based on both dynamical and logical rules; one example is a rock-collecting lunar rover. A rover has some maximum turning radius (a dynamical rule), while the charge in its batteries must remain above some threshold (a logical rule for operations). Finally, the rover needs to perform its task, necessitating a set of logical process rules: the rover must collect a sample from Region 1 before unloading a sample in Region 2, all while avoiding Region 3, where the ground is too steep. The mixed logical-dynamical system of the rover can be expressed as the product of a weighted transition system, $\mathcal{T}$ and a Büchi automaton, $\mathcal{B}$ [7]. The product of these two tuples, $\mathcal{B} \times \mathcal{T}$ is called the product automaton. The system of the product automaton is very useful because it allows us to impose linear temporal logic constraints on a process. Any LTL formula φ can be expressed in terms of atomic propositions (here, Boolean labels on each system state), as well as Boolean and temporal operators, allowing us to specify orders and time dependencies of states. By checking which φ are satisfied, we can determine which of a set of dynamical and logical rules govern the system.

The remaining preliminaries further describe MLD and LTL, providing formal definitions and establishing notation.

1.2 Mixed logical dynamical systems and the product automaton

1.2.1 Definition and syntax

Before a discussion of the constraint inference methodology, it is necessary to understand the form of the constraints we seek to infer. This thesis considers mixed logical dynamical (MLD) systems, as proposed in [8]. MLD systems offer a way to combine system dynamics, logical rules, and operating constraint into a single model. Here,

- the dynamics are the physical laws that govern the system's motion,
- the logical rules are the rules which govern the process(es) the system is meant to carry out, and

- the operating constraints are any imposed limits on the system's operational range.

We tackle constraint inference for the model of an MLD system described in [1], a combination of a weighted transition system and a Büchi automaton known as a "product" automaton.

The weighted transition system deals with the physical states and transitions of the system, and is defined by the tuple

$$\mathcal{T} := (Q, q_0, R, \Pi, \mathcal{L}, w)$$

where

- Q is a finite set of states
- $q_0 \in Q$ is an initial state
- $R \subseteq Q \times Q$ is a transition relation
- Π is a set of atomic propositions
- $\mathcal{L} : Q \to 2^{\Pi}$ is a labeling function
- $w : R \to \mathbb{R}_{>0}$ is a weight function

A run of this transition system thus begins with some state q_0 and then transitions to subsequent states according to R (i.e., for a transition from the state q_i at time t_i to the state q_{i+1} at time $t_{i+1} = t_i + w(q_i, q_{i+1})$, it must be true that $(q_i, q_{i+1}) \in R$. The resulting run $r_{\mathcal{T}}$ can be expressed as a sequence of states $q_0, q_1, q_2...$ (where $q_0 \in Q_0$ and all $q_i \in Q$).

The labeling function $\mathcal{L}(\Pi)$ takes this sequence of states and generates the associated "word" $\mathcal{L}(q_0), \mathcal{L}(q_1), \mathcal{L}_(q_2)...$, where $\mathcal{L}(q_i)$ gives the subset of atomic propositions satisfied when the system is in state q_i. These atomic propositions α_i are boolean variables which may describe process- or safety-relevant characteristics of each state. For instance, there may be some α_1 which is true if the state q_i allows access to a data-upload port for the robot, and another α_2 which is true if the state q_i allows access to a charging station.

Encoding the process-related logical rules to be satisfied by the system, the Büchi automaton is defined as the tuple

$$\mathcal{B} := (S, S_0, \Sigma, \delta, F)$$

where

- S is a finite set of states
- $S_0 \subseteq S$ is a set of initial sets
- Σ is an input alphabet
- $\delta \subseteq S \times \Sigma \times S$ is a non-deterministic transition relation
- $F \subseteq S$ is a set of accepting final states

The dynamics we consider for this MLD encoding are linear and subject to linear mixed-integer inequalities, where mixed-integer refers to a mix of continuous and binary variables.

1.2.2 State-space region specification for constraint formulation

Constraints on an MLD system can be expressed in part using bounds on continuous variables. In this way, constraints on a system may be thought of as sets of bounds on continuous variables, where any given set of bounds may be enforced or relaxed under the conditions specified by the constraint. For example, a constraint requiring that a system always remain in a given region will prescribe that the associated bounds are always enforced; a constraint specifying a final, "goal" state may only enforce the "goal" bounds on the final time step, i.e., it enforces that the system state must be within those bounds only at the final time.

Using this approach, all constraints on a system can be expressed as some set(s) of bounds on one or more of the system variables. A set of bounds can be expressed as a region of the system's state space, in particular using linear inequalities; this approach is taken by the formulation in [8]. Each individual inequality specifies a halfspace, where the intersection of all the specified halfspaces is the region to be governed by the associated constraint.

For our approach to constraint inference, we use a simplified formulation for states. Because we consider only binary labels—specifying the current state with a subset of the possible labels—the region of state space we occupy can be determined by checking which binary labels are applied at the desired

time step. In this way, we are disregarding (for now) the distinction between the physical states of the Büchi automaton and process states, choosing to perform inference purely over the vocabulary of labels.

In all, the regions of state space, represented as labels in our application, act as arguments in time-dependent system constraints. A natural way to encode such time- and logic-dependent constraints on an MLD system is linear temporal logic.

1.3 Linear temporal logic

1.3.1 Definition and syntax

Linear temporal logic (LTL) is an extension of propositional logic that integrates operators which deal with discrete time, making it a useful logic for enforcing processes, time dependencies, and long-term behavior in path planning. All LTL formulas may be expressed in terms of

- one or more **atomic propositions**, or boolean variables. In our case, these are the labels contained in the system vocabulary.
- the logical operators $\neg$ (not) and $\vee$ (or)
- the unary temporal operator $\mathbf{X}$ (next), which requires that its argument be true for the next time step
- the binary temporal operator $\mathbf{U}$ (until), which requires that its first argument be true until the time step at which its second argument occurs

A single atomic proposition, as well as any instantiation of the operators with LTL formulas, constitute an LTL formula. From these basic operators, additional operators can be constructed to streamline the syntax. Some common additional operators, defined in Table 1.1, are included in the scenario described below.

In the case of a robot operating in some state space, a path can be planned to fit LTL constraints which prescribe, on the most general level, the following:

- Safety: the robot should operate always within the bounds of a "safe" region of the state space, e.g., a table-top robot should never drive beyond the dimensions of the table, or it will fall off. Likewise, the robot's internal state should remain within its safe operational ranges, e.g., its processor should not exceed a given operational temperature.

- Objectives: the robot should consistently meet process objectives. These may be objectives to remain operational, e.g., it must always revisit a charging state before running out of battery. They may also be objectives related to the robot's operational purpose, e.g., a picture-taking robot must always revisit a picture-uploading state after collecting some amount of photograph data.

To fulfill these types of constraints, the system state may incorporate both physical and process-related variables. Whether or not the system is in a certain state is then assessed by the product automaton's labeling function. Again, "regions" of state space refers to sets of bounds on one or more of the state space variables, defined as described in 1.2.2. The variables themselves may be any subset of the set of product automaton states.

1.3.2 Formula types

We do not consider all LTL formula types, but a selection of the LTL vocabulary which is most applicable to task-performing robot-type systems. For these systems, the operators of interest are those that express the aforementioned safety- and objective-type constraints.

When discussing LTL formulas, we refer to a formula consisting of a single template from our set of predefined templates as **atomic**. Atomic formulas do not consist of conjunctions or disjunctions of other stand-alone template instantiations; in a sense, they are our smallest meaningful LTL units. In our application, we specifically consider the formula templates given in Table 1.2.

1.3.3 Satisfaction checking

Having defined the LTL formulas of interest, it is necessary to establish a method of checking formula satisfaction of LTL hypotheses on the traces of

Table 1.1: LTL temporal operators.

Operator	**Syntax**	**Meaning**
Global	$\mathbf{G}\varphi_1$	φ_1 is always true.
Eventual	$\mathbf{F}\varphi_1$	φ_1 eventually occurs.
Next	$\mathbf{X}\varphi_1$	φ_1 must occur at the next time step.
Until	$\varphi_1\mathbf{U}\varphi_2$	φ_1 remains true until φ_2 occurs (and φ_2 must occur)
Release	$\varphi_1\mathbf{R}\varphi_2$	φ_2 must remain true until (and including) the time step when φ_1 becomes true. If φ_1 is never true, φ_2 must always be true.
Weak until	$\varphi_1\mathbf{W}\varphi_2$	φ_1 must be true at least until φ_2 becomes true. If φ_2 is never true, then φ_1 must always be true.

our MLD system. For LTL formulas of general mixed-integer linear constraints, the formulation in [7] provides a method for satisfaction checking. In our case, since state space regions are treated as atomic proposition labels, we are able to perform LTL satisfaction checking using simple recursion over time steps. For example, a formula with the "eventually" operator **F** can be checked by iterating through time steps and ensuring that a boolean *value* remains true at every step, as follows:

Check Fp

For time $t \leq t_{\text{final}}$:

$$value = \textbf{check}((p \text{ true at } t) \text{ OR } (p \text{ true at } t+1))$$

For time $t = t_{\text{final}}$:

$$value = \textbf{check}(p \text{ true at } t)$$

Return *value*

Similar recursive methods are able to address the other LTL templates of interest. The entire script used in simulation for basic LTL formula checking was developed in [2] and is available on GitHub [9].

Table 1.2: Atomic LTL formula templates.

Template	**Syntax**	**Meaning**
Global	$\mathbf{G}p_1$	p_1 is always true.
Eventual	$\mathbf{F}p_1$	p_1 eventually occurs.
Until	$p_1\mathbf{U}p_2$	p_1 remains true until p_2 occurs (and p_2 must occur)
Response	$\mathbf{G}(p_1 \rightarrow \mathbf{XF}p_2)$	If p_1 occurs, p_2 eventually follows.
Stability	$\mathbf{FG}p_1 \wedge \mathbf{G}(p_1 \rightarrow (p_1\mathbf{WG}\neg p_1))$	p_1 eventually occurs and stays true forever.
At most once	$\mathbf{G}(p_1 \rightarrow (p_1\mathbf{WG}\neg p_1))$	p_1 may only be true on one contiguous time interval.
Sometime before	$(p_2 \wedge p_1)\mathbf{R}(\neg p_1)$	If p_1 occurs, p_2 must have occurred in the past.

CHAPTER 2

PROBLEM STATEMENT AND OVERVIEW OF APPROACH

2.1 Goal of inference

The problem we consider involves an MLD system subject to some set of LTL requirements and functioning in a finite domain. One concrete example of such a system is the robot presented in [1]. This robot operates on a tabletop layout of roads and stations, where the various stations in the layout function as battery-charging, data collection, and data upload sites. The tabletop layout, with stations labeled $P1 - P5$, is reproduced in Figure 2.1.

The robot aims to carry out a data collection-and-upload process while ensuring that it (1) remains charged, (2) penalizes travel time, and (3) stays within the prescribed roadways and stations. The data collection process, as well as (1) and (3), are achieved via enforcement of appropriate LTL formulas, such as those shown in Table 2.1.

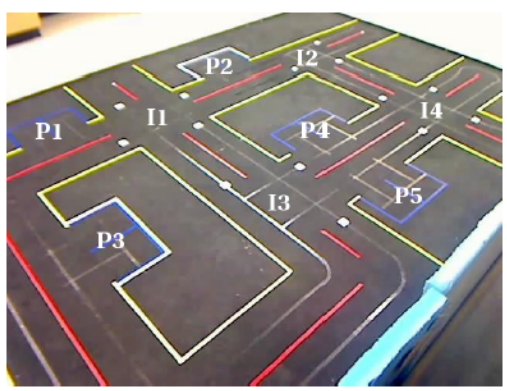

Figure 2.1: Table top with roads and parking stations labeled (reproduced from [1]).

In order for a run of the robot to be considered acceptable, every one of those LTL formulas must be satisfied by that run. In other words, the success

Table 2.1: Examples of rules governing tabletop robot in [1].

LTL formula	**Interpretation**
$\mathbf{GF}(P1) \wedge \mathbf{GF}(P4) \wedge \mathbf{GF}(P5)$	The robot keeps visiting each data gather location
$\mathbf{G}((P1 \vee P4 \vee P5) \implies \mathbf{X}(\neg(P1 \vee P4 \vee P5)\mathcal{U}(P2 \vee P3)))$	Whenever the robot gathers data, it uploads it before doing another data gather.
$\mathbf{G}((P2 \vee P3) \implies \mathbf{X}(\neg(P2 \vee P3)\mathcal{U}(P1 \vee P4 \vee P5)))$	Whenever the robot uploads data, it does not visit an upload location again before gathering new data.

of a run in terms of its requirements and goals can be described by the run's satisfaction (or violation) of a set of LTL formulas.

Our particular problem considers the scenario in which we are able to observe the runs of such a system—in this example, we are able to watch the robot as it performs—but we do not know the rules under which the system operates. More specifically, we assume:

1. **Unknown system constraints:** we do not know, a priori, the LTL constraints on the system.

2. **Labeled runs:** each run we observe is labeled, i.e., we are told whether or not each run "followed the rules" and was considered acceptable.

3. **Collectible run data:** we are able to collect the run data in terms of some set of state space variables at each t, where we assume discrete time.

By assuming collectible run data, we disregard scenarios in which necessary state data is missing, that is, we are able to observe all states governed by the unknown LTL rules.

Given these assumptions, our goal is to **infer the LTL rules satisfied by the successful runs of the system**. In particular, if we can generate candidate formulas that are both satisfied by "good" runs and violated by "bad" runs, these candidates provide potential explanation for the distinction between "good" and "bad". Such a distinction provides a description of the system's behavior and may be used as a starting point for explanation.

2.2 Problem setup: Hypothesis space and requirements

This section motivates and describes the framework for the problem solution, which takes the form of a "hypothesis space" satisfying a set of requirements. Sections 2.2.1 and 2.2.2 describe the hypothesis space and requirements qualitatively, while Sections 2.2.4 and 2.2.5 define them formally.

2.2.1 Hypothesis space overview

The set of candidate formulas generated by the inference process constitute a "hypothesis space," a list of rules that correctly classify the runs and may thus replicate the original, unknown rules that govern the system. Because every individual rule candidate must be satisfied by good traces (Rule 1 **and** Rule 2 **and** Rule 3 must be satisfied...), the hypothesis space can be represented as a list of conjunctions, as in Figure 2.2. From here, our requirements for inference can be expressed as requirements on this hypothesis space, describing the necessary qualities for the conjunctive formula.

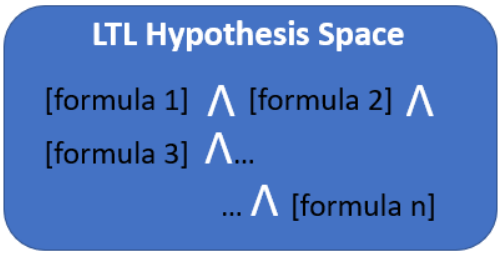

Figure 2.2: The hypothesis space can be represented as the conjunction of a list of rule candidates.

2.2.2 Requirements overview and justification

The hypothesis space of LTL formulas is required to possess three qualities: *validity, strictness*, and *conciseness*. This section gives a qualitative description and the relevance of each quality.

Validity: We seek to infer formulas which are satisfied on all "good" traces. A candidate formula which is satisfied on these traces will be denoted as *valid*. If a formula is not valid on all good traces, it would suggest that that formula must not be a true requirement for goodness; otherwise, traces

not satisfying the formula would not be labeled "good" in the first place. Thus, for a formula to be a candidate in the hypothesis space, we require validity.

Strictness: We seek to hypothesize all constraints which are *as strict as possible*. Since the hypothesis space can be expressed as the conjunction of a set of valid formulas, a fully-strict hypothesis space would not omit any valid formula which would strengthen the conjunction. Constraint strictness is critical because any loss of strictness equates to a failure to fully characterize the trajectories' shared qualities. That is, unnecessary loss of strictness means that the trajectories are in fact following some additional LTL rule(s) that our hypothesized constraints all fail to capture. However, since this approach considers only a "relevant" subset of LTL templates (i.e., LTL operators), the results of inference will remain limited to formulas constructed of instantiations of that subset. Given this qualification, we require strictness over the set of formula constructable with the chosen set of templates.

Conciseness: Finally, we require that the hypothesis space be as concise as possible. Conciseness means that the hypothesis space does not include any formulas which are redundant. This is equivalent to requiring that, for the conjuncts in the hypothesis space, a deletion of any conjunct would result in a less strict conjunction. Conciseness is important because it guarantees the smallest possible fully-valid and fully-strict hypothesis space.

2.2.3 Explanation of Notation

Table 2.2 establishes vocabulary for the proofs and definitions that follow.

Table 2.2: Explanations of Notation.

Representation	**Meaning**
Φ	The full hypothesis space, represented as a single 'superformula'
φ	An LTL formula
ϕ	An atomic LTL formula (an instantiation of a basic LTL template)
π	A trace (i.e., a run of the system, consisting of a list of states)
Π_A	The full set of good traces
$\varphi_i(\pi_k)$ true	LTL formula φ_i is satisfied on trace π_k
$\phi' \in \{\varphi\}$	ϕ' is one of the arguments of the disjunct φ (i.e., $\varphi = \phi_1 \vee ... \vee \phi'$)
$\varphi' \in \{\Phi\}$	φ' is one of the arguments of the conjunct Φ (i.e., $\Phi = \varphi_1 \wedge ... \wedge \varphi'$)

2.2.4 Formalized hypothesis space

In Section 2.2.1, we claim that the hypothesis space of all formulas can be expressed as a conjunction of individual formulas. In fact, the hypothesis that we infer can be expressed as a conjunction of disjunctions of atomic formulas. Formally, the hypothesis space can be expressed as in (2.1).

Proposition. **Form of Φ.**

Any valid, strict, and concise hypothesis space can be expressed in the form

$$\Phi := \bigwedge_{i=1}^{I} \varphi_i \quad \text{where} \quad \varphi_i = \bigvee_{j=1}^{J} \phi_j \tag{2.1}$$

Note here that each φ_i may have any finite number of disjuncts, and thus each J is distinct.

Proof. **Sufficiency of form for Φ.**

We require a form which can represent any possible conjunctions and disjunctions of atomic formulas ϕ. Suppose, towards contradiction, that the structure given in (2.1) is not sufficient. This would imply that, for sufficiency, either

1. Φ must actually include one or more disjunctions

2. φ_i must actually include one or more conjunctions

If (1) is true, then we have a Φ of the form

$$\Phi := \varphi_1 \wedge \varphi_2 \wedge ... \vee \varphi_{n-2} \wedge \varphi_{n-1} \vee \varphi_n \tag{2.2}$$

where there can be any nonzero number of disjunctions, and conjunctions of ϕ_j are possible within each φ_i. For example, take a Φ which includes both an outer disjunction, as in violation condition (1); and an inner conjunction, as in violation condition (2):

$$\Phi = (\phi_1 \vee \phi_2) \vee (\phi_3 \wedge \phi_4) \tag{2.3}$$

However, by the distributive property,

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

Letting $a = d \vee f$,

$$(d \vee f) \wedge (b \wedge c) = ((d \vee f) \vee b) \wedge ((d \vee f) \vee c)$$

Which, by the associative property, is equivalent to

$$(d \vee f \vee b) \wedge (d \vee f \vee c)$$

So any "outer" disjunctions of grouped conjunctions or disjunctions may be simplified, and our example Φ becomes

$$\Phi = (\phi_1 \vee \phi_2 \vee \phi_3) \wedge (\phi_1 \vee \phi_2 \vee \phi_4) \tag{2.4}$$

We can then define each grouping of disjunctions as a φ_i:

$$\Phi = \varphi_1 \wedge \varphi_2 \quad \text{where} \quad \varphi_1 = \phi_1 \vee \phi_2 \vee \phi_3,\ \varphi_2 = \phi_1 \vee \phi_2 \vee \phi_4 \tag{2.5}$$

A similar process can deconstruct any formula of the (likewise unacceptable) form

$$\Phi = (a \wedge b) \vee (c \wedge d) \tag{2.6}$$

By distributive property,

$$
\begin{aligned}
(a \wedge b) \vee (c \wedge d) &= ((a \wedge b) \vee c) \wedge ((a \wedge b) \vee d) \\
&= ((a \vee b) \wedge (a \vee c)) \wedge ((a \vee b) \wedge (a \vee d))
\end{aligned}
$$

which, by associative property,

$$
\begin{aligned}
&= (a \vee b) \wedge (a \vee c) \wedge (a \vee b) \wedge (a \vee d) \\
&= (a \vee b) \wedge (a \vee c) \wedge (a \vee d)
\end{aligned}
$$

Therefore, we finally have

$$\Phi = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \quad \text{where} \quad \varphi_1 = (a \vee b),\ \varphi_2 = (a \vee c),\ \varphi_3 = (a \vee d) \tag{2.7}$$

This means that, for any Φ constructed of conjunctions and disjunctions of atomic formulas, the Φ may always be reduced to the form

$$\Phi := \bigwedge_{i=1}^{I} \left(\bigvee_{j=1}^{J} \phi_j \right) \tag{2.8}$$

□

2.2.5 Formalized requirements

We define validity, strictness, and conciseness from Section 2.2.2 as follows:

Definition. **Validity.**

The LTL "superformula" Φ of the hypothesis space must be valid for the full set of good traces Π_A. Thus, Φ is valid if and only if it is satisfied on all traces π_k in Π_A. First, since Φ is a conjunction of φ_i, i.e.,

$$\Phi = \bigwedge_{i=1}^{I} \varphi_i$$

we can say *every individual* φ_i in Φ must be satisfied on every trace, since

$$\exists(\varphi_i \in \{\Phi\}, \pi_k \in \Pi_A) \quad \text{s.t.} \quad \varphi_i(\pi_k) \text{ false} \Longrightarrow \Phi(\pi_k) \text{ false} \tag{2.9}$$

by the definition of conjunction. Now, for each φ_i to be satisfied, we note

that φ_i is a disjunction of ϕ_j, so

$$\varphi_i(\pi_k) \text{ true} \iff \exists \phi'_j \in \{\varphi_i\} \quad \text{s.t.} \quad \phi'_j(\pi_k) \text{ true} \tag{2.10}$$

In other words, for each φ_i, at least one ϕ'_j among the $\phi_j \in \{\varphi_i\}$ must be satisfied on each trace $\pi_1, ...\pi_n$ in order for φ_i to be satisfied by all traces.

Definition. **Strictness.**

For Φ to be as strict as possible, we require that there does *not* exist some other Φ' such that all of the following are true:

1. Both Φ and Φ' are valid for all traces
2. Φ' satisfied $\implies \Phi$ satisfied
3. Φ satisfied $\not\implies \Phi'$ satisfied

(2) and (3) would mean that Φ' is stricter than Φ, i.e., if Φ' is satisfied, the less-strict Φ must also be, but Φ satisfied does not necessarily require that Φ' is satisfied.

More formally, take

$$\Phi := \bigwedge_{i=1}^{I} \varphi_i \quad \text{and} \quad \Phi' := \left(\bigwedge_{i=1}^{I} \varphi_i\right) \wedge \left(\bigwedge_{i'=1}^{I'} \varphi_{i'}\right) \tag{2.11}$$

Since $\Phi \wedge \Phi'$ is a conjunction of conjunctions, we have that

$$\left(\bigwedge_{i=1}^{I} \varphi_i\right) \wedge \left(\bigwedge_{i'=1}^{I'} \varphi_{i'}\right) \text{ true} \implies \left(\bigwedge_{i=1}^{I} \varphi_i\right) \text{ true} \tag{2.12}$$

but that, particularly in the case that some $\varphi_{i'} \in \{\bigwedge_{i'=1}^{I'} \varphi_{i'}\}$ is not true,

$$\bigwedge_{i=1}^{I} \varphi_i \text{ true} \not\implies \left(\bigwedge_{i=1}^{I} \varphi_i\right) \wedge \left(\bigwedge_{i'=1}^{I'} \varphi_{i'}\right) \text{ true} \tag{2.13}$$

Thus, for strictness, we require a Φ which is a conjunction of, at minimum, *every* φ'_i valid on all good traces such that φ'_i is not logically implied by another valid φ_i in Φ. (Additional φ'_i may be included without loss of strictness, provided they are valid)

To characterize all such φ_i', recall first the form of all φ_i:

$$\varphi_i := \bigvee_{j=1}^{J} \phi_j \tag{2.14}$$

Since φ_i is a disjunction of ϕ_j, it follows that for some

$$\varphi_i = \bigvee_{j=1}^{J} \phi_j \quad \text{and} \quad \varphi_i' = \left(\bigvee_{j=1}^{J} \phi_j\right) \vee \left(\bigvee_{j'=1}^{J'} \phi_{j'}\right) \tag{2.15}$$

i.e., a φ_i whose set of arguments are a subset of the arguments of φ_i', we have

$$\varphi_i \text{ satisfied} \Longrightarrow \varphi_i' \text{ satisfied} \tag{2.16}$$

by the definition of disjunction. Thus, $\varphi_i \wedge \varphi_i'$ is no stricter than φ_i alone. However, note that

$$\varphi_i' \text{ satisfied} \not\Longrightarrow \varphi_i \text{ satisfied} \tag{2.17}$$

so φ_i' is not as strict as $\varphi_i' \wedge \varphi_i$. This shows that, to guarantee maximal strictness of Φ, we include *at minimum* all φ_i such that

- φ_i is satisfied on all traces
- the removal of any of φ_i's disjuncts ϕ_j would result in φ_i becoming invalid on at least one trace in Π_A

Again, note here that the inclusion of additional φ which do not meet the second criterion would not lead to a decrease in strictness. However, their inclusion does violate the final requirement for Φ, conciseness.

Definition. **Conciseness.**

For Φ to be concise, we now prohibit the inclusion of any "redundant" φ_i in the conjunction. That is, for each conjunction member φ_i, there must *not* exist another member φ_i' such that

$$\varphi_i \text{ satisfied} \iff \varphi_i' \text{ satisfied} \tag{2.18}$$

Once again, such a prohibition does not cause loss of strictness of the con-

junction. Consider our scenario:

$$\left(\bigwedge_{a=1}^{A} \varphi_a\right) \wedge (\varphi_i \wedge \varphi_i') \quad \text{where} \quad \varphi_i \text{ satisfied} \Longrightarrow \varphi_i' \text{ satisfied} \tag{2.19}$$

Here, it follows from (φ_i satisfied $\Longrightarrow \varphi_i'$ satisfied) that

$$\varphi_i \text{ satisfied} \Longrightarrow (\varphi_i \wedge \varphi_i') \text{ satisfied} \tag{2.20}$$

Thus Φ is valid if and only if φ_i is valid, and its behavior is logically the same whether or not φ_i' is included in the conjunction. This is equivalent to stating that strictness is not affected by the removal of φ_i'.

Now we aim to characterize the redundant φ_i'. Since all φ are disjunctions, we come to the same conclusion as in Eqns. 2.15 through 2.17; that is, we can see that φ_i implies φ_i' whenever the set of the elements ϕ_j of φ_i are a subset of the set of elements in φ_i':

$$\{\varphi_i\} \subset \{\varphi_i'\} \Longrightarrow (\varphi_i \Longrightarrow \varphi_i') \tag{2.21}$$

Conciseness requires that all φ_i' which are redundant as described in (2.21) are omitted from Φ.

CHAPTER 3

ALGORITHM FOR CONSTRUCTION OF HYPOTHESIS SPACE

3.1 Algorithm for Directed Hypothesis Space Generation

We aim to build a hypothesis space of the form in (2.2.4) which satisfies all three criteria of validity, strictness, and conciseness. Section 3.1 outlines an algorithm which accomplishes this, and Section 3.2 provides a verification for the result.

3.1.1 Approach

In contrast with previous inference efforts which generate random initial guesses for hypotheses and then check them against the traces, such as in [2], the algorithm outlined here first infers the atomic formulas satisfied on each trace and uses them to "directly" generate the full space of valid formulas. Considering this distinction, we refer to the new algorithm as "directed," as opposed to random, hypothesis space generation (DHSG).

DHSG relies on the basic principle that every good trace must satisfy every rule. If an atomic formula is satisfied on every good trace, that formula is valid by itself and may be included in the hypothesis space. If the atomic formula ϕ_1 is not satisfied on every good trace, there are two possibilities:

- ϕ_1 is partially descriptive of the good traces, and can be included in a disjunction with other partially-descriptive formulas ϕ_2, ϕ_3, etc., such that the full disjunction is valid on all traces. We refer to the construction of such a disjunction as formula **reconciliation** over the traces.

- ϕ_1 is not part of any reconciled formula and may be discarded entirely.

At first glance, it appears that ϕ_j can always be reconciled with other atomic formulas to create a fully-valid disjunction. It is indeed true that a valid hypothesis space could be generated simply by proposing the disjunction of all atomic formula instantiations. However, this hypothesis space would not be full, in the sense that it would lack stricter valid candidates. Indeed, when the requirements of strictness and conciseness are applied, the possibilities for reconciliation of ϕ_j become far more limited. We are only able to propose disjunctions of ϕ_j which are not redundant or excessively relaxed. To accomplish this, the construction of candidate disjunctions φ_i follows the algorithm in Section 3.1.2.

3.1.2 Algorithm: Directed Hypothesis Space Generation

To generate an LTL hypothesis space Φ such that Φ is (1) valid on all traces, (2) as strict as possible, and (3) concise, we propose an algorithm which loops through each trajectory π_k one by one and reconciles them. Algorithm 1 gives the concise form of the algorithm. In words, DHSG does the following:

1. Infer all the atomic formulas ϕ_j which are valid on the first trace. This is accomplished using the inference method described in Section 1.3.3.
2. Initialize the hypothesis space Φ as the conjunction of all the valid ϕ_j.
3. Iterate through each remaining trace π_k. For each iteration, an updated Φ_k is constructed.
   (a) Infer all the atomic formulas valid on the current trace π_k.
   (b) Check which of the previous members φ_i of Φ are still valid for this iteration, given the set of atomic formulas currently satisfied.
      - If a given φ_i remains valid, pass it along (unchanged) to the new Φ. It becomes a member of the conjunction.
      - Store the list of ϕ_j which do not appear in any such valid φ_i. This set of "newly satisfied" ϕ_j will be used for the reconciliation process.
      - If a given φ_i from the previous Φ is not satisfied by π_k, store it in the set of "newly violated" φ_i for potential reconciliation.

(c) For each of the "newly satisfied" atomic formulas ϕ_j, and each of the "newly violated" φ_i,

- Construct the disjunction $\varphi_i^* := \phi_j \vee \varphi_i$. This is the reconciliation step.
- Conjoin φ_i^* to the current Φ.

4. Once all traces $\pi_1 ... \pi_n$ have been reconciled, return the final Φ. This is the full hypothesis space.

Algorithm 1: Directed Hypothesis Space Generation

Result: Returns full hypothesis space Φ

Infer all ϕ_j valid on π_1
Initialize $\Phi_{k-1} = \bigwedge_{i=1}^{I} \varphi_i$, where each $\varphi_i := \phi_j$
for $\pi_k \in \Pi_A,\ k \geq 2$ **do**
    Infer all ϕ_j valid on π_k
    SatisfiedOnπ_k_ϕ_j= inferred ϕ_js
    NotNewlySatisfied_ϕ_j = {}
    Φ_k= {}
    for $\varphi_i \in \{\Phi_{k-1}\}$ **do**
        for $\phi_j \in$ *SatisfiedOn*π_k_ϕ_j **do**
            if $\phi_j \in \{\varphi_i\}$ **then**
                Append φ_i to Φ_k (if not already in Φ_k)
                Append ϕ_j to notNewlySatisfied_ϕ_j (if not already in notNewlySatisfied_ϕ_j)
            end
        end
    end
    for $\phi_j \in$ *SatisfiedOn*π_k_ϕ_j
    & $\phi_j \notin$ *NotNewlySatisfied*_ϕ_j **do**
        for $\varphi_i \in \Phi_{k-1}$
        & $\varphi_i \notin \Phi_k$ **do**
            φ_{new}=$\varphi_i \vee \phi_j$
            Append φ_{new} to Φ_k
        end
    end
    $\Phi_{k-1} = \Phi_k$
end
Return Φ_k

3.2 Proof of algorithm correctness

We now walk through the algorithm and prove that the resulting Φ is (1) valid, (2) maximally strict, and (3) concise as required. We accomplish this through induction.

Proposition. **Correctness of algorithm for Φ.**

Given a set of good and bad traces, Algorithm 1 will produce a hypothesis space Φ which is valid, strict, and concise as defined in Section 2.2.5.

Proof. **Validity, strictness, and conciseness of Φ.**

Φ_1 **is valid, strict, and concise:** First infer all atomic LTL formulas ϕ_j which hold on π_1. Let $\Phi_1 = \bigwedge_{i=1}^{I} \varphi_i$, where each $\varphi_i := \phi_j$.

1. Φ_1 **valid?** Assuming a priori that every inferred ϕ_j is valid, the conjunction of all ϕ_j must also be. Then, since

$$\bigwedge_{i=1}^{I} \phi_j = \bigwedge_{i=1}^{I} \varphi_j =: \Phi_1 \tag{3.1}$$

   our Φ_1 must be valid. $\square$

2. Φ_1 **strict?** By our definition, there must not exist some other Φ_1' such that Φ_1' is valid on π_1 **and** Φ_1' implies Φ_1. We found that this is equivalent to requiring that we include all φ_i such that

   (a) φ_i is valid on all traces

   (b) φ_i is not implied by an existing $\varphi' \in \{\Phi\}$

   Moreover, we found that (b) is equivalent to $\{\varphi_i'\} \subset \{\varphi_i\}$. To check (a) subject to (b), suppose we have failed to include some such φ_i. Then, since we have included all valid ϕ_j by assumption, any other valid-but-omitted φ_i^* must take the form

$$\varphi_i^* = \bigvee_{j=1}^{J} \phi_j \quad \text{where} \quad J > 1 \tag{3.2}$$

   However, since

$$\varphi_i^* \text{ valid} \iff \exists \text{ a valid } \phi_j' \in \{\varphi_i^*\} \tag{3.3}$$

and we have already recovered all ϕ_j valid on π_1 and included them in Φ_1, we are left with

$$(\varphi_i := \phi'_j) \in \{\Phi_1\} \quad \text{and} \quad (\phi'_j \in \{\varphi_i^*\}) \in \{\Phi_1\} \tag{3.4}$$

This would violate (b), since

$$(\{\varphi_j\} \subset \{\varphi_i^*\}) \iff (\varphi_i \implies \varphi_i^*) \tag{3.5}$$

Thus, there must not exist any such φ_i^* that we fail to include in Φ_1, and Φ_1 is indeed strict. □

3. Φ_1 **concise?** If Φ_1 is concise, for any member $\varphi_i \in \{\Phi_1\}$, there must not exist another member $\varphi'_i \in \{\Phi_1\}$ such that $\{\varphi_i\} \subset \{\varphi'_i\}$.

   Since

   - all of our φ_i consist of a single ϕ_j, and
   - we add each valid ϕ_j only once, so each φ_i is unique,

   it is impossible for any of the $\varphi_i \in \{\Phi\}$ to be a subset of any other. Thus, Φ_1 is concise. □

* * *

Towards induction, assume validity, strictness, and conciseness hold for Φ_{k-1}.

* * *

Φ_k **is valid, strict, and concise:** We now continue to follow the algorithm, checking for satisfaction of our criteria across iterations. Each iteration updates Φ_{k-1} to Φ_k based on the current trace π_k.
For each $\varphi_i \in \{\Phi_{k-1}\}$**:**

- if $\varphi_i(\pi_k)$ true, leave it unchanged and include it in Φ_k.

  1. Φ_k **valid?** $\varphi_i(\pi_k)$ is true by our condition, so the conjunction of all such φ_i constructs a valid Φ_k.
  2. Φ_k **strict?** Each individual φ_i was as strict as possible on π_{k-1} by assumption. Thus, φ_i unchanged for Φ_k implies φ_i strictness

unchanged, which implies φ_i as strict as possible on π_k. (Note that Φ_k itself is not as strict as possible at this stage, since it lacks the subset of strict-as-possible φ_i which are constructed in the next algorithmic step.)

3. Φ_k **concise?** We know the following:

   (a) Φ_k is initially empty

   (b) Each φ_i from Φ_{k-1} conjoined to Φ_k at this stage remains unchanged

   (c) Φ_{k-1} is concise by assumption

   From (a) and (b), the φ_i in Φ_k are currently a subset of the set of those in Φ_{k-1}. A subset of concise φ_i must also be concise, so Φ_k at this stage is concise.

- if $\varphi_i(\pi_k)$ not true, for each $\{\phi_j \mid \phi_j(\pi_k) \text{ true } \& \ \phi_j \notin \{\Phi_{k-1}\}\}$, conjoin $\varphi_i \vee \phi_j$ to Φ_k.

1. Φ_k **valid?** We have that

   (a) ϕ_j valid on π_k

   (b) φ_i valid on π_{k-1} by assumption

   (a) and (b) mean that any $\varphi_i^* := \phi_j \vee \varphi_i$ must also be valid, and thus Φ_k remains valid following conjunction of any such φ_i^*. $\square$

2. Φ_k **strict?** Our definition of strictness requires that we include all (valid) φ_i in Φ_k such that φ_i is not implied by some other member of Φ_k, i.e., that any φ_i is not a subset of the elements of another. Thus, we must show that we successfully include all such φ_i. For the current case, where $\varphi_i(\pi_k)$ false,

   – We cannot pass φ_i along unchanged to Φ_k, since

   $$\varphi_j(\pi_k) \in \{\Phi\} \text{ false } \implies \Phi \text{ not valid.}$$

   – We cannot propose $\varphi_i \vee \phi_j^*$ for an **unsatisfied** ϕ_j^* for conjunction onto Φ_k, since

   $$\phi_j^*(\pi_k),\ \varphi_i(\pi_k) \text{ false } \implies [\varphi_i \vee \phi_j^*](\pi_k) \text{ false } \implies \Phi_k \text{ still not valid.}$$

Thus, we are limited to ϕ_j which are satisfied on π_k. However, if we propose $\varphi_i \vee \phi_j^*$ for **every** ϕ_j^* satisfied by π_k, we see that if $\phi_j^* =: \varphi_i^* \in \{\Phi_{k-1}\}$, i.e., if the satisfied ϕ_j^* is already a member of the conjunction of Φ_{k-1}, then any proposed $\{\varphi_i \vee \phi_j^*\} \supset \phi_i^*$, which violates the strictness condition and must be omitted.

Alternatively, in the case that ϕ_j^* is not already its own φ_i in Φ, the proposed $\varphi_i^* := \varphi_i \vee \phi_j^*$ is both satisfied and as strict as possible, since

- Φ_{k-1} is as strict as possible over $\pi_1, ...\pi_{k-1}$ by assumption, which means that the removal of any ϕ_j from the valid $\varphi_i \in \{\Phi_{k-1}\}$ would cause φ_i to lose validity across the traces $\pi_1, ...\pi_{k-1}$
- Without the additional disjunction of ϕ_j^*, φ_i^* alone is not satisfied on π_k

Thus, neither ϕ_j^* nor any of the $\phi_j \in \{\varphi_i\}$ can be removed from φ_i^* without causing φ_j^* to be violated on at least one trace and therefore lose validity. This is equivalent to saying that the disjunction of any subset of the $\phi_j \in \{\varphi_i^*\}$ would not be valid. In all, **there cannot exist any valid** $\varphi_i' \in \{\Phi\}$ **such that** $\varphi_i' \subset \{\varphi_i^*\}$, so all such φ_i^* are indeed as strict as possible.

However, we still must verify that there are no other types of φ_i which could be proposed and conjoined to Φ_k such that Φ_k becomes even stricter. In other words, we must verify that all $\varphi_i^* := \varphi_i \vee \phi_j^*$ conjoined with all $\varphi_i \in \{\Phi_{k-1}\}$ are sufficient for strictness of Φ_k.

We showed in Section 2.2.4 that the form of the hypothesis space can always be represented as conjunctions of φ_i with $\varphi_i := \bigvee_{j=1}^{J} \phi_j$, i.e., we may limit the form of φ_i to pure disjunction. Thus, our only remaining option for possible unrecovered φ_i^* would have the form

$$\varphi_i^* = \left(\bigvee_{j=1}^{J} \phi_j\right) \vee \left(\bigvee_{j^*=1}^{J^*} \phi_{j^*}\right) \quad \text{where} \quad \bigvee_{j=1}^{J} \phi_j =: \varphi_i \in \{\Phi_{k-1}\} \tag{3.6}$$

i.e., a φ_i^* to which we add more than one ϕ_j^* to the disjunction. However, for any such formula, there will always exist some φ_i^{**}

where

$$\varphi_i^{**} = \left(\bigvee_{j=1}^{J} \phi_j\right) \vee \left(\bigvee_{j^*=1}^{J^*-1} \phi_{j^*}\right) \quad \text{s.t.} \quad \varphi_i^{**} \subset \varphi_i^* \tag{3.7}$$

which is also valid for all $\pi_1, ...\pi_k$. This violates our strictness condition, since

$$(\varphi_i^{**} \subset \varphi_i^*) \Longrightarrow (\varphi_i^{**} \Longrightarrow \varphi_i^*)$$

$$\text{and} \quad \varphi_i^{**} = \varphi_i^* \vee \phi_{J^*} \not\Longrightarrow \varphi_i^*$$

Therefore, when we include

- all φ_i such that $[\varphi_i \in \{\Phi_{k-1}\}](\pi_k)$ true
- all $\varphi_i^* := \varphi_i \vee \phi_j^*$ such that $[\varphi_i \in \{\Phi_{k-1}\}](\pi_k)$ false and $[\phi_j^* \notin (\{\varphi_i\} \in \{\Phi_{k-1}\})](\pi_k)$ true

we produce a strict Φ_k as desired. □

- Φ_k **concise?** Conciseness requires that, for any $\varphi_i \in \{\Phi\}$, there may not exist another $\varphi_i' \in \{\Phi\}$ such that φ_i is satisfied if and only if φ_i' is satisfied. We have already shown in our proof of strictness that we do not add any $\varphi_i' \subseteq \varphi_i$, so Φ_k will remain concise.

Therefore, by induction, Φ_n will be valid, strict, and concise. □

CHAPTER 4

TEST IN SIMULATION

4.1 Introduction

This work was motivated in large part by an existing work of Kim et al. [2], on which we build. Kim et al. generate contrastive explanations by inferring LTL constraints on MLD systems through a Bayesian approach, summarized in Section 4.2.

The BayesLTL algorithm was implemented in a publicly-accessible Python script. In the simulation that follows, the DHSG algorithm was implemented in Python, and the general framework from BayesLTL (i.e., classes, input format, and LTL formula-trace checker functions) was reused. As implemented, the DHSG algorithm generates a full hypothesis space for a set of good traces; this full space can then be pared down by filtering it for the best contrastive formulas, as sought by BayesLTL. From here, it is possible to compare the outputs on the same input.

The following sections first describe the approach taken by BayesLTL, give a more detailed explanation of the simulation setup for DHSG, define performance and comparison metrics for the two approaches, and finally analyze the simulation results.

4.2 Existing work: Bayes LTL

4.2.1 Contrastive explanation

The BayesLTL approach begins by considering two sets of prelabeled traces, A (good) and B (bad). The good and bad sets are denoted as $\pi_{\mathbf{A}}$ and $\pi_{\mathbf{B}}$ respectively. Based on these traces, the algorithm attempts to infer the LTL

formulas φ which best serve as behavior classifiers; that is, φ should hold for the good traces $\pi_i \in \pi_{\mathbf{A}}$ while being violated on the bad traces $\pi_i \in \pi_{\mathbf{B}}$ as consistently as possible. Fittingly, they define the accuracy of any φ as

$$\frac{|\{\pi : \pi \models \varphi, \pi \in \pi_{\mathbf{A}}\}| + |\{\pi : \pi \not\models \varphi, \pi \in \pi_{\mathbf{B}}\}|}{|\pi_{\mathbf{A}}| + |\pi_{\mathbf{B}}|} \tag{4.1}$$

where $\pi \models \varphi$ means that π satisfies φ (equivalent to "$\varphi(\pi)$ true" in our previous notation). By this definition, an accuracy of 1 would mean that the corresponding φ is satisfied on all good trajectories and is violated on all bad trajectories.

These inferred contrastive constraints $\{\varphi\}$ now serve as explanations for the goodness and badness of the traces, illustrating the requirements for goodness based on the differences between the sets. This approach operates on the principle that the most relevant system rules lie in the distinctions between acceptable and unacceptable system. This is in response to the hypothesis that contrastive explanations offer valuable insight to humans [10].

4.2.2 Hypothesis space generation

Kim et al. define each trace π_i on a set of boolean propositions $\mathbf{V}$, called the vocabulary. A trace consists of a sequence of time steps; at each time step, there is a subset of the propositions which are true, as in the example in Table 4.1.

Table 4.1: Example of a trace over vocabulary $\mathbf{V} = \{p, q, r, s, t, u, v\}$.

Time	**State**
$t = 1$	v
$t = 2$	q, v
$t = 3$	p, r, v
$t = 4$	u, r, s
$t = 5$	r

When inferring LTL constraints to describe the behavior of the traces, Kim et al. restrict hypotheses to a finite set of predetermined templates for

Template	n_T	Formula	Meaning
φ_{global}	1	$\mathbf{G}p_i$	p_i is true throughout the entire trace.
$\varphi_{eventuality}$	1	$\mathbf{F}p_i$	p_i eventually occurs (may later become false).
$\varphi_{atmostonce}$	1	$\mathbf{G}(p_i \rightarrow (p_i \mathbf{W}\ \mathbf{G}\neg p_i))$	Only one contiguous interval exists where p_i is true.
$\varphi_{stability}$	1	$\mathbf{FG}p_i\ \wedge\ \mathbf{G}(p_i \rightarrow (p_i \mathbf{W}\ \mathbf{G}\neg p_i))$	p_i eventually occurs and stays true forever.
φ_{until}	2	$p_i \mathbf{U} p_j$	p_i has to be true until p_j eventually becomes true.
$\varphi_{response}$	2	$\mathbf{G}(p_i \rightarrow \mathbf{XF}p_j)$	If p_i occurs, p_j eventually follows.
$\varphi_{precedence}$	2	$(p_j \wedge \neg p_i)\mathbf{R}(\neg p_i)$	If p_i occurs, p_j occurred in the past.

Figure 4.1: Selected LTL formula templates (Reproduced from [2]).

basic LTL formulas, such as those in Figure 4.1. These templates T can then be instantiated with the appropriate number of arguments n_T, where each argument is an atomic proposition from the vocabulary. For instantiation, the set of selected propositions are denoted by $\mathbf{p} \in \mathbf{V}^{n_T}$.

The BayesLTL approach builds hypotheses by generating candidate formulas which are conjunctions of a finite number of these instantiated formulas, as in Eq. 4.2. It is important to note here that the template T is the same for all arguments of the conjunction; it is only the set of propositions $\mathbf{p}$ which differs between each argument $T(\mathbf{p})$.

$$\varphi_T = \bigwedge_{\mathbf{p} \in \{\mathbf{p}\}} T(\mathbf{p}) \tag{4.2}$$

By this method, the number of possible candidate φ for a single LTL template is $2^{|\mathbf{V}|^{n_T}}$. For k different templates, the hypothesis space grows with $\mathcal{O}(k \cdot 2^{|\mathbf{V}|^{n_T}})$, which becomes intractable as $\mathbf{V}$ increases.

Once an initial guess is made, it is time to identify the best classifiers φ such that

$$\varphi^* = \text{argmax}_{\Phi} P(\varphi|\mathbf{X})$$

where $\mathbf{X} = (\pi_{\mathbf{A}}, \pi_{\mathbf{B}})$ and

$$P(\varphi|\mathbf{X}) = \frac{P(\varphi)P(\mathbf{X}|\varphi)}{\sum_{\varphi \in \Phi} P(\varphi)P(\mathbf{X}|\varphi)} \tag{4.3}$$

(4.3) is an application of Bayes' theorem. In words, φ^* is the φ which maximizes the conditional probability that, given the good and bad traces $\pi_{\mathbf{A}}$ and $\pi_{\mathbf{B}}$ as evidence, the formula φ will hold on some $\pi_{\mathbf{A}}$ or fail on some $\pi_{\mathbf{B}}$. $P(\varphi)$ is the prior distribution over the hypothesis space, and $P(\mathbf{X}|\varphi)$ is the likelihood of observing $\mathbf{X}$ given φ.

It is clear that the hypothesis space of candidate φ depends heavily on the

prior function, which acts as a tuneable "preference module" that uses an adjustably-weighted cost function to favor certain features in hypotheses. In other words, the search for optimally contrastive candidates is directed by the prior, and the result has the potential to vary depending on the prior parameters.

The prior function is constructed such that template type T, number of conjunctions N (see (4.2)), and proposition type $\mathbf{p}$ can be tuned. Symbolically,

$$T \sim \text{Categorical}(w_T)$$

where $w_T \in \mathbb{R}^k$ is a scalar weight assigned to a given template type;

$$N \sim \text{Geometric}(\lambda)$$

where $N = |\{\mathbf{p}\}|$ is the number of conjunctions, which follows a geometric distribution that decays at rate λ and thus penalizes longer strings of template conjunctions; and

$$p \sim \text{Categorical}(w_p)$$

where $w_p \in \mathbb{R}^{|\mathbf{V}|}$ is a scalar weight assigned to each proposition in $\mathbf{V}$, which allows some propositions to be favored in instantiations of φ over others. The full prior function is then

$$P(\varphi) = P(T)P(N)P(\{\mathbf{p}\}) \tag{4.4}$$

where

$$P(\{\mathbf{p}\}) = \frac{\sum_{\mathbf{p}\in\{\mathbf{p}\}} \sum_{p\in\mathbf{p}} w_p}{N|\mathbf{p}|} \tag{4.5}$$

is the average categorical weight w_p over all propositions.

This prior function relies on the anticipated "saliency" of different templates and propositions in the context of the specific system under consideration. A human designer chooses appropriate weights for the function based on contextual knowledge of what sorts of formulas we may expect to infer.

Finally, to complete the formulation for $P(\varphi|\mathbf{X})$, the likelihood function

for $\mathbf{X}$ given φ is

$$P(\mathbf{X}|\varphi) = \prod_{i=1}^{|\pi_{\mathbf{A}}|} P(\pi_i|\varphi) \prod_{j=1}^{|\pi_{\mathbf{B}}|} P(\pi_j|\varphi) \tag{4.6}$$

where tolerance for imperfect classifiers and noise is introduced via slack variables, so that:

$$P(\pi_i|\varphi) = \begin{cases} 1-\alpha & \text{if} \quad \pi_i \models \varphi \\ \alpha & \text{otherwise} \end{cases}$$

$$P(\pi_j|\varphi) = \begin{cases} 1-\beta & \text{if} \quad \pi_j \not\models \varphi \\ \beta & \text{otherwise} \end{cases}$$

For its sampling at each iteration, this approach employs a Markov Chain Monte Carlo method, the Metropolis-Hastings algorithm [11], to draw samples for candidate φ and infer the φ^* which maximize $\text{argmax}_{\Phi} P(\varphi|\mathbf{X})$.

The number of iterations allowed can be manually set, and the optimal φ^* are returned as the contrastive explanation candidates at the end of the iteration.

4.3 Source code and hardware

The simulation for DHSG was implemented in a Python script which was designed to utilize the basic functions from BayesLTL, in particular the trace checkers. The BayesLTL source code was pulled from GitHub (see https://github.com/IBM/BayesLTL commit 379924d).

The simulation was run in Spyder for Python 3.8 on an AMD Ryzen 5 4600H.

4.4 Test cases

The following sections define comparison metrics and step through a number of test cases. Each test case was run on distinct sets of good- and bad-labeled traces which were analyzed using both DHSG and BayesLTL. The number of traces, as well as the length of the trace vocabulary, varies between test cases. Unless otherwise specified, the templates and weights shown in Table

4.2 were used for the BayesLTL prior.

Table 4.2: BayesLTL default templates and prior weights

Template	**Weight**
eventual	2.0
global	1.0
until	1.0
response	2.0
stability	1.0
atmostonce	1.0
sometime_before	1.0

4.4.1 Comparison metrics and interpretation

BayesLTL seeks the best-fitting contrastive explanations, returning its top 10 optimally-contrastive, optimally-"interesting" LTL rules. It includes built-in flexibility that allows for consideration of rules which are not perfectly contrastive, *including* rules which are not satisfied by all good traces. We omit these rules in the comparison because they do not meet the validity criterion and would suggest that the set of good traces is not uniformly good. However, we do allow for imperfect contrast (i.e., recovered rules may fail to be invalid on every single bad trace).

The DHSG algorithm produces the full hypothesis space which satisfies validity, strictness, and conciseness. The hypothesis space is represented as a list of every individual φ_i as defined in (2.1). (The conjunction of these is implicit, since if every individual φ_i is valid on the good traces, their conjunction is also valid.) However, for comparison to BayesLTL and in the interest of a contrastive explanation, we can filter the elements of the full hypothesis space to output only those which produce the best contrast through violation on negative traces. This contrastive subset of the hypothesis space is the focus for the comparisons.

The result metrics are as follows:

- Accuracy: Fraction of traces for which φ_i is a valid contrastive explanation. For example, given 4 good traces and 3 bad traces, a φ_i which

is valid on all good traces and invalid on 2 out of 3 bad traces would have accuracy $(4+2)/(4+3) = 6/7 = .857$.

- c-score: Fraction of negative traces which violate φ_i. For the example with 4 good traces and 3 bad, the φ_i violated by 2 of 3 bad traces would have a c-score of $2/3 = .667$.

- Recovery advantage: number of contrastive explanations with c-score at least 0.75 which are recovered by DHSG and could not be recovered by BayesLTL. This includes any applicable explanations consisting of disjunctions, as BayesLTL recovers only conjunctions of ϕ_j. Candidate φ that make up the recovery advantage for DHSG are colored green. Of these, the φ which were within the BayesLTL search space, and thus theoretically recoverable by BayesLTL, are a darker green.

- Runtime: time taken to run the inference, in seconds.

4.4.2 Test Case 1 (Simple)

This test case is designed to produce relatively simple hypothesis spaces and demonstrate the recovery advantage of DHSG. The distinguishing parameters of this test case are

- 4 good traces, 4 bad traces

- Vocabulary length: 7

The metrics for the case are given in Table 4.5. The full DHSG and adjusted BayesLTL contrastive outputs are given in Tables 4.6 and 4.7, respectively. In this case, all φ in the DHSG recovery advantage are categorically impossible for recovery by BayesLTL, as they are disjuncts. However, all φ found by DHSG within the search space of BayesLTL were successfully recovered by the latter. To observe this, compare entries 1, 2, 5, and 6 in Table 4.6 with entries 1, 2, 5, and 10 in Table 4.7.

In all, Case 1 demonstrates DHSG potential for high recovery advantage accomplished within a much shorter runtime than BayesLTL (though more complex cases will be shown to increase DHSG runtime significantly).

Table 4.3: Case 1 good (A) traces.

Time	$\pi_{\mathbf{A1}}$	$\pi_{\mathbf{A2}}$	$\pi_{\mathbf{A3}}$	$\pi_{\mathbf{A4}}$
$t = 1$	v	v	r, v	r, p, v
$t = 2$	q, v	r, v	r, v	r, v
$t = 3$	p, r, v	p, r, v	p, r	r, p
$t = 4$	-	-	u, r, s	-
$t = 5$	-	-	r	-

Table 4.4: Case 1 bad (B) traces.

Time	$\pi_{\mathbf{B1}}$	$\pi_{\mathbf{B2}}$	$\pi_{\mathbf{B3}}$	$\pi_{\mathbf{B4}}$
$t = 1$	p, u	q, r	p, u	q, r
$t = 2$	s, r	u, p	s	q, r, u
$t = 3$	r	p, u	r, p	p, u
$t = 4$	s, t, p	q, u, v	p, v, r	u, v

Table 4.5: Case 1 metrics summary.

DHSG $\lvert\{\Phi\}\rvert = 103$, 13 contrastive φ_i		
Metric	**BayesLTL**	**DHSG**
Runtime (s)	0.560	0.037
# of φ with c-score $\geq$ .75 (non-redundant)	8	13
Recovery advantage	-	9

Table 4.6: Case 1 DHSG contrastive output (minimum c-score =.75)

i	φ_i **in words**	**c-score**	**Accuracy**
1	(v) has to be true until (p) eventually becomes true	1	1
2	(v) has to be true until (r) eventually becomes true	1	1
3	(r) has to be true until (v) eventually becomes true, **OR** (v) is true throughout the entire trace	1	1
4	(p) has to be true until (v) eventually becomes true, **OR** (r) has to be true until (s) eventually becomes true, **OR** (v) is true throughout the entire trace	1	1
5	If (p) occurs, (r) eventually follows	.75	.875
6	(r) eventually occurs and stays true forever	.75	.875
7	(r) has to be true until (s) eventually becomes true, **OR** If (u) occurs, (q) eventually follows	.75	.875
8	(r) has to be true until (v) eventually becomes true, **OR** If (u) occurs, (q) eventually follows	.75	.875
9	(p) has to be true until (r) eventually becomes true, **OR** (r) has to be true until (s) eventually becomes true, **OR** (v) is true throughout the entire trace	.75	.875
10	(p) has to be true until (r) eventually becomes true, **OR** If (p) occurs, (s) eventually follows, **OR** (v) is true throughout the entire trace	.75	.875
11	(p) has to be true until (r) eventually becomes true, **OR** If (v) occurs, (s) eventually follows, **OR** (v) is true throughout the entire trace	.75	.875
12	(p) has to be true until (v) eventually becomes true, **OR** If (p) occurs, (s) eventually follows, **OR** (v) is true throughout the entire trace	.75	.875
13	(p) has to be true until (v) eventually becomes true, **OR** If (v) occurs, (s) eventually follows, **OR** (v) is true throughout the entire trace	.75	.875

Table 4.7: Case 1 BayesLTL output (minimum c-score =.75)

#	**Rule in words**	**c-score**	**Accuracy**
1	(v) has to be true until (r) eventually becomes true	1	1
2	(v) has to be true until (p) eventually becomes true	1	1
3*	(v) has to be true until (p) eventually becomes true **AND** (v) has to be true until (r) eventually becomes true	1	1
4	If (s) occurs, (r) eventually follows **AND** If (v) occurs, (p) eventually follows	1	1
5*	If (p) occurs, (r) eventually follows	.75	.875
6	If (p) occurs, (r) eventually follows **AND** If (v) occurs, (r) eventually follows	.75	.875
7	If (p) occurs, (r) eventually follows **AND** If (q) occurs, (v) eventually follows	.75	.875
8	If (p) occurs, (r) eventually follows **AND** If (q) occurs, (p) eventually follows	.75	.875
9	If (p) occurs, (r) eventually follows **AND** If (u) occurs, (r) eventually follows	.75	.875
10	(r) eventually occurs and stays true forever	.75	.875

* denotes a candidate formula which is logically redundant when taken together with the other candidates.

4.4.3 Test Case 2 (Many traces)

This test case is designed to show the effect of a larger quantity of good traces on algorithm performance. The distinguishing parameters of this test case are

- 8 good traces, 4 bad traces
- Vocabulary length: 7

The traces are the same as in Test Case 1 except for the addition of 4 more good traces, given in Table 4.8.

As summarized in Table 4.9, the doubling of the number of traces slowed DHSG significantly, though it remained under the BayesLTL benchmark. The contrastive output also noticeably outperformed BayesLTL, recovering 3 more perfectly-contrastive φ_i, including one which is within the search space for BayesLTL. This can be observed by comparing the first four entries of Table 4.10 with Table 4.11.

Table 4.8: Case 2 good traces (additional).

Time	$\pi_{\mathbf{A5}}$	$\pi_{\mathbf{A6}}$	$\pi_{\mathbf{A7}}$	$\pi_{\mathbf{A8}}$
$t = 1$	p, q, s, v	r, t, v	p, q, v	p, q, r, v
$t = 2$	q, r, u, v	r, v	q, v, s	q, r, u, s, v
$t = 3$	p, r, v	p, r, v	p, q, v, u	p, s, v
$t = 4$	-	u, v, s	q, r, s, v	u, v
$t = 5$	-	u, v	-	s, v

Table 4.9: Case 2 metrics summary.

DHSG $\lvert\{\Phi\}\rvert = 487$, 12 contrastive φ_i		
Metric	**BayesLTL**	**DHSG**
Runtime (s)	0.528	0.280
# of φ with c-score $\geq$ .75 (non-redundant)	1	12
Recovery advantage	-	11

Table 4.10: Case 2 DHSG contrastive output (minimum c-score =.75)

i	φ_i **in words**	**c-score**	**Accuracy**
1	(v) has to be true until (p) eventually becomes true	1	1
2	(v) has to be true until (r) eventually becomes true	1	1
3	(r) has to be true until (v) eventually becomes true, **OR** (v) is true throughout the entire trace	1	1
4	(p) has to be true until (v) eventually becomes true, **OR** (r) has to be true until (s) eventually becomes true, **OR** (v) is true throughout the entire trace	1	1
5	If (p) occurs, (r) eventually follows	.75	.917
6	(r) has to be true until (v) eventually becomes true, **OR** If (u) occurs, (q) eventually follows	.75	.917
7	(r) has to be true until (s) eventually becomes true, **OR** If (u) occurs, (q) eventually follows	.75	.917
8	(p) has to be true until (v) eventually becomes true, **OR** If (v) occurs, (s) eventually follows, **OR** (v) is true throughout the entire trace	.75	.917
9	(p) has to be true until (v) eventually becomes true, **OR** If (p) occurs, (s) eventually follows, **OR** (v) is true throughout the entire trace	.75	.917
10	(p) has to be true until (r) eventually becomes true, **OR** (r) has to be true until (s) eventually becomes true, **OR** (v) is true throughout the entire trace	.75	.917
11	(p) has to be true until (r) eventually becomes true, **OR** If (v) occurs, (s) eventually follows, **OR** (v) is true throughout the entire trace	.75	.917
12	(p) has to be true until (r) eventually becomes true, **OR** If (p) occurs, (s) eventually follows, **OR** (v) is true throughout the entire trace	.75	.917

Table 4.11: Case 2 BayesLTL output (minimum c-score =.75)

#	**Rule in words**	**c-score**	**Accuracy**
1	(v) has to be true until (r) eventually becomes true	1	1

4.4.4 Test Case 3 (Large vocabulary)

This test case is designed to show the effect of a larger vocabulary on algorithm performance. The distinguishing parameters of this test case are

- 8 good traces, 4 bad traces
- Vocabulary length: 14

As shown in Table 4.12, the large vocabulary caused a significant slowdown in DHSG runtime, while BayesLTL saw little change.

Table 4.12: Case 3 metrics summary.

DHSG $\lvert\{\Phi\}\rvert = 19385$, 2318 contrastive φ_i		
Metric	**BayesLTL**	**DHSG**
Runtime (s)	0.546	26.123
# of φ with c-score $\geq$ .75 (non-redundant)	1	2318
Recovery advantage	-	2317

4.4.5 Test Case 4 (Perturbed BayesLTL Prior)

This test case focuses on BayesLTL inference. It is designed to show the effect of varying the prior weights for the same traces as given in Test Case 1. The distinguishing parameters of this test case are

- 4 good traces, 4 bad traces (see Tables 4.3 and 4.4)
- Vocabulary length: 7
- **Altered prior weights** as in Table 4.13. Note that Case 4.2 is the uniform-weight case.

The metrics in Table 4.14 confirm that the adjustment of prior weights, as expected, has an effect on the hypotheses output by BayesLTL. Choice of weights had a sufficient influence on the search space to alter which formulas were ultimately recovered, influencing the number of accurate hypotheses returned.

Table 4.13: BayesLTL templates and prior weights

Template	**Case 1**	**Case 4.1**	**Case 4.2**
eventual	2.0	1.0	1.0
global	1.0	1.0	1.0
until	1.0	2.0	1.0
response	2.0	2.0	1.0
stability	1.0	1.0	1.0
atmostonce	1.0	1.0	1.0
sometime_before	1.0	2.0	1.0

Table 4.14: Case 4 metrics summary.

DHSG $\lvert\{\Phi\}\rvert = 103$, 13 contrastive φ_i			
Metric	**Case 1**	**Case 4.1**	**Case 4.2**
Runtime (s)	0.560	0.497	0.485
# of φ with c-score $\geq$ .75 (non-redundant)	8	10	9
DHSG recovery advantage	9	7	8

4.5 Practicality considerations

DHSG has the potential to produce very large hypothesis spaces. Very large hypothesis spaces can occur due to

- large vocabulary size resulting in many permutations to be checked for each LTL template
- potential for long disjunctive formulas (up to n disjuncts for n good traces)

Large hypothesis spaces present two distinct challenges, one in interpretation and one in computation. To ease the interpretive burden of processing so many candidate contrastive formulas in the output, the simulation output may be adjusted by

1. increasing the accuracy requirement: only output candidate φ_i which have very high or perfect accuracy

2. limiting the acceptable length for disjunctive formulas: only output φ_i which have less than a given number of disjuncts

While item (1) can significantly decrease the size of the output, particularly in the case of few or simple bad traces, there may remain a very large number of highly accurate φ_i candidates. On the other hand, item (2) offers a useful reduction in the output size; however, the retroactive reduction of the output does not improve computational burden. To simultaneously limit the output size and improve computation time, then, the algorithm may be adapted not to consider any further disjunctions in a φ_i beyond a set length. While this reduces the strictness of the hypothesis space, in the case where long disjunctive (and potentially "overfitted") hypotheses are not of interest, it is a reasonable change to make.

For example, consider Case 3. If we require a perfect c-score and the length of disjunctions is limited, the hypothesis space size, contrastive output size, and computation time can be greatly reduced, as in Table 4.15.

Table 4.15: Case 3, limited disjunction length.

	No limit	$\lvert\{\varphi_i\}\rvert \leq 3$	$\lvert\{\varphi_i\}\rvert \leq 2$
$\lvert\{\Phi\}\rvert$	19385	3503	658
# contrastive φ_i	792	192	48
Runtime (s)	26.083	7.549	1.719

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Results and discussion

The simulation suggests that there are different appropriate use cases for BayesLTL and DHSG. Depending on the size and complexity of the traces, prior knowledge of the expected LTL formula structures, and preference between speed and accuracy, each approach offers distinct advantages.

In general, we note that DHSG

- offers the full, strict hypothesis space of all φ_i which perfectly describe the good traces (+)
- does not require a guess for a prior distribution (+)
- requires a much longer runtime for large vocabularies (-)
- produces a large number of long disjunctive formulas unless limited (+/-)

Meanwhile, BayesLTL

- may be limited to a short runtime (+)
- cannot return all valid formulas for the good traces (-)
- is dependent on the weighting of the prior (-)
- allows for adjustment of the prior if sufficient knowledge of the expected formula structure is known (+/-)

Section 4.5 offers methods to partly alleviate the long runtime and large hypothesis space complications of DHSG, at the cost of complete strictness.

5.2 Application: Inference of unknown regions

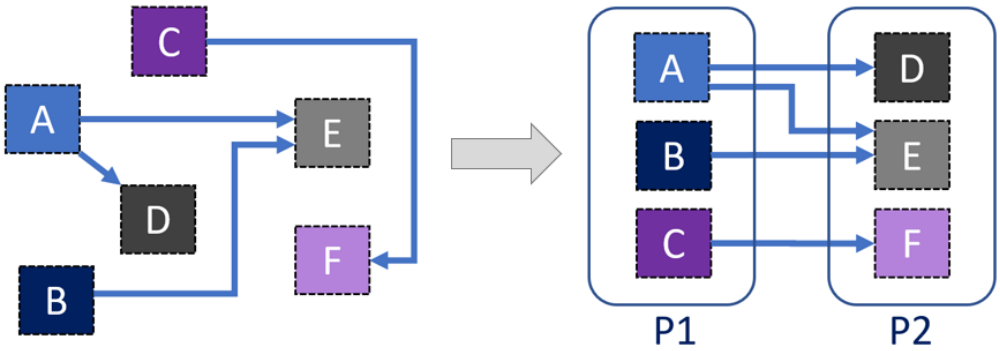

Figure 5.1: Unit regions A-F are found to comprise two larger true regions, P1 and P2.

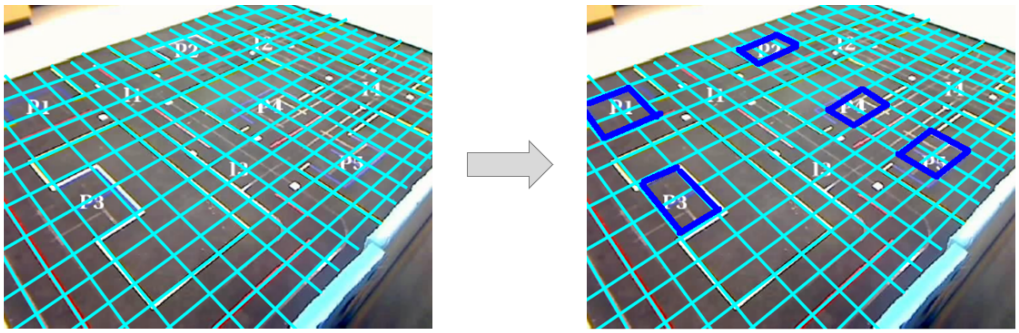

Figure 5.2: Example of region inference on the tabletop from [1].

A major application of this inference method is to determine the true size and shape of state-space regions as understood by the unfamiliar system. To clarify the distinction between a "true" and a guessed region, consider the tabletop robot example from Section 2.1.

For the tabletop robot, we are given the relevant regions a priori; the size and locations of parking lots and roads, the relevant regions, are labeled. However, when observing an unfamiliar system, we cannot expect to be provided with bounded, labeled regions as in this example. Instead, states such as position would need to be collected and post-processed to determine apparent region boundaries. Consider Figure 5.1. Here, the larger, true regions P1 and P2 are initially unknown, and the state space is discretized into units A through F. When inference is performed on such a discretized state space, LTL rules that appear to be distinct rules may in fact all be manifestations

of a single true rule. For instance, the true rule

$$\text{If P1, then eventually P2} \tag{5.1}$$

may appear as

$$(\text{If A, eventually D}) \vee (\text{If B, eventually E}) \vee (\text{If A, eventually E}) \tag{5.2}$$

A sketch of such a region reconciliation for the tabletop robot is given in Figure 5.2.

An approach such as BayesLTL, which assumes the state-space regions are known, does not lend itself to inference of true regions. The purely-conjunctive rule format, operating on the $A - E$ domain, would not be able to recover (5.2) and would thus return no representation of the true rule given by (5.1).

DHSG, in allowing for hypotheses of disjunctions, offers a method to build true regions through comparison of disjuncts. DHSG would guarantee recovery of (5.2). From the full hypothesis space, it would be possible to filter all recovered disjunctive rules such that

1. every disjunct contains the same LTL template type
2. the set of atomic propositions across the same argument in the template are in a similar region of the state space

If (2) is true, it suggests that the propositions actually represent a sample of a larger, true region–in our example, that A and B make up $P1$, and that D and E make up $P2$. From here, a new formula of the form in (5.1) may be hypothesized.

Moreover, this process may be performed first on a subset of the good traces:

1. Select a subset of the good traces in Π_A
2. Perform DHSG
3. Filter output for disjunctions of like templates
4. Group propositions by argument

5. Propose new "true" regions for inference and rewrite the disjunctive hypothesis accordingly

6. Check new hypothesis for accuracy and contrast across entire set of traces

An algorithm which allows for inference of state space regions further removes us from reliance on prior knowledge of the system we aim to understand. This provides a method of approach to less-understood problems, where we are forced to guess the state variables.

5.3 Towards explainable AI

As we near the end, we return to the beginning. Section 1.1 outlined a three-step approach to explanation construction for explainable AI; this work rooted itself in the first step, model construction. In approaching the next two steps, when the model is distilled for relevance and clarity, it is fruitful to look to the fields of translation and pragmatics (Figure 5.3).

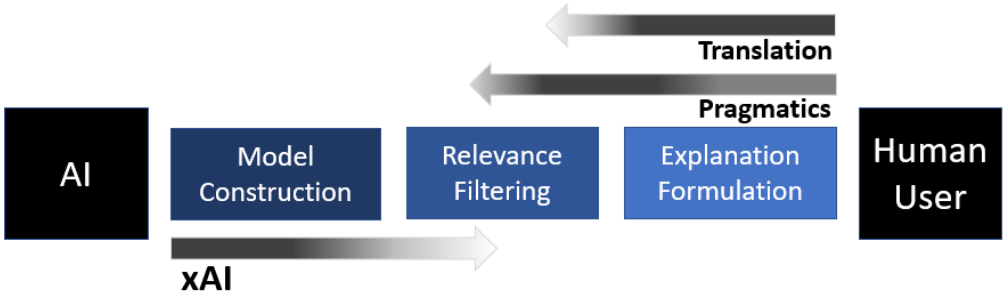

Figure 5.3: Bridging the gap. Future research could combine principles of optimization and pragmatics.

These fields recognize that any participant in a communication act brings with him a set of assumptions, and a successful communication act only takes place when the participants can understand each other within their own frameworks [12]. While xAI has considered framework comparison as a model-reconciliation problem [13], little has been done to investigate the variation in human "models" [14]; approaches generally seek a single "optimal" explanation, despite the fact that the field of pragmatics has long recognized

significant differences between individual humans' frameworks [15]. What's more, psychological factors such as confirmation bias and motivated reasoning threaten to blur the directly-factual explanations favored by xAI approaches [16]. Oversights in these areas present hefty obstacles in aerospace, where explanations need not only be relevant and clear, but immediately so; a split-second decision can make the difference in missile evasion or collision avoidance, and the cockpit environment is already very taxing on a pilot's attention [17]. In light of these complications, optimal relevance and clarity seem only achievable via an approach which accounts for individual frames and biases.

Pragmatics and translation theory indeed offer a lens for frames and bias. Relevance and frame theory from pragmatics, as well as translation's extensive body of work on major communicative issues such as "translation loss", cohesion and coherence [18], and skopos theory [19] explore the nature of human communication and understanding. Moreover, these concepts may be combined with mathematical structures such as the association matrix model [20] to build a framework in which an AI can express human expectations and associations. Optimization-focused studies of goal-oriented communication provide statistical approaches to account for human bias when formulating messages [21]. A reconciliation of these areas promises an interdisciplinary route towards individualized explanation.

In short, an effective explanation must be both humanized and personalized, to a degree far beyond the current scope of xAI work. The pursuit of such a human-oriented explanation certainly makes for a long road, but the destination is a whole new world of human-AI harmony.

REFERENCES

[1] S. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning under temporal logic constraints," 2010.

[2] J. Kim, C. Muise, A. Shah, S. Agarwal, and J. Shah, "Bayesian inference of linear temporal logic specifications for contrastive explanations," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, 2019, pp. 5591–5598.

[3] "Avoidance rerouter arr 7000," 2021. [Online]. Available: https://www.collinsaerospace.com/en/what-we-do/Military-And-Defense/Avionics/Software-Applications/Avoidance-Re-Router-Arr-7000

[4] "Slingshot edge," 2021. [Online]. Available: https://slingshotaerospace.com/products

[5] M. Ornik, S. Carr, A. Israel, and U. Topcu, "Myopic control of systems with unknown dynamics," in *2019 American Control Conference (ACC)*. IEEE, 2019, pp. 1064–1071.

[6] T. Lombrozo, "The structure and function of explanations," *Trends in cognitive sciences*, vol. 10, no. 10, pp. 464–470, 2006.

[7] E. M. Wolff, "Control of dynamical systems with temporal logic specifications," Ph.D. dissertation, California Institute of Technology, May 2014. [Online]. Available: https://resolver.caltech.edu/CaltechTHESIS:02172014-121159358

[8] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints." *Automatica*, vol. 35, pp. 407–427, 1999.

[9] J. Kim, C. Muise, S. Agarwal, and M. Agarwal, "Bayesltl," 2019. [Online]. Available: https://github.com/IBM/BayesLTL commit 379924d

[10] N. Elzein, "The demand for contrastive explanations," *Philosophical Studies*, vol. 176, no. 5, pp. 1325–1339, 2019.

[11] S. Chib and E. Greenberg, "Understanding the metropolis-hastings algorithm," *The american statistician*, vol. 49, no. 4, pp. 327–335, 1995.

[12] M. Agar, *Language shock: Understanding the culture of conversation*. New York, NY: Perennial, 2008.

[13] T. Chakraborti, S. Sreedharan, Y. Zhang, and S. Kambhampati, "Plan explanations as model reconciliation: Moving beyond explanation as soliloquy," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017. [Online]. Available: https://doi.org/10.24963/ijcai.2017/23 pp. 156–163.

[14] B. Mittelstadt, C. Russell, and S. Wachter, "Explaining explanations in ai," in *Proceedings of the conference on fairness, accountability, and transparency*, 2019, pp. 279–288.

[15] M. A. Bednarek, "Frames revisited—the coherence-inducing function of frames," *Journal of pragmatics*, vol. 37, no. 5, pp. 685–705, 2005.

[16] U. Hahn and A. J. Harris, "What does it mean to be biased: Motivated reasoning and rationality," in *Psychology of learning and motivation*. Elsevier, 2014, vol. 61, pp. 41–102.

[17] T. Deveans and R. H. Kewley, "Overcoming information overload in the cockpit," MILITARY ACADEMY WEST POINT NY OPERATIONS RESEARCH CENTER, Tech. Rep., 2009.

[18] S. Blum-Kulka, "Shifts of cohesion and coherence in translation," in *The Translation Studies Reader*, 2nd ed., L. Venuti, Ed. Routledge, 2004, pp. 298–313.

[19] H. Vermeer, "Skopos and commission on translational action," in *The Translation Studies Reader*, 2nd ed., L. Venuti, Ed. Routledge, 2004, pp. 191–202.

[20] M. A. Nowak and N. L. Komarova, "Towards an evolutionary theory of language," *Trends in cognitive sciences*, vol. 5, no. 7, pp. 288–295, 2001.

[21] O. Goldreich, B. Juba, and M. Sudan, "A theory of goal-oriented communication," *Journal of the ACM (JACM)*, vol. 59, no. 2, pp. 1–65, 2012.