Fanoos: Multi-Resolution, Multi-Strength, Interactive Explanations for Learned Systems Additional Technical Appendix with References in Body

Anonymous for Double-Blind Review

Anon. Institution `Anon. Email`

Abstract. Machine learning becomes increasingly important to control the behavior of safety and financially critical components in sophisticated environments, where the inability to understand learned components in general, and neural nets in particular, poses serious obstacles to their adoption. Explainability and interpretability methods for learned systems have gained considerable academic attention, but the focus of current approaches on only one aspect of explanation, at a fixed level of abstraction, and limited if any formal guarantees, prevents those explanations from being digestible by the relevant stakeholders (e.g., end users, certification authorities, engineers) with their diverse backgrounds and situation-specific needs. We introduce Fanoos, a flexible framework for combining formal verification techniques, heuristic search, and user interaction to explore explanations at the desired level of granularity and fidelity. We demonstrate the ability of Fanoos to produce and adjust the abstractness of explanations in response to user requests on a learned controller for an inverted double pendulum and on a learned CPU usage model.

1 Problem Overview

Explainability and safety in machine learning (ML) are a subject of increasing academic and public concern. As ML continues to grow in success and adoption by wide-ranging industries, the impact of these algorithms' behavior on people's lives is becoming highly non-trivial. Unfortunately, many of the most performant contemporary ML algorithms—neural networks (NNs) in particular—are widely considered black-boxes, with the method by which they perform their duties not being amenable to direct human comprehension. The inability to understand learned components as thoroughly as more traditional software poses serious obstacles to their adoption [7, 1, 13, 32, 87, 31, 88, 54] due to safety concerns, difficulty to debug and maintain, and explicit legal requirements.[1] Symbiotic human-machine interactions can lead to safer and more robust agents, but this task requires effective and versatile communication [77, 68].

Interpretability of learned systems has been studied in the context of computer science intermittently since at least the late 1980s, particularly in the area

[1] For example, the "right to an explanation" legislation [25] adopted by the European Union.

of rule extraction (e.g., [6]), adaptive/non-linear control analysis (e.g., [19]), various rule-learning paradigms (e.g., inductive logic programming[57], association rule learning [3], and its predecessors [63, 26]), and formal analysis (e.g., [15, 84, 85, 79, 43, 56]). Notwithstanding this long history, main-stream attention has risen only recently due to increased impact on daily life of opaque AI [1] with novel initiatives focused on the problem domain [33, 59, 4, 86].

Despite this attention, however, most explanatory systems developed for ML are hard-coded to provide a single type of explanation with descriptions at a certain fixed level of abstraction and a fixed type of guarantee about the system behavior, if any. This not only prevents the explanations generated from being digestible by multiple audiences (the end-user, the intermediate engineers who are non-experts in the ML component, and the ML-engineer for instance), but in fact limits the use by any single audience since the levels of abstraction and formal guarantees needed are situation and goal specific, not just a function of the recipient's background. When using a microscope, one varies between low and high magnification in order to find what they are looking for and explore samples; these same capabilities are desirable for XAI for much the same reasons. For example, most consumers of autonomous vehicles may prefer to ask general questions — for instance, "What do you do when you detect a person in front of you?" — and receive a break-down of qualitatively different behaviors for different situations, such as braking when traveling slowly enough, and doing a sharp swerve when traveling too fast to brake. An engineer checking actuator compliance, however, might require greater details, opting to specify precise parameters of the scene and preferring that the car report exact motor commands; the context of use and the audience determine which level of abstraction is best, and supporting multiple types of abstractions in turn supports more use-cases and audiences. Further, the explanations for such a component need to range from formal guarantees to rough tendencies—one might want to formally guarantee that the car will avoid collisions always, while it might be sufficient that it usually (but perhaps not always) drives slowly when its battery is low.

The divide between formal and probabilistic explanations also relates to events that are imaginable versus events that may actually occur; formal methods may check every point in a space for conformance to a condition, but if bad behavior only occurs on measure-zero sets, the system would be safe while not being provably so in formalizations lacking knowledge of statistics (e.g., when a car must keep distance $>10\,\text{cm}$ to obstacles, formally we can get arbitrarily close but not equal; in practice, the difference with $\geq 10\,\text{cm}$ might be irrelevant). Explainable ML systems should enable these sorts of search and smooth variation in need—but at the moment they do not in general.

To address these needs, we introduce Fanoos,[2] an algorithm blending a diverse array of technologies to interactively provide explanations at varying levels of abstraction and fidelity (i.e., probabilistic versus formal guarentees) to meet user's needs. Our algorithm is applicable to currently ubiquitous ML methods—

[2] "Fanoos" means lantern in [redacted]. Our approach shines a light on black-box AI. Source code can be found at `https://github.com/anon-73bea4a3`.

such as feed-forward neural networks (FFNNs) and high-dimensional polynomial kernels.

2 The Fanoos Approach

Fanoos is an interactive system that allows users to pose a variety of questions grounded in a domain specification (e.g., what environmental conditions cause a robot to swerve left), receive replies from the system, and request that explanations be made more or less abstract. An illustration of the process and component interactions can be found in Appendix B, with a fuller example of interaction and discussion of the UI located in Appendix C. A more detailed technical overview of the implementation can be found in Appendix F. Crucially, Fanoos provides explanations of high fidelity[3] while considering whether the explanation should be formally sound or probabilistically reasonable (which removes the "noise" incurred by measure-zero sets that can plague formal descriptions). To this end, we combine techniques from formal verification, interactive systems, and heuristic search over knowledge domains when responding to user questions and requests. Here, we do not focus on the information presentation aesthetics so much as ensuring that the proper information can be produced (see Appendix 1.5).

2.1 Knowledge Domains and User Questions

In the following discussion, let L be the learned system under analysis (which we will assume is piece-wise continuous), q be the question posed by the user, S_I be the (bounded) input space to L, and S_O be the output space to L, $S_{IO}{=}S_I \cup S_O$ be the joint of the input and output space[4], and r be the response given by the system. In order to formulate question q and response r, a library listing basic domain information D is provided to Fanoos; D lists what S_I and S_O are and provides a set of predicates, P, expressed over the domain symbols in S_{IO}, i.e., for all $p \in P$, the free variables $FV(p)$ are chosen from the variable names $V(S_{IO})$, that is $FV(p) \subseteq V(S_{IO})$.

```
(Fanoos) when_do_you_usually and(outputtorque_low ,
    ↪ statevalueestimate_high )?
```

Listing 1.1: Question to illuminate input space S_I

For queries that formally guarantee behavior (see the first three rows in Table 1), we require that the relevant predicates in P can expose their internals as first-order formulas; this enables us to guarantee they are satisfied over all members of a given set[5] via typical SAT-solvers (such as Z3 [20]). The other query types require only being able to evaluate question q on a variable assignment provided. The members of P can be generated in a variety of ways, e.g.,

[3] Since Fanoos is a decompositional approach; see Section 3.

[4] Subscripts I for input, O for output, etc., are simply symbols, not any sort of richer mathematical object.

[5] The box abstractions we introduce in a moment to be more precise.

Table 1: Description of questions that Fanoos can respond to

Type q_t	Description	Question content q_c		Example
		accepts	illum. restrictions	
When Do You	Tell all sets (formal consideration of all cases) in the input space S_I that have the potential to cause q_c	Subset s of S_O s.t. there exists a member of s that causes q_c to be true. Found with SAT-solver.	S_I Cannot contain variables from S_O.	when_do_you $\underbrace{\text{move\_at\_high\_speed?}}_{\text{Predicate } p \in D}$
What Do You Do When	Tell all possible learner responses in the collection of input states that q_c accepts	Subset s of S_I s.t. there exists a member of s that causes q_c to be true. Found with SAT-solver.	S_O Cannot contain variables from S_I.	what_do_you_do_when *and*(close_to_target_orientation, close_to_target_position)?
What are the Circumstances in Which	Tell information about what input-output pairs occur in the subset of input-outputs accepted by q_c	Subset s of S_{IO} s.t. there exists a member of s that causes q_c to be true. Found with SAT-solver.	S_{IO} None	what_are_the_circumstances_in_which *and*(close_to_target_position, steer_to_right) *or* move_at_low_speed?
... Usually	Statistical tendency. Avoids measure-zero sets that are unlikely seen in practice.	q_c was found to be true at least once via statistical sampling. Discussion in Appendix 1.4.		when_do_you_usually move_at_low_speed *or* steer_to_left? what_do_you_usually_do_when moving_toward_target_position? what_are_the_usual_circumstances_in_which *and*(close_to_target_position, steer_close_to_center)?

by forming most predicates through procedural generation and then using a few hand-tailored predicates to capture particular cases.[6] Further, since the semantics of the predicates are grounded, they have the potential to be generated from demonstration (e.g., a discussed in Appendix 1.2).

2.2 Reachability Analysis of L

Having established what knowledge the system is given, we proceed to explain our process. First, users select a question type q_t and the content of the question q_c to query the system. That is, $q = (q_t, q_c)$, where q_t is a member of the first column of Table 1 and q_c is a sentence in disjunctive normal form (DNF) over a subset of P that obeys the restrictions listed in Table 1. To ease discussion, we will refer to variables and sets of variable assignments that p accepts (AC) and those that p illuminates (IL), with the intuition being that the user wants to know what configuration of illuminated variables result in the variable configurations accepted by q_c; see Table 1 for example queries. In other words, when

[6] For example, operational definitions of "high", "low", etc., might be derived from sample data by setting thresholds on quantile values—e.g., 90% or higher might be considered "high".

a users asks a question, Fanoos answers by describing a collection of situations that necessarily include those related to the user's question; this answer is conservative in that it may include additional situations, but never intentionally excludes cases.

With question q provided, we analyze the learned system L to find subsets in the inputs S_I and outputs S_O that agree with configuration q_c and may over-approximate the behavior of L. Specifically, we use CEGAR [16, 15] with boxes (hyper-cubes) as abstractions and a random choice between a bisection or trisection along the longest normalized axis as the refinement process to find the collect of box tuples, B, specified below:

$$B=\{(B_I^{(i)}, B_O^{(i)}) \in \mathrm{BX}(S_I)\times\mathrm{BX}(S_O) \mid B_O^{(i)} \supseteq L(B_I^{(i)}) \\ \wedge (\exists\, (c,d) \in \mathrm{T}.(AC_q(B_c^{(i)}) \wedge IL_q(B_d^{(i)})))\}$$

where $\mathrm{BX}(X)$ is the set of boxes over space X and $\mathrm{T}=\{(O,I),(I,O),(IO,IO)\}$. See Fig. 1 for an example drawn from analysis conducted on the model in Section 4.1. For feed-forward neural nets with non-decreasing activation functions, B may be found by covering the input space, propagating boxes through the network, testing membership to B of the resulting input- and output-boxes, and refining the input mesh as needed over input-boxes that produce output-boxes overlapping with B. Implementation details can be found in Appendix 6.1. The exact size of the boxes found by CEGAR are determined by a series of hyper-parameters, [7] which the system maintains in *states*, a fact we will return to in Section 2.4.

Prior to proceeding, B may undergo some limited merging, particularly when an increase of abstraction level is sought. Our merging process is closed over the family of abstract states we have selected; up to a numerical precision threshold[8], boxes may only merge together to form larger boxes, and only if the smaller boxes formed a partition of the larger box. Merging (when applicable) increases the size of abstract states without anywhere increasing the union of the states — this is not necessarily what would occur if one attempted the CEGARanalysis again with parameters promoting higher granularity. Essentially, merging here is one strategy of increasing abstraction level while retaining some finer-resolution details. As before, the state maintains the parameters which control the extent of this stage's merging. Box-merging itself is an NP-hard task in general, so we adopted roughly a greedy approximation scheme interlaced with hand-written heuristics for accelerating match-finding (e.g., feasibility-checks via shared-vertex lookups) and parameters bounding the extent of computation.

[7] For example, the maximum number of refinement iterations or the minimal size abstractions one is willing to consider; for details on such hyper-parameters of CEGAR and other bounded-model checking approaches, the interested reader may refer to [16, 15, 11].

[8] Value differences within this threshold allow the merge to proceed. This enables some sets of boxes to merge into larger boxes with slightly larger volumes. Note that allowing abstract states to grow continues to make our estimates conservative, and thus continues to ensure the soundness of Fanoos.

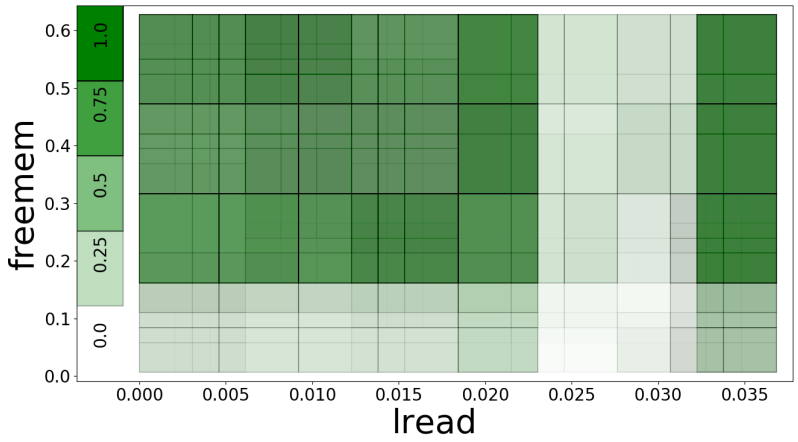

Fig. 1: Example of reachability results. Shown is a 2-D projection of 5-D input-boxes which were selected for retention since their corresponding output-boxes satisfied the user query. Darker areas had boxes with greater volume along the axes not shown; the volumes were normalized in respect to the input-space bounding-box volume over the non-visible axes. A scale is along the y-axis.

2.3 Generating Descriptions

Having generated B, we produce an initial response, r_0, to the user's query in three steps as follows: (1) for each member of B, we extract the box tuple members that were illuminated by q (in the case where S_{IO} is illuminated, we produce a joint box over both tuple members), forming a set of joint boxes, B'; (2) next, we heuristically search over P for members that describe B' and compute a set of predicates covering all boxes; (3) finally, we format the box covering for user presentation. A sample result answer is shown in listing 1.2, and details on steps (2) and (3) how to produce it follow below. Pseudocode for the process can be found in Appendix 6.4.

```
(0.45789160, 0.61440409, 'x Near Normal Levels')
(0.31030792, 0.51991449, 'pole2Angle_rateOfChange Near Normal
    ↪ Levels')
(0.12008841, 0.37943400, 'pole1Angle_rateOfChange High')
(0.06128723, 0.22426058, 'pole2Angle Low')
```

Listing 1.2: Initial answer to question in listing 1.1. Also see Appendix 1.5 and C.

Producing a Covering of B' Our search over P for members covering B' is largely based around the greedy construction of a set covering using a carefully designed candidate score.

For each member $b \in B'$ we want to find a set of candidate predicates capable of describing the box and for which we would like to form a larger covering. We find a subset $P_b \subseteq P$ that is consistent with b in that each member of P_b passes the checks called for by q_t when evaluated on b (see the Description column of Table 1). This process is expedited by a feasibility check of each member of P on a vector randomly sampled from b, prior to the expensive check for inclusion in P_b. Having P_b, we filter the candidate set further to P'_b: members of P_b that appear most specific to b; notice that in our setting, where predicates of varying abstraction level co-mingle in P, P_b may contain many members that only loosely fit b. The subset P'_b is formed by sampling outside of b at increasing radii (in the ℓ_∞ sense), collecting those members of P_b that fail to hold true at the earliest radius (see the pseudo-code in Appendix E for further details). Importantly, looking ahead to forming a full covering of B, if none of the predicates fail prior to exhausting[9] this sampling, we report P'_b as empty, allowing us to handle b downstream; this avoids having "difficult" boxes force the use of weak predicates that would "wash out" more granular details. Generally speaking, we try to be specific at this phase under the assumption that the desired description granularity was determined earlier, presumably during the CEGAR analysis but also (in extensions to our approach) by altering[10] P. For instance, if we want a subset of P_b that was less specific to b than P'_b, we might perform the CEGAR-analysis so to produce larger abstract states.

We next leverage the P'_b sets to construct a covering of B', proceeding in an iterative greedy fashion. Specifically, we form an *initial* covering

$$K_f = \mathscr{C}_f(\cup_{b\in B'} \cup_{p\in P'_b} \{(p, b)\}, P)$$

where $\mathscr{C}_i(R, H)$ is the covering established at iteration i, incrementing to

$$\mathscr{C}_{i+1}(R, H) = \mathscr{C}_i(R, H) \cup \left\{\text{argmax}_{p\in H\setminus\mathscr{C}_i(R,H)}\mu(p, \mathscr{C}_i(R, H), R)\right\}$$

where the cover score μ is

$$\mu(p, \mathscr{C}_i(R, H), R) = \sum_{b\in B'} \mathbb{1}(|\mathsf{UV}(b, \mathscr{C}_i(R, H)) \cap \mathsf{FV}(p)| > 0)\mathbb{1}(\ (p, b) \in R)$$

and $\mathsf{UV}(b, \mathscr{C}_i(R, H))$ is the set of variables in b that are not constrained by $\mathscr{C}_i(R, H) \cap P_b$; since the boxes are multivariate and our predicates typically constrain only a subset of the variables, we select predicates based on how many boxes would have open variables covered by them. Pseudocode for this process can be found in algorithm 6. Notice that K_f is not necessarily an approximately minimal covering of B with respect to members of P—by forcing $p \in P'_b$ when

[9] The operational meaning of "exhausting", as well as the radii sampled, are all parameters stored in the state.

[10] For example, filtering using a (possibly learned) taxonomy. Our current implementation has first-steps in this direction, allowing users to enable an optional operator that filters P based on estimates of a model trained on previous interactions' data (see Appendix 6.3 for technical details). We comment further on this in Section 2.4 and indicate why this operator is left as optional in Section 4.

calculating the cover score μ, we enforce additional specificity criteria that the covering should adhere to. At this stage, due to the nature of P'_b being more specific than P_b, it is possible that some members of K_f cover one another — that is, there may exist $p \in K_f$ such that $K_f \setminus \{p\}$ still covers as much of B' as K_f did. By forming K_f, we have found a collection of fairly specific predicates that are capable of covering B' to the largest extent possible — but we now must remove predicates that are dominated by the collection of other, potentially less-specific predicates that we had to include. To address this concern, we simply perform another covering,

$$C_F = \mathscr{C}_F(\cup_{b \in B'} \cup_{p \in P_b} \{(p, b)\}, K_f)$$

After forming C_F, any boxes that fail to be covered even partially (for example, because P_b or P'_b are empty) are reported with a box-range predicate: atomic predicates that simply list the variable range in the box (ex: "Box(x : [-1, 0], y: [0.5, 0.3])"). [11] In other words, we extend C_F to a set C'_F by introducing new predicates specific to each completely uncovered box so that C'_F does cover all boxes in B'.

Cleaning and Formatting Output for User Having produced C'_F, we collect the covering's content into a formula in DNF. If $b \in B'$ and s is a maximal, non-singleton subset of $C_F \cap P_b$, then we form a conjunction over the members of s, doing some filtering for conjunctions that would be implied by others. Concretely, if $A = \cup_{b \in B'} \{P_b \cap K'_F\}$, we construct:

$$d_0 = \{\wedge_{p \in s} p | (s \in A) \wedge \neg(\ \exists s' \in A . s \subsetneq s'\ )\}\ .$$

Notice that the filtering done in d_0 is only a minor efficiency aid — we do a final redundancy check momentarily that, had we not filtered here, would have achieved similar results. Ultimately, the members of d_0 are conjunctions of predicates,[12] with their membership to the set being a disjunction. Prior to actually converting d_0 to DNF, we form d'_0 by: (1) removing any $c \in d_0$ that are redundant given the rest of d_0 (see algorithm 10)—in practice, d_0 is small enough to simply do full one-vs-rest comparison and determine results with a sat-solver; (2) attempting to merge any remaining box-range predicates into the minimal number necessary to cover the sets they are responsible for. Note that this redundancy check is distinct from our process to form C_F from K_f, since the latter worked at the abstract-state level (so it could not tell if a disjunction of predicates covered a box when no individual predicate covered the entirety of the box) and attempted to intelligently select predicates to retain based maximizing a score.

Finally, r_0 is constructed by listing each c that exists in d'_0 sorted by two relevance scores: first, the approximate proportion of the volume in B' uniquely

[11] Minimizing occurrences of box-range predicates (if desirable) can be related to discussion in [70] regarding the need for weak, universal rules to — as the phrase goes — prevent items failing through the cracks of what is covered elsewhere.

[12] From hereon, we use "conjunct" only to reference non-degenerate (i.e., not 1-ary or 0-ary) cases.

covered by c, and second by the approximate proportion of total volume c covers in B'. The algorithm is shown in algorithm 9. These sorting-scores can be thought of similarly to recall measures. Specificity is more difficult to tackle, since it would require determining the volume covered by each predicate (which may be an arbitrary first-order formula) across the box bounding the universe, not just the hyper-cubes at hand; this can be approximated for each predicate using set-inversion, but requires non-trivial additional computation for each condition (albeit work that may be amenable to one-time pre-computation up to a certain granularity).

2.4 User Feedback and Revaluation

Based on the initial response r_0, users can request a more abstract or less abstract explanation. We view this alternate explanation generation as another heuristic search, where the system searches over a series of states to find those that are deemed acceptable by the user. The states primarily include algorithm hyper-parameters, the history of interaction, the question to be answered, and the set B. Abstraction and refinement operators take a current state and produce a new one, often by adjusting the system hyper-parameters and recomputing B. This state-operator model of user response allows for rich styles of interaction with the user, beyond and alongside of the three-valued responses of acceptance, increase, or decrease of the abstraction level shown in listing 1.3.

```
1 (0.11771033, 0.12043966, 'And(pole1Angle Near Normal Levels,
    ↪ pole1Angle_rateOfChange Near Normal Levels, pole2Angle
    ↪ High, pole2Angle_rateOfChange Low, vx Low)')
2 (0.06948142, 0.07269412, 'And(pole1Angle High,
    ↪ pole1Angle_rateOfChange Near Normal Levels,
    ↪ pole2Angle_rateOfChange High, vx Low, x Near Normal
    ↪ Levels)')
3 (0.04513659, 0.06282974, 'And(endOfPole2_x Near Normal
    ↪ Levels, pole1Angle Low, pole1Angle_rateOfChange High,
    ↪ pole2Angle High, pole2Angle_rateOfChange Near Normal
    ↪ Levels, x High)')q
4 type letter followed by enter key: b - break and ask a
    ↪ different question,
5  l - less abstract , m - more abstract, h - history travel
```

Listing 1.3: Response to "less abstract" than listing 1.2

For instance, a history-travel operator allows the state (and thus r) to return to an earlier point in the interaction process, if the user feels that response was more informative; from there, the user may investigate an alternate path of abstractions. Other implemented operators allow for refinements of specified parts of explanations as opposed to the entire reply; the simplest form of this is by regenerating the explanation without using a predicate that the user specified be ignored, while a more sophisicated operator determines the predicates to filter out automatically by learning from past interaction. Underlying the discussion of these mechanisms is the utilization of a concept of abstractness, a notion we further comment on in the next subsection.

As future work, we are exploring the use of active learning leveraging user interactions to select from a collection of operators, with particular interest in bootstrapping the learning process using operationally defined oracles to approximate users. See Appendix 1.1 for further discussion.

2.5 Capturing the Concept of Abstractness

The criteria to judge degree-of-abstractness in the lay sense are often difficult to capture.[13] We consider abstractness a diverse set of relations that subsume the part-of-whole relation, and thus also generally includes the subset relation. For our purposes, defining this notion is not necessary, since we simply wish to utilize the fact of its existence. We understand abstractness to be a semantic concept that shows itself by producing a partial ordering over semantic states (their "abstractness" level) which is in turn reflected in the lower-order semantics of the input-output boxes, and ultimately is reflected in our syntax via explanations of different granularity. Discussions of representative formalisms most relevant to computer science can be found in [17, 74, 73, 49, 50][14] and an excellent discussion of the philosophical underpinnings and extensions can be found in [27].

In this work, the primary method of producing explanations at desired levels of abstraction is entirely implicit—that is, without explicitly tracking what boxes or predicates are considered more or less abstract (note that the operator we mentioned that attempts to learn such relations is invoked optionally by human users, and it not used in any of the evaluations we present). Instead, we leverage the groundedness of our predicates to naturally form partial orderings over semantic states (their "abstractness" level) which in turn are appropriately reflected in syntax.

On the opposite end of the spectrum is explicit expert tuning of abstraction orderings to be used in the system. Fanoos can easily be adapted to leverage expert-labels (e.g., taxonomies as in [72], or even simply type/grouping-labels without explicit hierarchical information) to preference subsets of predicates conditionally on user-responses, but for the sake of this paper, we reserve agreement with expert-labels as an independent metric of performance in our evaluation, prohibiting the free use of such knowledge by the algorithm during testing. As a side benefit, by forgoing direct supervision, we demonstrate that the concept of abstractness is recoverable from the semantics and structure of the problem itself.

3 Related Work and Discussion

Many methods are closely related to XAI, stemming from a diverse body of literature and various application domains, e.g., [19, 6, 3, 37, 71, 65, 42, 82, 10].

[13] See Appendix G.

[14] [17] features abstraction in verification, [74] features abstraction at play in interpretating programs, [73] is an excellent example of interfaces providing a notion of abstractness in network communications, and [49, 50] discuss notions of abstractness relavent for type systems in object-oriented programming languages.

Numerous taxonomies of explanation families have been proposed [53, 7, 45, 48, 6, 1, 32, 12, 30, 66, 78, 14, 67, 61, 34, 13], with popular divisions being (1) between explanations that leverage internal mechanics of systems to generate descriptions (decompositional approaches) versus those that exclusively leverage input-output relations (pedagogical)[15], (2) the medium that comprises the explanation (such as with most-predictive-features [65], summaries of internal states via finite-state-machines [46], natural language descriptions [37, 45] or even visual representations [41, 45]), (3) theoretical criteria for a good explanation (see, for instance, [54]), and (4) specificity and fidelity of explanation. Overall, most of these approaches advocate for producing human-consumable information—whether in natural language, logic, or visual plots—conveying the behavior of the learned system in situations of interest.

Rule-based systems such as expert systems, and work in the (high-level) planning community have a long history of producing explanations in various forms; notably, hierarchical planning [37, 55] naturally lends itself to explanations of multiple abstraction levels. All these methods, however, canonically work on the symbolic level, making them inapplicable to most modern ML methods. High fidelity, comprehensible rules describing data points can also be discovered with weakly-consistent inductive logic programming [57] or association rule learning [40, 3] typical in data-mining. However, these approaches are typically pedagogical—not designed to leverage access to the internals of the system—do not offer a variety of descriptions abstractions or strengths, and are typically not interactive. While some extensions of association rule learning (e.g., [72, 36, 35]) do consider multiple abstraction levels (e.g., [72, 36]), they are still pedagogical and non-interactive. Further, they only describe subsets of the data they analyze [16] and only understand abstractness syntactically, requiring complete taxonomies be provided explicitly and up-front. Our approach, by contrast, leverages semantic information, attempts to efficiently describe all relevant data instances, and produces descriptions that are necessarily reflective of the mechanism under study.

Decision support systems [60, 81, 23, 24, 22] typically allow users to interactively investigate data, with operations such as drill-ups in OLAP (OnLine Analytical Processing) cubes analogous to a simple form of abstraction in that setting. The typical notions of analysis, however, largely operate by truncating portions of data distributions and running analytics packages on selected subregions at user's requests, failing to leverage access to the data-generation mechanism when present, and failing to provide explicit abstractions or explicit guarantees about the material it presents.

More closely related to our work are approaches to formally analyze neural networks to extract rules, ensure safety, or determine decision stability, which we

[15] We have also found this to be referred to as "introspective" explanations versus "rationalizations", such as in [45]

[16] While support and confidence thresholds may be set sufficiently low to ensure each transaction is described by at least one rule, the result would be a deluge of highly redundant, low-precision rules lacking most practical value (this may be considered the most extreme case of the "rare itemset problem" as discussed in [51]).

discuss in more detail below. Techniques related to our inner-loop reachability analysis have been used for stability or reachability analysis in systems that are otherwise hard to analyze analytically. Reachability analysis for FFNNs based on abstract interpretation domains, interval arithmetic, or set inversion has been used in rule extraction and neural net stability analysis [6, 21, 75, 83] and continues to be relevant, e.g., for verification of multi-layer perceptrons [64], estimating the reachable states of closed-loop systems with multi-layer perceptrons in the loop [87], estimating the domain of validity of neural networks [2], and analyzing security of neural networks [80]. While these works provide methods to extract descriptions that faithfully reflect behavior of the network, they do not generally ensure descriptions are comprehensible by end-users, do not explore the practice of strengthening descriptions by ignoring the effects of measure-zero sets, and do not consider varying description abstraction.

The high-level components of our approach can be compared to [38], where hand-tunable rule-based methods with natural language interfaces encapsulate a module responsible for extracting information about the ML system, with explanation generation in part relying on minimal set-covering methods to find predicates capturing the model's states. Extending this approach to generate more varying-resolution descriptions, however, does not seem like a trivial endeavor, since (1) it is not clear that the system can appropriately handle predicates that are not logically independent, and expecting experts to explicitly know and encode all possible dependencies can be unrealistic, (2) the system described does not have a method to vary the type of explanation provided for a given query when its initial response is unsatisfactory, and (3) the method produces explanations by first learning simpler models via MDPs. Learning simpler models by sampling behavior of more sophisticated models is an often-utilized, widely applicable method to bootstrap human understanding (e.g. [12, 46, 33]), but it comes at the cost of failing to leverage substantial information from the internals of the targeted learned system. Crucially, such a technique cannot guarantee the fidelity of their explanations in respect to the learned system being explained, in contrast to our approach.

In [62], the authors develop vocabularies and circumstance-specific human models to determine the parameters of the desired levels of abstraction, specificity and location in robot-provided explanations about the robot's specific, previous experiences in terms of trajectories in a specific environment, as opposed to the more generally applicable conditional explanations about the internals of the learned component generated by Fanoos. The particular notions of abstraction and granularity from multiple, distinct, unmixable vocabularies of [62] evaluate explanations in the context of their specific application and are not immediately applicable nor easily transferable to other domains. Fanoos, by contrast, does not require separate vocabularies and enables descriptions to include multiple abstraction levels (for example, mixing them as in the sentence "House X and a 6m large patch on house Y both need to be painted").

Closest in spirit to our work are the planning-related explanations of [71],[17] providing multiple levels of abstraction with a user-in-the-loop refinement process, but with a focus on markedly different search spaces, models of human interaction, algorithms for description generation and extraction, and experiments. Further, we attempt to tackle the difficult problem of extracting high-level symbolic knowledge from systems where such concepts are not natively embedded, in contrast to [71], who consider purely symbolic systems.

In summary, current approaches focus on single aspects of explanations, fixed levels of abstraction, and inflexible guarantees about the explanations given. Our stance is that an interleaving between automated formal techniques, search heuristics, and user interaction is an effective strategy to achieve the desired flexibility in explanations and the desired adjustable level of fidelity.

4 Experiments and Results

In this section we discuss empirical demonstrations of Fanoos's ability to produce and adjust descriptions across two different domains. The code implementing our method, the models analyzed, the database of raw-results, and the analysis code used to generate the results presented will be released in the near future. Our presentation keeps with norms presently established in XAI work presented across multiple venues (e.g., [4] and section 4.8 of [7]).

4.1 Systems Analyzed

We analyze learned systems from robotics control and more traditional ML predictions to demonstrate the applicability to diverse domains. Information on the predicates available for each domain can be found in Table 2.

Table 2: Summary statistics of predicates in each domain. For description of MA/LA labels, see Section 4.3. Percentages are rounded to three decimal places.

	CPU	IDP
Input space predicates	33	62
Output space predicates	19	12
Joint input-output space predicates	8	4
Total with MA (more abstract) label	20	15
Total with LA (less abstract) label	40	63
Percentage of Pred.s Labeled MA	0.333	0.192

[17] We note that [71] was published after the core of our approach was developed; both of our thinkings developed independently.

Inverted Double Pendulum (IDP) The control policy for an inverted double-pendulum is tasked to keep a pole steady and upright; the pole consists of two under-actuated segments attached end-to-end, rotationally free in the same plane; the only actuated component is a cart with the pivot point of the lower segment attached. While similar to the basic inverted single pendulum example in control, this setting is substantially more complicated, since multi-pendulum systems are known to exhibit chaotic behavior [44, 47]. The trained policy was taken from reinforcement learning literature.[18]. The seven-dimensional observation space — the bounding box for which can be found in Table 4 located in Appendix D — includes the segment's angles, the cart x-position, their time derivatives, and the y-coordinate of the second pole. The output is a torque in $[-1, 1]$Nm and a state-value estimate, which is not a priori bounded. Internal to the analyzed model is a transformation to convert the observations we provide to the form expected by the networks—chiefly, the angles are converted to sines and cosines and observations are standardized in respect to the mean and standard deviation of the model's training runs. The values chosen for the input space bounding box were inspired by the 5% and 95% quantile values over simulated runs of the model in the rl-zoo framework. We expanded the input box beyond this range to allow for the examination of rare inputs and observations the model was not necessarily trained on;[19] whether the analysis stays in the region trained-over depends on the user's question.

CPU Usage (CPU) We also analyze a more traditional ML algorithm for a non-control task — a polynomial kernel regression for modeling CPU usage. Specifically, we use a three-degree fully polynomial basis over a 5-dimensional input space[20] to linearly regress-out a three-dimensional vector. We trained our model using the publicly available data from [76][21]. The observations are [*lread*, *scall*, *sread*, *freemem*, *freeswap*] and the response variables we predict are [*lwrite*, *swrite*, *usr*]. For analysis convenience, we normalized the input space in respect to the training set min and max prior to featurization, which was the same method of normalization used during model training and evaluation; alternatively, Fanoos could have used the raw (unnormalized) feature space and insert the normalization as part of the model-pipeline. We opted to analyze an algorithm with a degree-3 polynomial feature-set after normalizing the data in respect to the minimum and maximum of the training set since this achieved the highest performance—over 90% accuracy—on a 90%-10% train-test split of

[18] `https://github.com/araffin/rl-baselines-zoo` trained using PPO2 [69] which, as an actor-critic method, uses one network to produce the action, and one to estimate state-action values.

[19] For instance, the implemented train and test environments exit whenever the end of the second segment is below a certain height. In real applications, a user may wish to check that a sensible recovery is attempted after entering such unseen situations.

[20] The input space includes cross-terms and the zero-degree element—e.g., x^2y and 1 are members.

[21] Dataset available at `https://www.openml.org/api/v1/json/data/562`

the data compared to similar models with 1,2, or 4 degree kernels[22]. While the weights of the kernel may be interpreted in some sense (such as indicating which individual feature is, by itself, most influential), the joint correlation between the features and non-linear transformations of the input values makes it far from clear how the model behaves over the original input space. For Fanoos, the input space bounding box was determined from the 5% and 95% quantiles for each input-variable over the full, normalized dataset; the exact values can be found in Table 5 located in Appendix D.

4.2 Experiment Design

Tests were conducted using synthetically generated user interactions, with the goal of determining whether our approach properly changes the description abstractness in response to the user request. The domain and question type were randomly chosen, the latter selected among the options listed in Table 1. The questions themselves were randomly generated to have up to four disjuncts, each with conjuncts of length no more than four; conjuncts were ensured to be distinct, and only predicates respecting the constraints of the question-type were used. Interaction with Fanoos post-question-asking was randomly selected from four alternatives (here, MA means "more abstract" and LA means "less abstract"): Initial refinement of 0.25 or 0.20 $\rightarrow$ make LA $\rightarrow$ make MA $\rightarrow$ exit; Initial refinement of 0.125 or 0.10 $\rightarrow$ make MA $\rightarrow$ make LA $\rightarrow$ exit. For the results presented here, over 130 interactions were held, resulting in several hundred responses from Fanoos.

4.3 Metrics

We evaluated the abstractness of each response from Fanoos using metrics across the following categories: reachability analysis, structural description, and expert labeling.

Reachability Analysis We compare the reachability analysis results when producing descriptions of different abstraction levels, which call for different levels of refinement. Specifically, we record statistics about the input-boxes generated during the CEGAR-like analysis after normalizing them in respect to the input space bounding box so that each axis is in $[0, 1]$, yielding results which are comparable across domains. The following values are examined:

- Volume of the box (product of its side lengths).
- Sum of the box side lengths. Unlike the box volume, this measure is at least as large as the maximum side length.
- Number of boxes used to form the description.

[22] Note that while we did do due-diligence in producing a performant and soundly-trained model, the primary point is to produce a model worthy of analysis.

The volume and summed-side-lengths are distributions, reported in terms of the summary statistics shown in Table 3. These metrics provide a rough sense of the abstractness notion implicit in the size of boxes and how they relate to descriptions.

Description Structure Fanoos responds to users with a multi-weighted DNF description. This structure is summarized as follows to give a rough sense of how specific each description is by itself:

- Number of disjuncts, including atomic predicates
- Number of conjuncts, excluding atomic predicates[23]
- Number of named predicates: atomic user-defined predicates that occur anywhere in the description (even in conjunctions). This excludes box-range predicates.
- Number of box-range predicates that occur anywhere (i.e., in conjuncts as well as stand-alone).

The Jaccard score and overlap coefficients below are used to measure similarity in the verbiage used in two descriptions.

- Jaccard score: general similarity between two descriptions, viewing the set of atomic predicates used in each description as a bag-of-words.
- Overlap coefficient: measures whether one description is simply a more "verbose" variant of the other, in the sense that the set of atomic predicates of one is a subset of the other using $\frac{|S_1 \cap S_2|}{min(|S_1|,|S_2|)}$, where S_1 and S_2 are the sets of predicates used in the two descriptions.

Expert Labeling As humans, we understand which atomic predicates map to comparatively more abstract notions in the world, and as such can evaluate the responses based on usage of more vs. less abstract verbiage. It is important to note that this approach—on descriptions built from atomic predicates—yields an informative approximation rather than a true measure of abstractness for the following reasons: it is not clear that the abstractness of a description's components translates in an obvious fashion to the abstractness of the whole (in a similar vein, we do not rule out the possibility that predicates of the same level in the partially ordered set of abstractness can be combined to descriptions of different abstractness[24]). This phenomenon becomes more pronounced in coarsely grained partitions, where nuances are hidden in the partitions. For simplicity we choose two classes, more abstract (MA) vs. less abstract (LA), in the measures below:

[23] By excluding atomic predicates, this provides some rough measure of the number of "complex" terms.

[24] For example, just because two description use verbiage from the same expert-labeled category of abstractness, it does not mean the two descriptions have the same level of abstractness.

- Number of predicates accounting for multiplicity, i.e., if an atomic predicate q has label MA and occurs twice in a sentence, it contributes two to this score.
- Number of unique predicates: e.g., if an atomic predicate q has label MA and occurs twice in a sentence, it contributes one to this score.
- Prevalence: ratio of unique predicates to the total number of atomic predicates in a description. This measure is particularly useful when aggregating the results of multiple descriptions into one distribution, since the occurrence of predicates is statistically coupled with the length of descriptions; under a null hypothesis of random generation of content, one would expect longer sentences to contain more MA,LA predicates, but expect the proportion to remain constant.

Each of the above measures have obvious counter-parts for predicates with MA/LA labels. We note that prevalence will not necessarily sum to 1, since box-range predicates are atomic predicates without either label.

4.4 Results

Running the experiments described in Section 4.2, we collected a series of states and the summary statistics on them described in Section 4.3. Since we are chiefly interested in evaluating whether a description changes to reflect the abstraction requested by the user, we examine the relative change in response to user interaction. Specifically, for pre-interaction state S_t and post-interaction state S_{t+1}, we collect metrics $m(S_{t+1}) - m(S_t)$, describing *relative* change for each domain-response combination. This same process is used for the Jaccard score and overlap coefficients, except the values in question are computed as $m(S_{t+1}, S_t)$. The medians of these distributions are reported in Table 3.

In summary, the reachability and structural metrics follow the desired trends: when the user requests greater abstraction (MA), the boxes become larger, and the sentences become structurally less complex—namely, they become shorter (fewer disjuncts), have disjuncts that are less complicated (fewer explicit conjuncts, hence more atomic predicates), use fewer unique terms overall (reduction in named predicates) and resort less often to referring to the exact values of a box (reduction in box-range predicates). Symmetric statements can be made for when requests for less abstraction (LA) are issued. From the overlap and Jaccard scores, we can see that the changes in response complexity are not simply due to increased verbosity—simply adding or removing phrases to the descriptions from the prior steps—but also the result of changes in the verbiage used; this is appropriate since abstractness is not exclusively a function of description specificity.

Trends for the expert labels are similar, though more subtle to interpret. We see that use of LA-related terms follows the trend of user requests with respect to multiplicity and uniqueness counts (increases for LA-requests, decreases for MA-requests), while being less clear with respect to prevalence (uniform 0 scores). For use of MA terms, we see that the prevalence is correlated with user requests in the expected fashion (decrease on LA requests, increase on MA requests).

Table 3: Median relative change in description before and after Fanoos adjusts the abstraction in the requested direction Results are rounded to three decimal places. Further notes in Appendix C.

			CPU	CPU	IDP	IDP
		Request	LA	MA	LA	MA
Reachability	Boxes	Number	8417.5	-8678.0	2.0	-16.0
	Sum side lengths	Max	-1.125	1.125	-1.625	1.625
		Median	-1.187	1.188	-2.451	2.438
		Min	-0.979	0.986	-2.556	2.556
		Sum	21668.865	-22131.937	582.549	-553.007
	Volume	Max	-0.015	0.015	-0.004	0.004
		Median	-0.003	0.003	-0.004	0.004
		Min	-0.001	0.001	-0.003	0.003
		Sum	-0.03	0.03	-0.168	0.166
Structural	Jaccard		0.106	0.211	0.056	0.056
	Overlap coeff.		0.5	0.714	0.25	0.25
	Conjuncts		1.0	-2.0	0.5	-2.5
	Disjuncts		7.0	-7.5	2.0	-2.5
	Named preds.		1.0	-1.0	1.0	-4.5
	Box-Range preds.		2.0	-2.0	1.5	-1.5
Expert	MA terms	Multiplicity	3.0	-3.0	24.0	-20.0
		Uniqueness	0.0	0.0	1.0	-1.5
		Prevalence	-0.018	0.014	-0.75	0.771
	LA terms	Multiplicity	20.0	-21.5	68.5	-86.0
		Uniqueness	2.0	-2.0	12.0	-14.0
		Prevalence	0.0	0.0	0.0	0.0

Further, we see that this correlation is mirrored for the MA counts when taken relative to the same measures for LA terms. Specifically, when a user requests greater abstraction (MA), the counts for LA terms decrease far more than those of MA terms, and the symmetric situation occurs for requests of lower abstraction (LA), as expected. While they depict encouraging trends, we take these expert-label measures with caution, due to the fragility of reasoning about the complete description's abstractness based on its constituents (recall that the abstractness of a description is not necessarily directly linked to the abstractness of its components). Nevertheless, these results—labelings coupled with the structural trends—lend solid support that Fanoos can recover substantial elements of an expert's notions about abstractness by leveraging the grounded semantics of the predicates.

5 Conclusions And Future Work

Fanoos is an explanitory framework for ML systems that mixes technologies ranging from heuristic search to classical verification. Our experiments lend solid support that Fanoos can produce and navigate explanations at multiple

granularities and strengths. We are investigating operator-selection learning and further data-driven predicate generation to accelerate knowledge base construction - the latter focusing on representational power, extrapolation intuitiveness, behavioral certainty, and data efficiency. Finally, this work can adopt engineering improvements to ML-specific reachability computations. Further discussion is in Appendix A. We will continue to explore Fanoos's potential, and hope that the community finds inspiration in both the methodology and philosophical underpinnings presented here.

References

1. Adadi, A., Berrada, M.: Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). IEEE Access **6**, 52138–52160 (2018)
2. Adam, S.P., Karras, D.A., Magoulas, G.D., Vrahatis, M.N.: Reliable estimation of a neural network's domain of validity through interval analysis based inversion. In: 2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015. pp. 1–8 (2015). https://doi.org/10.1109/IJCNN.2015.7280794, `https://doi.org/10.1109/IJCNN.2015.7280794`
3. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Acm sigmod record. vol. 22, pp. 207–216. ACM (1993)
4. Aha, D.W., Darrell, T., Pazzani, M., Reid, D., Sammut, C., Stone, P.: IJCAI 2017 workshop on explainable artificial intelligence (XAI). Melbourne, Australia, August (2017)
5. Althoff, M.: An introduction to CORA 2015. In: Frehse, G., Althoff, M. (eds.) 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems, ARCH@CPSWeek 2014, Berlin, Germany, April 14, 2014 / ARCH@CPSWeek 2015, Seattle, WA, USA, April 13, 2015. EPiC Series in Computing, vol. 34, pp. 120–151. EasyChair (2015), `https://easychair.org/publications/paper/xMm`
6. Andrews, R., Diederich, J., Tickle, A.: Survey and critique of techniques for extracting rules from trained artificial neural networks. Knowledge-Based Systems **6**, 373–389 (12 1995). https://doi.org/10.1016/0950-7051(96)81920-4
7. Anjomshoae, S., Najjar, A., Calvaresi, D., Främling, K.: Explainable agents and robots: Results from a systematic literature review. In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems. pp. 1078–1088. International Foundation for Autonomous Agents and Multiagent Systems (2019)
8. Auer, P.: Using upper confidence bounds for online learning. In: 41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA. pp. 270–279. IEEE Computer Society (2000). https://doi.org/10.1109/SFCS.2000.892116, `https://doi.org/10.1109/SFCS.2000.892116`
9. Ballard, D.: Generalising the Hough transform to detect arbitary patterns. Pattern Recognition **13** (1981)
10. Benz, A., Jäger, G., Van Rooij, R., Van Rooij, R.: Game theory and pragmatics. Springer (2005)
11. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y., et al.: Bounded model checking. Advances in computers **58**(11), 117–148 (2003)

12. Biran, O., Cotton, C.: Explanation and justification in machine learning: A survey. In: IJCAI-17 workshop on explainable AI (XAI). vol. 8, p. 1 (2017)
13. Chakraborti, T., Kulkarni, A., Sreedharan, S., Smith, D.E., Kambhampati, S.: Explicability? legibility? predictability? transparency? privacy? security? the emerging landscape of interpretable agent behavior. In: Proceedings of the International Conference on Automated Planning and Scheduling. vol. 29, pp. 86–96 (2019)
14. Chuang, J., Ramage, D., Manning, C., Heer, J.: Interpretation and trust: Designing model-driven visualizations for text analysis. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 443–452. ACM (2012)
15. Clarke, E., Fehnker, A., Han, Z., Krogh, B., Stursberg, O., Theobald, M.: Verification of hybrid systems based on counterexample-guided abstraction refinement. In: Garavel, H., Hatcliff, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 192–207. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
16. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: International Conference on Computer Aided Verification. pp. 154–169. Springer (2000)
17. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. pp. 238–252 (1977)
18. Cropper, A., Muggleton, S.H.: Metagol system. https://github.com/metagol/metagol (2016), `https://github.com/metagol/metagol`
19. David, Q.: Design issues in adaptive control. IEEE Transactions on Automatic Control **33**(1) (1988)
20. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. TACAS'08/ETAPS'08, Springer-Verlag, Berlin, Heidelberg (2008), `http://dl.acm.org/citation.cfm?id=1792734.1792766`
21. Driescher, A., Korn, U.: Checking stability of neural narx models: An interval approach. IFAC Proceedings Volumes **30**(6), 1005–1010 (1997)
22. Eom, H.B., Lee, S.M.: A survey of decision support system applications (1971–April 1988). Interfaces **20**(3), 65–79 (1990)
23. Eom, S., Kim, E.: A survey of decision support system applications (1995–2001). Journal of the Operational Research Society **57**(11), 1264–1278 (2006)
24. Eom, S.B., Lee, S.M., Kim, E., Somarajan, C.: A survey of decision support system applications (1988–1994). Journal of the Operational Research Society **49**(2), 109–120 (1998)
25. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (General Data Protection Regulation) (2016), `http://data.europa.eu/eli/reg/2016/679/oj`
26. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., et al.: Advances in knowledge discovery and data mining, vol. 21. AAAI press Menlo Park (1996)
27. Floridi, L.: The method of levels of abstraction. Minds and machines **18**(3), 303–329 (2008)
28. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid sys-

tems. In: International Conference on Computer Aided Verification. pp. 379–395. Springer (2011)
29. Fridovich-Keil, D., Herbert, S.L., Fisac, J.F., Deglurkar, S., Tomlin, C.J.: Planning, fast and slow: A framework for adaptive real-time safe trajectory planning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). pp. 387–394. IEEE (2018)
30. Friedrich, G., Zanker, M.: A taxonomy for generating explanations in recommender systems. AI Magazine **32**(3), 90–98 (2011)
31. Garcıa, J., Fernández, F.: A comprehensive survey on safe reinforcement learning. Journal of Machine Learning Research **16**(1), 1437–1480 (2015)
32. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM computing surveys (CSUR) **51**(5), 93 (2019)
33. Gunning, D.: Explainable artificial intelligence (XAI). Defense Advanced Research Projects Agency (DARPA), nd Web **2** (2017), https://www.darpa.mil/attachments/XAIProgramUpdate.pdf
34. Hailesilassie, T.: Rule extraction algorithm for deep neural networks: A review. arXiv preprint arXiv:1610.05267 (2016)
35. Han, J., Fu, Y.: Discovery of multiple-level association rules from large databases. In: VLDB. vol. 95, pp. 420–431. Citeseer (1995)
36. Han, J., Fu, Y.: Mining multiple-level association rules in large databases. IEEE Transactions on knowledge and data engineering **11**(5), 798–805 (1999)
37. Hayes, B., Scassellati, B.: Autonomously constructing hierarchical task networks for planning and human-robot collaboration. In: 2016 IEEE International Conference on Robotics and Automation (ICRA). pp. 5469–5476. IEEE (2016)
38. Hayes, B., Shah, J.A.: Improving robot controller transparency through autonomous policy explanation. In: 2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI). pp. 303–312. IEEE (2017)
39. Herbert, S.L., Chen, M., Han, S., Bansal, S., Fisac, J.F., Tomlin, C.J.: FaSTrack: A modular framework for fast and guaranteed safe motion planning. In: 2017 IEEE 56th Annual Conference on Decision and Control (CDC). pp. 1517–1522. IEEE (2017)
40. Hipp, J., Güntzer, U., Nakhaeizadeh, G.: Algorithms for association rule mining-a general survey and comparison. SIGKDD explorations **2**(1), 58–64 (2000)
41. Huang, S.H., Held, D., Abbeel, P., Dragan, A.D.: Enabling robots to communicate their objectives. Autonomous Robots **43**(2), 309–326 (Feb 2019). https://doi.org/10.1007/s10514-018-9771-0, https://doi.org/10.1007/s10514-018-9771-0
42. Katz, G., Barrett, C.W., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: An efficient SMT solver for verifying deep neural networks (2017). https://doi.org/10.1007/978-3-319-63387-9_5, https://doi.org/10.1007/978-3-319-63387-9_5
43. Kearfott, R.B.: Interval computations: Introduction, uses, and resources. Euromath Bulletin **2**(1), 95–112 (1996)
44. Kellert, S.H.: In the wake of chaos: Unpredictable order in dynamical systems. University of Chicago press (1993)
45. Kim, J., Rohrbach, A., Darrell, T., Canny, J.F., Akata, Z.: Textual explanations for self-driving vehicles (2018). https://doi.org/10.1007/978-3-030-01216-8_35, https://doi.org/10.1007/978-3-030-01216-8_35
46. Koul, A., Fern, A., Greydanus, S.: Learning finite state representations of recurrent policy networks (2019), https://openreview.net/forum?id=S1gOpsCctm

47. Levien, R., Tan, S.: Double pendulum: An experiment in chaos. American Journal of Physics **61**(11), 1038–1044 (1993)
48. Lipton, Z.C.: The mythos of model interpretability. arXiv preprint arXiv:1606.03490 (2016)
49. Liskov, B.: Keynote address-data abstraction and hierarchy. In: Addendum to the proceedings on Object-oriented programming systems, languages and applications (Addendum). pp. 17–34 (1987)
50. Liskov, B.H., Wing, J.M.: A behavioral notion of subtyping. ACM Transactions on Programming Languages and Systems (TOPLAS) **16**(6), 1811–1841 (1994)
51. Liu, B., Hsu, W., Ma, Y.: Mining association rules with multiple minimum supports. In: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 337–341 (1999)
52. Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E.: The Scratch programming language and environment. ACM Trans. Comput. Educ. **10**(4) (Nov 2010). https://doi.org/10.1145/1868358.1868363, `https://doi.org/10.1145/1868358.1868363`
53. Miller, T.: Explanation in artificial intelligence: Insights from the social sciences. Artificial Intelligence (2018)
54. Miller, T., Howe, P., Sonenberg, L.: Explainable AI: Beware of inmates running the asylum or: How i learnt to stop worrying and love the social and behavioural sciences. arXiv preprint arXiv:1712.00547 (2017)
55. Mohseni-Kabir, A., Rich, C., Chernova, S., Sidner, C.L., Miller, D.: Interactive hierarchical task learning from a single demonstration. In: Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction. pp. 205–212. ACM (2015)
56. Moore, R.E.: Interval analysis, vol. 4. Prentice-Hall Englewood Cliffs, NJ (1966)
57. Muggleton, S.: Inductive logic programming: issues, results and the challenge of learning language in logic. Artificial Intelligence **114**(1-2), 283–296 (1999)
58. Muggleton, S.H., Lin, D., Pahlavi, N., Tamaddoni-Nezhad, A.: Meta-interpretive learning: application to grammatical inference. Machine learning **94**(1), 25–49 (2014)
59. Neema, S.: Assured autonomy (2017), `https://www.darpa.mil/attachments/AssuredAutonomyProposersDay_Program%20Brief.pdf`
60. Palaniappan, S., Ling, C.: Clinical decision support using olap with data mining. International Journal of Computer Science and Network Security **8**(9), 290–296 (2008)
61. Papadimitriou, A., Symeonidis, P., Manolopoulos, Y.: A generalized taxonomy of explanations styles for traditional and social recommender systems. Data Mining and Knowledge Discovery **24**(3), 555–583 (2012)
62. Perera, V., Selveraj, S.P., Rosenthal, S., Veloso, M.: Dynamic generation and refinement of robot verbalization. In: 2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN). pp. 212–218. IEEE (2016)
63. Piatetsky-Shapiro, G., Frawley, W.: Knowledge discovery in databases (1991)
64. Pulina, L., Tacchella, A.: An abstraction-refinement approach to verification of artificial neural networks. In: International Conference on Computer Aided Verification. pp. 243–257. Springer (2010)
65. Ribeiro, M.T., Singh, S., Guestrin, C.: "Why should I trust you?": Explaining the predictions of any classifier (2016). https://doi.org/10.1145/2939672.2939778, `https://doi.org/10.1145/2939672.2939778`

66. Richardson, A., Rosenfeld, A.: A survey of interpretability and explainability in human-agent systems. In: XAI Workshop on Explainable Artificial Intelligence. pp. 137–143 (2018)
67. Roberts, M., Monteath, I., Sheh, R., Aha, D., Jampathom, P., Akins, K., Sydow, E., Shivashankar, V., Sammut, C.: What was i planning to do. In: ICAPS workshop on explainable planning. pp. 58–66 (2018)
68. Rosenthal, S., Biswas, J., Veloso, M.: An effective personal mobile robot agent through symbiotic human-robot interaction. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1. pp. 915–922. International Foundation for Autonomous Agents and Multiagent Systems (2010)
69. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
70. Shrager, J.: John laird, paul rosenbloom and allen newell, universal subgoaling and chunking: The automatic generation and learning of goal hierarchies. Artif. Intell. **32**(2), 269–273 (1987). https://doi.org/10.1016/0004-3702(87)90014-2, https://doi.org/10.1016/0004-3702(87)90014-2
71. Sreedharan, S., Madhusoodanan, M.P., Srivastava, S., Kambhampati, S.: Plan explanation through search in an abstract model space pp. 67–75 (2018)
72. Srikant, R., Agrawal, R.: Mining generalized association rules (1995)
73. Standardization, I.: Iso/iec 7498-1: 1994 information technology–open systems interconnection–basic reference model: The basic model. International Standard ISOIEC **74981**, 59 (1996)
74. Tennent, R.D.: The denotational semantics of programming languages. Commun. ACM **19**(8), 437–453 (Aug 1976). https://doi.org/10.1145/360303.360308, https://doi.org/10.1145/360303.360308
75. Thrun, S.: Extracting rules from artificial neural networks with distributed representations. In: Advances in neural information processing systems. pp. 505–512 (1995)
76. Vanschoren, J., van Rijn, J.N., Bischl, B., Torgo, L.: OpenML: Networked science in machine learning. SIGKDD Explorations **15**(2), 49–60 (2013). https://doi.org/10.1145/2641190.2641198, http://doi.acm.org/10.1145/2641190.2641198
77. Veloso, M.M., Biswas, J., Coltin, B., Rosenthal, S.: CoBots: Robust symbiotic autonomous mobile service robots. In: IJCAI. p. 4423 (2015)
78. Ventocilla, E., Helldin, T., Riveiro, M., Bae, J., Boeva, V., Falkman, G., Lavesson, N.: Towards a taxonomy for interpretable and interactive machine learning. In: XAI Workshop on Explainable Artificial Intelligence. pp. 151–157 (2018)
79. Walter, E., Jaulin, L.: Guaranteed characterization of stability domains via set inversion. IEEE Transactions on Automatic Control **39**(4), 886–889 (April 1994). https://doi.org/10.1109/9.286277
80. Wang, S., Pei, K., Whitehouse, J., Yang, J., Jana, S.: Formal security analysis of neural networks using symbolic intervals. In: 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018. pp. 1599–1614 (2018), https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi
81. Wasylewicz, A.T.M., Scheepers-Hoeks, A.M.J.W.: Clinical Decision Support Systems, pp. 153–169. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-319-99713-1_11, https://doi.org/10.1007/978-3-319-99713-1_11
82. Wellman, H.M., Lagattuta, K.H.: Theory of mind for learning and teaching: The nature and role of explanation. Cognitive development **19**(4), 479–497 (2004)

83. Wen, W., Callahan, J.: Neuralware engineering: develop verifiable ANN-based systems. In: Proceedings IEEE International Joint Symposia on Intelligence and Systems. pp. 60–66. IEEE (1996)
84. Wen, W., Callahan, J., Napolitano, M.: Towards developing verifiable neural network controller. Department of Aerospace Engineering, NASA/WVU Software Research Laboratory (1996)
85. Wen, W., Napolitano, M., Callahan, J.: Verifying stability of dynamic soft-computing systems (1997)
86. XAIP: XAIP 2018: Proceedings of the 1st Workshop on Explainable Planning (2018)
87. Xiang, W., Johnson, T.T.: Reachability analysis and safety verification for neural network control systems. CoRR **abs/1805.09944** (2018), `http://arxiv.org/abs/1805.09944`
88. Yasmin, M., Sharif, M., Mohsin, S.: Neural networks in medical imaging applications: A survey. World Applied Sciences Journal **22**(1), 85–96 (2013)

A Fanoos Extensions

Here, we elaborate further on-going work for extensions of Fanoos in three thrusts: operator-selection learning (1.1), more advanced data-driven predicate generation (1.2), and engineering improvements (1.3, 1.4, and 1.5).

1.1 Learning to Select Operators

Fanoos lends itself to the use of active learning to improve the operator-selection procedure, utilizing a proxy-user to help bootstrap the process. Initial developments include an approach that attempts to balance exploration and exploitation, use a priori knowledge of the process and potentially the domain, and leverage structure of the states.[25] We are in the process of carrying out and refining experiments; we hope to report on this work in the near future.

1.2 Predicate Generation

We view the predicate generation problem as important, but distinct from the challenge Fanoos attempts to tackle and one that is amenable to many solutions in context (for instance, the methods we adopted in Section 4.1 backed by elementary statistics). Importantly, we do not believe requiring predicates simply pushes our core problem to a different arena. Our belief is that Fanoos addresses a large swath of the desiderata and resulting pipeline while allowing improvements to be plugged in for specific subproblems - subproblems with reasonably mature literature, existing methods, and communities working there-in to provide enhancements. Grounded predicate generation is one such subproblem. To elaborate with illustrative examples: databases are considered sensible solutions despite the fact that query optimizers can often be improved, while it is not sensible to say that the traveling salesman problem has been solved on the basis of it reducing to the knapsack problem; we view Fanoos's relationship to the predicate generation problem as similar to the former, not the latter. That noted, we ultimately want an entire system that produces well-tailored explanations for ML users while requiring as little human effort as possible - particularly effort that is unintuitive or requires uncommon expertise. Achieving this greater ease of deployment in more circumstances requires making the subproblems - such as grounded predicate generation - more effortless, more often.

On this front, we are further investigating learning techniques for the generation of domain predicates. Among these, Generalized Hough Transforms [9] were an early candidate, while more recently our attention has been drawn to advancements in inductive logic programming, Metagol [58, 18] chief among them. These methods have the potential to provide representational flexibility, predicates amenable to human review, intuitiveness of the extrapolations that may

[25] While these three desiderata have tangled interrelations, they do not seem necessarily redundant: the first two seem possible without the third in a problem possessing bandit-esque labeled levers and an opaque oracle, while the first and third seem able to exclude the second if one simply has rewards and distance measures between states.

be necessary in the process of generating predicates, and data-efficiency. With Metagol there also is potential for incorporating human-desirable invariants into predicates automatically, and handle input data with a spectrum of structure; in our experiments thus far, however, non-trivial effort has been required to handle numerical values as desired.

It is particularly desirable to enable the generation of predicates that are invariant under certain types of user-defined transformations. For example, one might learn the concept of closing a left-gripper and generalize it to closing at least one gripper by making the truth of the predicate invariant to renaming of hands. We might learn the concept of a sharp turn by only observing left turns and ensure that the hypothesis produced did not become curtailed to leftness by enabling invariance under rotation and reflection (possibly as a wrapper function that converts turns, using these transforms, to left turns). To keep such an approach both practical and meaningful, we must restraint its scope to not venture into more general AI challenges, such as attempting to solve the whole of transfer learning. Implementing invariance relating to naming, copying, and basic geometric transformations seems naturally desirable and not unmanageable in common use cases, and is thus one of the sub-focuses of our research.

The space of possible solutions in the modern ML landscape to engineer predicate generation is vast, particularly if one is willing to admit predicates that work reliably in practice but are not necessarily perfect in all circumstances.[26] For instance, one might utilize an oft-deployed vision-based recognition technique to pick-out objects reliably, despite it not being adversarially robust; indeed, this is not dissimilar to the approach adopted by [45], for instance. There might not be a panacea to the predicate generation challenge when it is conceived too broadly, especially if one insists on retaining certainty, transparency, and data-efficiency while attempting to capture sophisticated concepts with minimal human effort. Even in the face of this, we maintain that it is reasonable to suppose that understanding of a target system can most often (in practice) be improved through analysis/ decomposition utilizing these potentially imperfect materials.[27]

Given this overview of how predicate generation may be supported, it is worth taking a moment to reflect on whether predicates of this style are fundamentally necessary to the project of ML explainability. Not all XAI systems may require *explicit* semantic-grounding mechanisms that are *separate* from the learned system being analyzed—for instance, ranking features by weight,[28] generic salience maps, LIME [65], and finding exemplar datum [41] do not seem to utilize such mechanisms under casual examination. Further, the non-necessity of these ele-

[26] This might involve, for instance, relaxing one's confidence in a predicate's reliability and/or asking Fanoos questions about usual behavior instead of worst-case behavior.

[27] It may be necessary to conduct the analysis recursively on components.

[28] Whether this counts as an explanation, particularly if the features fail to have clear meaning to the consumer, is debatable. Further, one could rank the weights of the polynomial regression we experimented on, but that would fail to address all of the difficulties we highlighted in Section 4.1.

ments seems intuitively true[29] if we momentarily indulge in considering how a person explains their behavior—though of course people produce explanations of questionable truthfulness, reliability, and accuracy when it comes to both daily activity and deeper psychological phenomena. Necessity aside, explicit grounding approaches in the vein of this subsection are not uncommon in XAI efforts, and for the foreseeable future will most likely have fundamental benefits and drawbacks compared to the alternative.

1.3 Reachability Analysis Performance

The reachability analyzer of Fanoos is designed in a generic fashion and amenable to having its implementation swapped out without fundamentally altering the overall approach. While reachability analysis is in principle computationally expensive, there are many algorithms that have undesirable worse-case bounds in theory—for example SAT-solvers and the simplex algorithm—but routinely demonstrate useful performance in practice. Methods to potentially draw upon for engineering improvements include the reachability toolboxes CORA [5] and SpaceEx [28], as well as FaSTrack [39, 29], a safe planning framework that addresses a related family of problems; all have pushed forward the frontier of practical applications.

1.4 The Sampling Process Backing "Usually True" Queries

Our sampling method can be modified to user-provided distributions, such as estimates of the typical input distribution. Currently, we use a uniform distribution over the hypercubes where the condition is potentially true; this can help examine the range of behaviors/circumstances to a fuller extent and enable counter-factual reasoning. We can envision both sampling approaches as providing distinct values in practice, and thus both potentially useful over the course of dealing with a learned system.

Note that areas of the input space that are logically impossible and have non-zero measure under the uniform measure can be ignored by instructing Fanoos to add a predicate to "what do you usually do when ..." queries and subsequently filtering by that predicate when forming boxes in response to "when do you usually ..." queries.[30] In the output space, Fanoos attempts to deal with the pushforward of the provided learned component under the input distribution. It is worth noting, then, that for some applications, Fanoos in a sense characterizes what the learned component "attempts to do" or "signals" as opposed to what occurs down-stream, which would not atypically involve additional contributors to the system outside of Fanoos's purview[31]. For instance, Fanoos may be able

[29] We note that "intuitive" here is not cause for added certainty, since people's intuitions about the organization of cognition, etc., are not always spot-on.

[30] Both of these operations could be supported in a push-button fashion under the hood of a UI.

[31] We do not rule-out a user developing a model for down-stream behavior and incorporating it into something visible to Fanoos—arguably our experiments show signs

to say that the controller of an autonomous vehicle "attempts a hard-stop", but it may be the case that the vehicle as a whole exhibits a non-identical behavior due to ice on the road, unmodeled performance of low-level controllers, or other conditions either internal or external to the machine.

1.5 Fanoos's User Interface

Our focus while developing Fanoos has been to ensure that the desired information can be generated. In application, a thin front-end can be developed to provide a more aesthetically pleasing presentation, using the vast array of infographics and related tools available. Various input-output wrapper packages from the extensive literature on decision support systems may provide useful guidelines, as well as particularly promising works in interactive robotics (for instance, [55]). Presenting results as English-like sentences using templates (similar to [38]), as bar-graphs sorted on height, or word-clouds emphasizing relative importance are all easily facilitated from Fanoos's output. For input, template-supported English-like phrases, drag-and-drop flowcharts, [32] or even basic HTML drop-down menus with sub-categories for option filtering[33] are all possibilities. Although certainly important in practical application, in this paper we rather focus on presenting the underlying algorithmic contributions than the user-facing presentation.

B Fanoos Structural Overview

Fig. 2 illustrates the component interactions in Fanoos. Sections detailing the component are italicized. Components requiring user interaction are oval, internal modules are rectangular, and the knowledge database cylindrical.

C Extended Example User Interactions

We present here a typical user interaction with our system, Fig. 3. The interested reader can find the predicate definitions with the code at `https://github.com/anon-73bea4a3`. In practice, if users want to know more about the operational meaning of predicates (e.g., the exact conditions that each predicate tests for), these are revealed in the domain specification[34]—a large part of the point of this system is to provide functionality beyond just cross-referencing code.

Notice that our code uses a Unix-style interaction in the spirit of the `more` command, so not to flood the screen beyond a preset line limit. We also support auto-complete, finishing tokens when unambiguous and listing options available

of that by having actuator limits for the inverted double-pendulum—but naturally this is not something to assume is always present in sufficient fidelity.

[32] A particularly user-friendly and open-source example being [52].

[33] Either grouped by variables of interest or some richer semantic notion.

[34] This is easily facilitated by open-on-click hyperlinks and/or hover text.

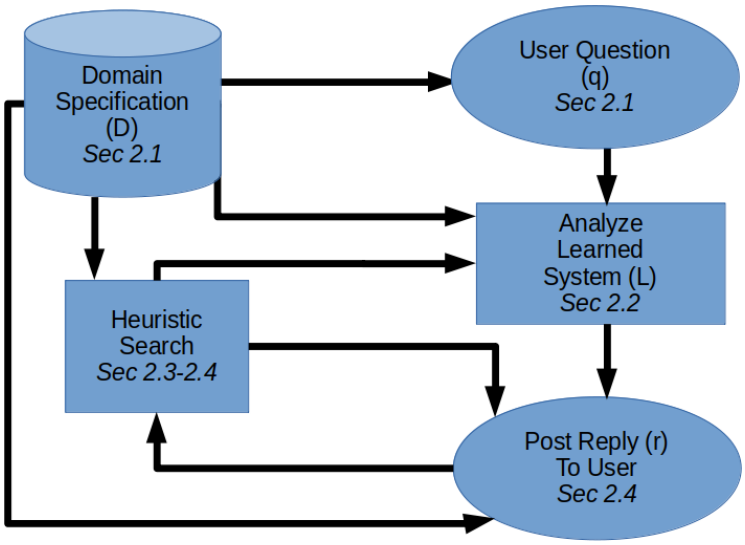

Fig. 2: User-interaction with Fanoos

in the context whenever the user hits tab.[35] Whenever we insert a comment in the interface trace that was not originally there, we put `//` at the beginning of the line.

We show the user posing two questions in the inverted double pendulum domain after providing the model they wished to analyze. The response to the first question by the user was a request for less abstraction, while the user response to the second was for greater abstraction. We see that in both cases the explanations adjusted as one would expect, both in respect to the verbosity of the descriptions returned and the verbiage used.

D Input-Space Bounding Box Values

In this section, we list the input-space bounding boxes used in our experiments. We list the values here up to four significant figures. Listings with further precision can be found in the code bases.

E Pseudo-Code for Specific-Selection Subroutine

[35] For instance, suggestions / completions for predicates obey restrictions imposed by Table 1 based on the question type thus-far specified by the user.

```
1  (Fanoos) when_do_you_usually and(
       outputtorque_low ,
       statevalueestimate_high )?
2  Enter a fraction of the universe box
       length to limit refinement to at the
       beginning.
3   Value must be a positive real number less
        than or equal to one.
```

User requests box length ↓ `0.125` 4

```
5  5 of 6 lines to print shown. Press enter
       to show more. Hit ctrl+C or enter
       letter q to break. Hit a to list all.
6  =========
7  //Description:
8  (0.45789160, 0.61440409, 'x Near Normal
       Levels')
9  (0.31030792, 0.51991449, '
       pole2Angle_rateOfChange Near Normal
       Levels')
10 (0.12008841, 0.37943400, '
       pole1Angle_rateOfChange High')
11 (0.06128723, 0.22426058, 'pole2Angle Low')
12 (0.02395519, 0.13633780, 'vx Low')a
13 (0.01147175, 0.01359231, 'pole1Angle Low')
14 type letter followed by enter key: b -
       break and ask a different question,
15  l - less abstract , m - more abstract, h
       - history travel
```

User requests less abstract, continue at (b) ↓ `l` 14

```
15 5 of 18 lines to print shown. Press enter
       to //[...]
16 =========
17 (0.16153820, 0.31093854, 'And(endOfPole2_x
        Near Normal Levels, pole1Angle Low,
       pole1Angle_rateOfChange High,
       pole2Angle Near Normal Levels,
       pole2Angle_rateOfChange High, x High)
       ')
18 (0.14268581, 0.18653883, 'And(endOfPole2_x
        Near Normal Levels, pole1Angle Low,
       pole1Angle_rateOfChange High,
       pole2Angle Near Normal Levels,
       pole2Angle_rateOfChange Near Normal
       Levels, x High)')
19 (0.11771033, 0.12043966, 'And(pole1Angle
       Near Normal Levels,
       pole1Angle_rateOfChange Near Normal
       Levels, pole2Angle High,
       pole2Angle_rateOfChange Low, vx Low)
       ')
20 (0.06948142, 0.07269412, 'And(pole1Angle
       High, pole1Angle_rateOfChange Near
       Normal Levels,
       pole2Angle_rateOfChange High, vx Low,
        x Near Normal Levels)')
21 (0.04513659, 0.06282974, 'And(endOfPole2_x
        Near Normal Levels, pole1Angle Low,
       pole1Angle_rateOfChange High,
       pole2Angle High,
       pole2Angle_rateOfChange Near Normal
       Levels, x High)')q
```

User break, continue at (c) ↓ `b` 22

(a) Initial question response, followed by request for less abstract explanation

(b) Less abstract explanation, user satisfied and continues with different question

```
23 (Fanoos) what_are_the_circumstances_in_which and(
       pole1angle_rateofchange_low__magnitude , outputtorque_high__magnitude )?
```

Fanoos answers ↓

```
24 5 of 32 lines to print shown. Press enter to //[...]
25 =========
26 (0.12099418, 0.18835537, 'pole2angle_rateofchange_high__magnitude')
27 (0.10147897, 0.17831770, 'And(pole1angle_on_the_left, pole2angle_on_the_left,
       pole2angle_rateofchange_low__magnitude)')
28 (0.09885232, 0.16335186, 'And(pole1angle_on_the_left, pole2angle_on_the_left,
       pole2angle_turning_counterclockwise)')
29 (0.07900125, 0.14467123, 'And(pole1angle_on_the_right, pole2angle_on_the_right,
       pole2angle_turning_clockwise)')
30 (0.06693577, 0.12822191, 'And(pole1angle_down, pole2angle_to_right,
       statevalueestimate_very_low)')q
31 type letter followed by enter key: b - break and ask a different question,
32  l - less abstract , m - more abstract, h - history travel
```

User requests more abstract ↓ `m` 28

```
29 3 of 3 lines to print shown.
30 =========
31 (0.44378316, 0.48588134, 'pole2 not near target position')
32 (0.33605014, 0.36551887, 'pole2angle_rateofchange_high__magnitude')
33 (0.22016670, 0.23739381, 'And(pole2angle_to_right, statevalueestimate_very_low)')
```

(c) Next question, initial response, and user request to make more abstract

Fig. 3: A sample user session with Fanoos on the inverted double pendulum example

Table 4: Inverted double pendulum input-space boxes

Variable name	Lower bound	Upper bound
x	-1	1
vx	-0.8	0.8
pole2_ endpoint[1]	-0.5	0.5
pole1angle	-0.2	0.2
pole1angle_ rateOfChange	-0.6	0.6
pole2angle	-0.04	0.04
pole2angle_ rateOfChange	-0.7	0.7

[1] *pole2_ endpoint* is a delta-value with respect to x. That is, in the observation given to the model to standardize, we add x to the value reported for *pole2_ endpoint*. This choice is motivated by the fact that the model was trained on the *pole2_ endpoint* position being measured in free-space, despite the fact that sensible values for this in an observation are highly dependent on x, the horizontal position of the cart's center.

Table 5: CPU Usage input-space boxes

Variable name	Lower bound	Upper bound
lread	0.0	0.0369
scall	0.0095	0.4245
sread	0.0028	0.0992
freemem	0.0061	0.6275
freeswap	0.4324	0.8318

input : box to fit, b; number of random samples to try, n; list of predicates to try, P; a list of strictly increase real numbers of length starting with 1.0, ℓ
output: A set of indices into P of the most specific predicates, s

$s \leftarrow \{\}$;
$dimsCovered \leftarrow \{\}$;
$bCenter \leftarrow getBoxCenter(b)$;
$bDim = getDimension(b)$;
for $i \leftarrow 0$ **to** $length(\ell) - 1$ **do**
  $lowerR \leftarrow \{\ell\}_i$;
  $upperR \leftarrow \{\ell\}_{(i+1)}$;
  $innerB \leftarrow ((b - bCenter(b)) \times lowerR) + bCenter(b)$;
  $outterB \leftarrow ((b - bCenter(b)) \times upperR) + bCenter(b)$;
  $randomSamples \leftarrow \{\}$;
  for $j \leftarrow 1$ **to** n **do**
    $randomSamples \leftarrow randomSamples \cup \{getRandVecBetweenBoxes(innerBox, outterBox)\}$;
  end
  for $pIndex \leftarrow 0$ **to** $length(P)$ **do**
    if $pIndex \in s$ **then**
      continue;
    end
    else if $freeVars(\{P\}_{pIndex}) \subseteq dimsCovered$ **then**
      continue;
    end
    foreach $v \in randomSamples$ **do**
      `/* We evaluate the predicate at index pIndex on v to see if it returns false */`
      if $\neg(\{P\}_{pIndex}).eval(v)$ **then**
        $s \leftarrow s \cup \{pIndex\}$;
        $dimsCovered \leftarrow dimsCovered \cup \text{freeVars}(P[pIndex])$;
        break;
      end
    end
  end
  if $length(dimsCovered) == bDim$ **then**
    return s;
  end
end
return s;

Algorithm 1: Find Most Specific Consistent Predicates. Note here that the notation $\{list\}_{index}$ is sequence-access notation, i.e, $\{< a_i | i \in [n] >\}_j = a_j$

Pseudo-code for our method of finding the most-specific conditions for a box are in Algorithm 1. In our code, we used $\ell = c\exp(\alpha \times c)$ where

$$c = [1.0, 1.01, 1.05, 1.1, 1.2, 1.4, 1.8, 2.6]$$

and α is a non-negative real-valued parameter we store and manipulate in the state. Similarly, n is stored in the states.

F Further Technical Descriptions and Details

6.1 Neural Networks, AIDs, and Propagating Boxes Through Networks

In this subsection, we discuss how we conduct our abstract interpretation domains (AIDs) analysis as applied to a feed-forward neural network. A crucial take-away from this discussion is that this process allows us to produce guarantees over uncountably infinite many elements by considering a finite collection of abstractions.

The abstract domains we consider for our AIDs are boxes. Boxes are easy to manipulate and easy to produce for bounding outputs, in contrast to more complex convex polytopes that would require solving constrains to determine boundaries and membership. This ease of manipulation comes at the cost of boxes being "less precise" than more sophisticated alternatives that have smaller volume while containing the same critical elements; larger sets suggest that certain relationships among variables may be feasible when in reality such combinations of values could not occur[36]. As a separate note, observe that the process we will describe is related to local reachability analysis, since we compute the image of an input set under a policy π.

The process described here leverages the fact that we are dealing with a pre-trained, fixed-weight feed-forward neural net, that has a typical MLP-like structure — namely, that the network consists of layers of units, each unit being comprised of a scalar-valued affine transformation of a subset of the previous layer's activations that is then passed through a non-decreasing (and typically non-linear) activation function to form the final output. In the case of recurrent neural nets, or other systems with loops, more sophisticated mechanisms — such as reachable-set fixed-point calculations (as discussed in [17], for instance) — would be necessary in general.

In order to propagate boxes through the network, we need to examine how to transform a box once it passes through a unit in the network. That is, if $u : \mathbb{R}^{I_u} \to \mathbb{R}^{O_u}$ is a unit of the network, and $\times_{i \in I_u}[a_i, b_i]$ is the input box, we need to calculate $u(\times_{i \in [I_u]}[a_i, b_i])$.

[36] In the case of interval arithmetic, this over-approximation and inclusion of additional elements is often called the "wrapping effect" [43].

For ease of discussion, we suppose that the network in question only has one type of activation function, ρ, which is non-decreasing ; our reasoning trivially extends to encompass networks with multiple such functions. For a unit u in the network, let $w \in \mathbb{R}^{I_u}$ be the weights of the unit, $\beta \in \mathbb{R}$ be the bias, and $x \in \mathbb{R}^{I_u}$ be the input value. We have that:

$$u_{linear}(x) =< w, x > +\beta$$
$$u_\rho(x) = \rho(< w, x > +\beta) = \rho(u_{linear}(x))$$

here, $< ., . >$ is the L2-norm. Notice that since ρ is a non-decreasing function, the maximum and minimum of $u_\rho(x)$ occurs for the same candidate x values as the respective max or min as $u_{linear}(x)$. To find the inputs that produce the extreme values of the activation function over the input space, it thus suffices to find the values in $\times_{i\in[I_u]}[a_i, b_i]$ that maximize or minimize $< w, x >$. Trivial algebra show that:

$$\underset{x\in\times_{i\in[I_u]}[a_i,b_i]}{argmin} < x, w >=< b_i\mathbb{1}(\{w\}_i \leq 0) + a_i\mathbb{1}(\{w\}_i > 0)|i \in [I_u] >$$

$$\underset{x\in\times_{i\in[I_u]}[a_i,b_i]}{argmax} < x, w >=< a_i\mathbb{1}(\{w\}_i \leq 0) + b_i\mathbb{1}(\{w\}_i > 0)|i \in [I_u] >$$

where here, $< z_i|i >$ is sequence construction notation. With this, we can compute the images of the input space under the activation functions as follows: for $u \in \{u_{linear}, u_\rho\}$,

$$u(\times_{i\in I_u}[a_i, b_i]) = [u(\underset{x\in\times_{i\in[I_u]}[a_i,b_i]}{argmin} < x, w >), u(\underset{x\in\times_{i\in[I_u]}[a_i,b_i]}{argmax} < x, w >)]$$

Having established how a box should be propagated through a unit in the network, it is trivial to extend this to propagating a box through the entire network. If $u_{i,j}$ is the i^{th} unit on the j^{th} layer, and $B_{i,j}$ is the input box to unit $u_{i,j}$, then one simply feeds the output box from one layer into the next as one would for a normal feed-forward network:

$$u_{i,j+1}(B_{i,j+1}) = u_{i,j+1}(\times_{h\in[I_{u_{i,j+1}}]}u_{h,j}(B_{h,j}))$$

Trivial induction would formalize this argument by tracing the propagation from the input layer all the way to the output layer of the network.

Various extensions to this line of reasoning exist. To close this subsection, we briefly discuss one example that may appear in a pre-processing or post-processing function. For instance, suppose we are given a network that is trained to output an action — but that the action must be properly post-processed by a function **properlyTransformAction** that scales and clips the output prior to finally passing the command to actuators. As often happens in practice, suppose that **properlyTransformAction** preserves partial orderings of vector inputs, so the lexical ordering of the outputs is never the reverse of the

lexical ordering of the inputs (i.e., if $\forall i \in [dim(\boldsymbol{x}_1)].\{\boldsymbol{x}_1\}_i \leq \{\boldsymbol{x}_2\}_i$ then $\forall i \in [dim(\textbf{properlyTransformAction}(\boldsymbol{x}_1))].\{\textbf{properlyTransformAction}(\boldsymbol{x}_1)\}_i \leq \{\textbf{properlyTransformAction}(\boldsymbol{x}_2)\}_i$). It is easy to see that this gives:

$$\Big(\forall w \in \textbf{properlyTransformAction}(\times_{i\in[I_u]}[a_i, b_i]).$$

$$\big(\forall h \in [dim(w)].\{\textbf{properlyTransformAction}(< a_i | i \in [I_u] >)\}_h \leq \{w\}_h$$

$$\leq \{\textbf{properlyTransformAction}(< b_i | i \in [I_u] >)\}_h. \ \big) \ \Big)$$

or, put another way, if

$$A = \textbf{properlyTransformAction}(< a_i | i \in [I_u] >)$$

and

$$B = \textbf{properlyTransformAction}(< b_i | i \in [I_u] >)$$

, then we have:

$$\textbf{properlyTransformAction}(\times_{i\in[I_u]}[a_i, b_i]) \subseteq \times_{i\in[dim(A)]}[\{A\}_i, \{B\}_i]$$

6.2 Our Counter Example Guided Abstraction Refinement (CEGAR) Inspired Process

In the model-checking world, CEGAR [16] is a well regarded technique for soundly ensuring a system meets desirable properties. In short, the approach uses intelligently adapted AIDs to attempt verification or refutation; in the case that the desirable property cannot be proven, the algorithm iteratively refines the abstraction based on where the property is in doubt, stopping when the property is either provable or has been disproven by a discovered counter example. When applied to certain families of discrete program, results returned by CEGAR are both sound and complete — this, however, comes at the cost of there not generally being a termination guarantee for CEGAR unless one is willing to allow sufficient approximations. In practice, approximations used with CEGAR tend to err on the side of safety— that if CEGAR indicates a property holds, then it is true, but the converse might not hold. This flexibility has allowed for extensions of the technique to many domains, including in hybrid system analysis [15], where the state space is necessarily uncountability infinite and system dynamics do not typically have exact numerically representations.

We now overview the CEGAR-like technique we implemented, using the abstraction domain described in Appendix 6.1 as a base. As before, we let π be a learned system $\pi : \mathscr{I} \to \mathbb{R}^{\mathscr{O}_\pi}$, where $\mathscr{I} \subseteq \mathbb{R}^{I_\pi}$ is the box $\times_{i\in[I_\pi]}[a_{\pi,i}, b_{\pi,i}]$ specifying the input space.[37] Let $\phi : \mathbb{R}^{I_\pi} \times \mathbb{R}^{\mathscr{O}_\pi} \to \{\top, \bot\}$ be a formula which we would like to characterize π's conformance to over $\mathscr{I}$ (i.e., find $\{(w, y) \in \mathscr{I} \times \mathbb{R}^{\mathscr{O}_\pi} | \phi(w, y) \wedge (y = \pi(w))\}$). Notice that ϕ need-not use all of its arguments — so, for instance, the value of ϕ might only vary with changes to input-space

[37] Or a superset of the input space .

variables, thus specifying conditions over the input space but none over the output space. Since CEGAR is not generally guaranteed to terminate (and certainly would not in our CEGAR-like implementation), we introduce a function **STOP** : $\mathbb{R}^{\mathscr{I}_\pi} \to \{\top, \bot\}$ which will be used to prevent unbounded depth exploration when determining whether counterexamples are spurious or not. The outline of the basic algorithm is shown in algorithm 2. As we will discuss in a moment, some steps that typical CEGAR might not be present, due to our slightly different aims.

The algorithm begins by forming an initial abstraction of the input space. In our implementation, the initial abstraction does not leverage any expert impressions as to what starting sets would be informative for the circumstances; instead, we opted for the simple, broadly-applicable strategy of forming high-dimensional "quadrants": 2^{I_π} hyper-cubes formed by bisecting the input space along each of its axes. The algorithm takes an input-abstraction, w, that has yet to be tried and generates a abstract state, approxOutputB, that contains $\pi(w)$.[38] If no member of w $\times$ approxOutputB is of interest (i.e., meets the condition specified by ϕ), the algorithm returns the empty set. On the other hand, if w $\times$ approxOutputB has the potential to contain elements of interest then the algorithm continues, attempting to find the smallest allowed abstract states that potentially include interesting elements. In general, further examination is performed by refining the input abstraction, then recursing on the refinements; for efficiency, we also check whether the entire abstract state satisfies ϕ, in which we are free to partition it into smaller abstractions without further checks.

While much of our process is in line with a canonical implementation of CEGAR , there are aspects which we have modified or did not need to implement for our purposes. For example, in a canonical implementation of CEGAR, whenever an AID is found that potentially violates the verification condition, concrete states within the offending AID are then examined in order to determine if the violation is spurious (that is, a result of wrapping/approximation effects from working in the abstract space, as oppossed to behaviour exhibited by the concrete system). Often, if a concrete counterexample is found, the analysis simply stops, but if it is spurious, the AID is refined. This arrangement does not necessarly fit our use case, however. While our process may be analogous or mappable to CEGAR and its standard extensions,[39] we will refer to our exact approach as a CEGAR-like process from here-on in order to avoid confusion over details or suggest a commitment to stringent canon adherence (e.g., using [16] verbatim).

Currently, our method of refinement is to split the box in question along the longest "scaled" axis. Rigorously, given a box to refine, $\times_{i\in[I_\pi]}[a'_i, b'_i]$, we form k-many new boxes as follows:

$$h = \underset{i\in[I_\pi]}{argmax} \frac{b'_i - a'_i}{b_{\pi,i} - a_{\pi,i}}$$

[38] Notice here that w and $\pi(w)$ are both sets.

[39] For example, sampling-based feasibility checks we perform prior to calling a sat-checker on verdict$_1$ and verdict$_2$ in algorithm 2 may be comparable to the spuriousness check.

$$C_k = \frac{b'_h - a'_h}{k}$$

$$\textbf{refine}_k(\times_{i\in[I_\pi]}[a'_i, b'_i]) = \cup_{j=0}^{k-1}\Big\{\times_{i\in[I_\pi]}[\ a'_i{+}\mathbb{1}(i=h)jC_k,\ \ b'_i{+}\mathbb{1}(i=h)(j{+}1{-}k)C_k\ ]\Big\}$$

In our current implementation, we select k at random each time we refine, selecting $k = 2$ with probability 0.8 and $k = 3$ with probability 0.2; this was motivated out of an attempt to balance between the desires for faster CEGAR analysis, reasonable abstract state sizes, diversity of abstract states, minimal growth of the abstract state space, and to mitigate against some potential pathological/ adversarial instances. The use of $b_{\pi,i}{-}a_{\pi,i}$ in the denominator for h is an attempt to control for differences in scaling and meaning among the variables comprising the input space. For instance, 20 millimeters is not commiserate with 20 radians, and our sensitivity to 3 centimeters of difference may be different given a quality that is typically on par of kilometers versus one never exceeding a decimeter.

Our analysis used the following **STOP** function, motivated for similar reasons as **refine** from Appendix 6.1:

$$\textbf{STOP}(\times_{i\in[k]}[a'_i, b'_i]) = (\ \ (\underset{i\in[I_\pi]}{argmax}\frac{b'_i - a'_i}{b_{\pi,i} - a_{\pi,i}}) \leq \epsilon\ \ )$$

Here, ϵ is the refinement parameter initially specified by the user, but which is then automatically adjusted by operators as the user interactions proceed.

The pseudocode in algorithm 2 shows the process for the formally sound question types. For the probabilistic questions types (i.e., those denoted with "...usually..."), verdict_1 is determined by repeated random sampling, and verdict_2 is fixed as $\perp$. In our implementation, feasibility checks are done prior to calling the sat-solver when handling a formally sound question-type; we spare a through description of efficiency-related modifications for the sake of presentation clarity.

```
1  Function CEGARLikeAnalysis(inputB , STOP, φ, π):
2      approxOutputB ← approxImage_π(inputB);// AIDs-based image approx.; see App.6.1
3      verdict_1 ← satSolverCheck(∀x ∈ inputB × approxOutputB.¬φ(x).);
4      if verdict_1 then
5          return {} ;
6      end
7      if STOP(inputB) then
8          return {inputB};
9      end
10     verdict_2 ← satSolverCheck(∀x ∈ inputB × approxOutputB.φ(x).) ;
11     if verdict_2 then
12         boxesToRefine ← {inputB};
13         boxesToReturn ← {};
14         while |boxesToRefine| > 0 do
15             thisB ← boxesToRefine.pop();
16             if STOP(thisB) then
17                 boxesToReturn ← boxesToReturn ∪ {thisB};
18             end
19             else
20                 boxesToRefine ← boxesToRefine ∪ refine(thisB);
21             end
22         end
23         return boxesToReturn;
24     end
25     refinedBoxes ← refine(inputB);
26     return ∪_{b∈refinedBoxes} CEGARLikeAnalysis(b , STOP, φ, π);
```

Algorithm 2: Pseudocode for our CEGAR-like analysis. Here, inputB is a AID element over the input space (i.e., inputB $\subseteq \mathscr{I}$).

6.3 Further Details on Automatic Predicate Filter Operator Currently Implemented

In this subsection, we provide a slightly more thorough description of the optional operator implemented in Fanoos which, when utilized, automatically determines a predicate to remove from consideration while forming a new description. For ease of statement, we will refer to the operator in question as APS (for "automated predicate selector") in this subsection.

Let S_T be the state whose description, D_T, the user currently wants altered. Let $Q(s)$ be the specific question instance[40] for which, in the process of producing replies, a state s was generated. To determine which named predicate occurring in D_T to remove, APS examines records of previous interactions to select a candidate that best balances exploration with exploitation.[41] Given a state that occurred in the past, S_t, let:

- $\omega(S_t, p)$ be the number of times a named predicate, p, occurs in the description of state S_t. This may be greater than one if, for instance, p occurs in multiple conjuncts.
- $rm(S_t)$ and $rl(S_t)$ be predicates indicating that the user requested the description to become, respectively, more abstract and less abstract (rm: "request more")
- $rb(S_t)$ indicate that the user requested to exit the inner QA-loop (i.e., "b" in listing 1.3) after seeing S_t's description (rb : "request break")

Further, let $r_T = rm$ and $r_{T+1} = rl$ if the user requested that D_T (the current description) become more abstract, and $r_T = rl,\ r_{T+1} = rm$ if the user requested lower abstraction. The predicate which APS removes is determined using the index returned by

$$\text{UCB}(\ \langle |\text{occ}(p)| \ \big|\ p \text{ occurs in } D_T \rangle,\ \langle |\text{succ}(p)| \ \big|\ p \text{ occurs in } D_T \rangle\ )$$

where UCB is the deterministic Upper Confidence Bound algorithm [8] and

$$\text{occ}(p) = \{S_t \in \text{history} | r_T(S_t) \wedge (\omega(S_t, p) > \omega(S_{t+1}, p))\}$$

$$\text{succ}(p) = \{S_t \in \text{occurs}(p) | r_{T+1}(S_{t+1}) \vee rb(S_{t+1})\}$$

where "$S_t \in$ history" is a slightly informal reference to accessing S_t from *all* previous interaction records (i.e., not just replies about $Q(S_t)$ or records from this user session). S_{t+1} indicates the state that followed S_t *while responding to the same question,* $Q(S_t)$ (i.e., it is not simply any state that comes chronologically after S_t in database records); in the cases where S_{t+1} does not exist, we substitute infinity for $\omega(S_{t+1}, p)$, and false for both $r_{T+1}(S_{t+1})$ and $rb(S_{t+1})$.

While alternatives to the adopted method could be used — particularly approaches with greater stochasticity — we believe our choice of a UCB algorithm is most likely appropriate at this stage, considering data efficiency and the likely nature of the environment.[42] Future improvements or novel operators may introduce different or more sophisticated methods for predicate selection.

---

[40] Here, if the same question is asked later, it is considered a different instance.

[41] Exploration: trying the available options often enough to be informed of each potential outcome; Exploitation: choosing the option that will most likely result in the outcome the user requested — changing the abstraction level in the desired direction.

[42] For instance, we do not expect an adversarial environment, nor do we expect — provided the history of states/replies responding to $Q(S_T)$ — explicit time dependence.

6.4 Pseudocode for the Generation Process after Determining Boxes to Describe

In our pseudocode, $\mathscr{B}$ is the set of all boxes that are subsets of $\mathscr{I}$ (see Appendix 6.2), and $\rightharpoonup$ denotes a partial function.

Note that this code assumes pass by copy, not pass by reference.

Further, assume any parameters not passed into the function arguments are accessed through a globally accessible state.

Finally, as noted in the main write-up, the boxes being described here may have undergone some merging after discovery (i.e., post-CEGAR), but prior to reaching here. Such merging attempts to increase the abstraction level while retaining some finer-details; it is not the case in general that increasing ϵ in the **STOP** function from Appendix 6.2 would produce the same results.

```
/* Input: Bs is the collection of boxes found by our CEGAR-esque process after any
   desired post-discovery merging, Cs is the collection of conditions (i.e., named
   predicates or conjunctions of them) that may be used to produce descriptions, and
   produceGreaterAbstraction is a boolean parameter stored in the state that
   operators may toggle. nSample is an integer determining the number of
   random-sample feasibility checks to do prior to calling a sat-solver, a procedure
   only to aid efficiency.                                                          */
/* Output: a description in DNF form, weights for each conjunct in the DNF formula
   representing unique coverage of each conjunct, weights representing the total
   coverage of each conjunct (including possible redundancies), and the list of
   conditions after any additions formed during the description generation (i.e.,
   new conjunctions or box-range predicates) .                                      */
Function generateDescription( Bs , nSample , Cs , produceGreaterAbstraction):
    if |Bs| == 0 then
        throw exception("No Situation Corresponds to the Event User Described Occurring");
    end
    ( Cs_2 , csAndBs , bsAndGoodCs ) ← getInitialListOfConditionsConsistentWithBoxes( Bs,
      nSample, Cs, produceGreaterAbstraction) ;
    coveringCs ← getApproximateMultivariateSetCover( bsAndGoodCs );
    // The below line is needed because coveringCs may contain new conjuncts.
    Cs_3 ← Cs_2 ∪ coveringCs;
    Make: csToBs : Cs_2 ⇀ ℬ
    s.t. ∀c ∈ Cs_2.∀b ∈ ℬ.( (b ∈ csToBs(c)) ⟺ ((c, b) ∈ csAndBs) ) ;
    csToBs_2 ← handleNewConjuctions( Cs_3 , csToBs );
    Make: bsToCs_2 : ℬ ⇀ coveringCs
    s.t. ∀b ∈ ℬ.∀c ∈ coveringCs.( (c ∈ bsToCs_2(b)) ⟺ (b ∈ csToBs_2(c) ));
    /* Below, a second covering is done in order to account for predicates that
       cover a superset of the boxes of another predicate returned in coveringCs.
       Notice that this would not necessarily have been handled by the previous call
       to getApproximateMultivariateSetCover since there we only listed which boxes
       a predicate was among the most specific for, not the the total set of boxes it
       was consistent with. So, for instance, it is possible that a box was only
       describable by one vague predicate in the results from the first covering -
       this covering would handle the fact that the vague predicate may imply the
       behaviour of many of the other specific predicates. See earlier in the
       write-up for how we avoid this ''washing-out'' finer-grained detail.        */
    coveringCs_2 ← getApproximateMultivariateSetCover(bsToCs_2);
    Cs_4 ← Cs_3 ∪ coveringCs_2;
    csToBs_3 ← handleNewConjuctions( Cs_4, csToBs );
    (csToTV, csToUV) ← getVolumesCoveredInformation( Bs, coveringCs_2, csToBs_3);
    coveringCs_3 ← removePredicatesImpliedByOthers( coveringCs_2, csToBs_3, csToUV );
    (coveringCs_4, Cs_5, csToBs_4) ←
      handleNewInstancesOfBoxRangePred(coveringCs_3, Cs_4, csToBs_3)
    (csToTV_2, csToUV_2) ← getVolumesCoveredInformation( Bs, coveringCs_3, csToBs_4 );
    /* Note that, when presenting results to users, we primarily use elements from
       coveringCs_4 to index into other structures. Thus, it is acceptable if other
       structures happen to have domains that are supersets of coveringCs_4.       */
    return (coveringCs_4, Cs_5, csToTV_2 , csToUV_2) ;
```

Algorithm 3: Pseudocode for the description generation after boxes to described have been determined.

```
26 Function getInitialListOfConditionsConsistentWithBoxes(Bs, nSample, Cs,
     produceGreaterAbstraction):
27     Cs_2 ← Cs;
28     csAndBs ← {};// csAndBs ⊆ Cs_2 × Bs
29     bsAndGoodCs ← {}; // bsAndGoodCs ⊆ B × Cs_2
30     for b ∈ Bs do
31         CsForThisB ← getConsistentConditions(b, nSample, Cs);
32         for c ∈ CsForThisB do
33             csAndBs ← csAndBs ∪ {(c, b)};
34         end
35         mostSpecificCsForThisB ← getMostSpecificCondition(b, nSample, CsForThisB);
36         if (mostSpecificCsForThisB == Null) ∨ (|CsForThisB| == 0) then
37             if (¬ produceGreaterAbstraction) ∨ (|CsForThisB| == 0) then
38                 newBoxP ← createBoxPredicate(b);
39                 Cs_2 ← Cs_2∪ { newBoxP };
40                 csAndBs ← csAndBs ∪ { (newBoxP, b) };
41                 bsAndGoodCs ← bsAndGoodCs ∪ { (b,newBoxP) };
42             end
43             else
44                 bsAndGoodCs ← bsAndGoodCs ∪ { (b,c) | c ∈ CsForThisB };
45             end
46         end
47         else
48             bsAndGoodCs ← bsAndGoodCs ∪ {(b, c) | c ∈ mostSpecificCsForThisB };
49         end
50     end
51     return ( Cs_2 , csAndBs , bsAndGoodCs );
```

Algorithm 4: Pseudocode for determining which predicates cover boxes and which are most specific.

```
52 Function getConsistentConditions(b, nSamples, ps):
53     samples ← {};
54     for i ← 1; i <= nSamples; i ← i + 1 do
55         samples ← samples ∪ {randomVectorInBox(b)};
56     end
57     candidatePs = {};
58     for thisP ∈ ps do
59         success ← ⊤;
60         for thisS ∈ samples do
61             if ¬ thisP( thisS) then
62                 success ← ⊥;
63                 break;
64             end
65         end
66         if success then
67             candidatePs ← candidatePs ∪ {thisP};
68         end
69     end
70     consistentPs ← {};
71     for thisP ∈ candidatePs do
72         verdict ← satSolverCheck(∀v ∈ b.thisP(v));
73         if verdict then
74             consistentPs ← consistentPs ∪ {thisP};
75         end
76     end
77     return consistentPs;
```

Algorithm 5: Helper function for algorithm 4

```
78 Function getApproximateMultivariateSetCover(bsAndPs):
79     (bsToPs, psToBs) ← startFunct(bsAndPs) ;
80     filteredDomainMaxPsToBs ← getMaxCover(bsToPs, psToBs);
81     setsOfSetOfPsToConjunct ← reverseOut(filteredDomainMaxPsToBs);
82     conditionList ← couplePs( setsOfSetOfPsToConjunct ) ;
83     return conditionList ;

84 Function startFunct(bsAndPs):
85     Make:  bsToPs : ℬ ⇀ 𝒫(AllPreds)
86     Initailize such that:  ∀b ∈ ℬ. bsToPs(b) = {p ∈ AllPreds|(b, p) ∈ bsAndPs};
87     Make:  psToBs : AllPreds ⇀ 𝒫(ℬ)
88     Initailize such that:  ∀p ∈ AllPreds. psToBs(p) = {b ∈ ℬ|(b, p) ∈ bsAndPs};
89     return (bsToPs, psToBs) ;

90 Function getMaxCover(bsToPs, psToBs):
91     Make:  maxPsToBs : AllPreds ⇀ 𝒫(ℬ);
92     Initialized so that:  ∀p ∈ AllPreds. maxPsToBs(p) = {}.;
93     Make:  bsToCoveredXs : ℬ ⇀ 𝒫(SetOfVariables);
94     Initialized so that:  ∀b ∈ ℬ. bsToCoveredXs(b) = {};
95     while |domain(bsToPs)| > 0 do
96         maxP ← Null; // empty value for now
97         bsCoveredByMaxP ← {};
98         for thisP ∈ domain(psToBs) do
99             if  (|psToBs(thisP)| > |bsCoveredByMaxP|)∨
100            ( (|psToBs(thisP)| == |bsCoveredByMaxP|) ∧ (x > 0.5 where x ∼ Uniform([0, 1])) )
                then
101                maxP ← thisP;
102                bsCoveredByMaxP ← psToBs(thisP);
103            end
104        end
105        maxbsAndPs(maxP) ← maxbsAndPs(maxP) ∪ {bsCoveredByMaxP} ;
106        xsCoveredByMaxP ← freeVars(maxP) ;
107        for thisB ∈ bsCoveredByMaxP do
108            bsToCoveredXs(thisB) ← bsToCoveredXs(thisB) ∪ xsCoveredByMaxP ;
109            psNowCoveringThisB ← bsToPs(thisB) ;
110            for thisP ∈ psNowCoveringThisB do
111                if  ¬ ( freeVars(thisP) ⊆ bsToCoveredXs(thisB) ) then
112                    continue ;
113                end
114                psToBs(thisP) ← psToBs(thisP) \ {thisB};
115                bsToPs(thisB) ← bsToPs(thisB) \ {thisP};
116            end
117            if  bsToPs(thisB) == {} then
118                bsToPs ← bsToPs ↾ (domain(bsToPs) \ {thisB})
119            end
120        end
121    end
122    Make: filteredDomainMaxPsToBs :  AllPreds2 ⇀ 𝒫(ℬ),   AllPreds2 ⊆ AllPreds
123    s.t. filteredDomainMaxPsToBs = maxPsToBs ↾ {p ∈ AllPreds|maxPsToBs(p) ≠ {}};
124    return filteredDomainMaxPsToBs;
```

Algorithm 6: Pseudocode for the multi-dimensional set covering we conduct.

```
125 Function reverseOut(fDMaxPsToPs):
126 |  /* Below, each element is exactly a set of predicates used to cover a member of
    |     B.                                                                        */
127 |  /* SSPC stands for ''set of set of predicates to conjunct''                   */
128 |  SSPC ← {ps ∈ 𝒫(AllPreds)|
129 |     ∃b ∈ B. ∀p ∈ domain(fDMaxPsToBs).(b ∈ fDMaxPsToBs(p) ⟺ (p ∈ ps))};
130 |  return SSPC;

131 Function couplePs(SSPC):
132 |  resultCs ← {};
133 |  for thisSetOfPs ∈ SSPC do
134 |  |  if |thisSetOfPs| > 1 then
135 |  |  |  resultCs ← resultCs ∪ {∧_(p∈thisSetOfPs) p};
136 |  |  end
137 |  |  else
138 |  |  |  resultCs ← resultCs ∪ thisSetOfPs;
139 |  |  end
140 |  end
141 |  return resultCs ;
```

Algorithm 7: Continuation of algorithm 6, helper functions for the multi-dimensional covering process.

```
// Reminder: this pseudocode does not mutate the caller's copy of arguments
Function handleNewConjuctions(Cs , csToBs):
    for thisC ∈ Cs do
        if thisC ∈ domain(csToBs) then
            continue ;
        end
        /* Below, isAConjunct: returns ⊤ IFF thisC is of form ∧_{p∈ps} p for a set of
           predicates (atomic statements) ps                                          */
        if isAConjunct(thisC) then
            bsCoveredByThisC ← Null; // empty value to start
            // Below iterate of thisP where thisC = ∧_{thisP∈ps} thisP
            for thisP ∈ getAtoms(thisC) do
                bsCoveredByThisP ← csToBs(thisP);
                if bsCoveredByThisC == Null then
                    bsCoveredByThisC ← bsCoveredByThisP;
                end
                else
                    bsCoveredByThisC ← bsCoveredByThisC ∩ bsCoveredByThisP;
                end
                domain(csToBs) ← domain(csToBs) ∪ {thisC};
                csToBs(thisC) ← bsCoveredByThisC;
            end
        end
    end
    return csToBs ;
```

Algorithm 8: Pseudocode to properly adjust book-keeping structures after introduction of conjuncts during the covering process.

```
/* Below, ''Conditions'' (typically denoted with a ''c'' in naming) include atomic
   predicate( i.e., named predicates and box-range predicates) and conjunctions of
   atomic predicate.                                                              */
Function getVolumesCoveredInformation(B, Cs, csToBs):
    Make: csToBs : Cs ⇀ 𝒫(ℬ)
    s.t. ∀c ∈ ∪_{b∈domain(bsToCs)} bsToCs(b). csToBs(c) = {b ∈ ℬ | c ∈ bsToCs(b)};
    // For total volume
    Make:  csToTV : Cs ⇀ {r ∈ ℝ | r >= 0}
    Initialized so that ∀c ∈ Cs. cvToTV(c) = 0.0;
    /* Below, UV stands for ''unique volume'', the total volume a condition is
       responsible for uniquely covering. Depending on one's definition, it may be
       considered an approximation -- though sound lower bound -- for the ''unique''
       volume, since multiple predicates might cover the same box, but different
       axes (i.e., variables), a sort of ''uniqueness'' not accounted for by csToUV .
       Naturally, the alternate notion could be implemented using trivial,
       additional bookkeeping; this alternative, however, would likely be redundant
       with csToTV in most cases.                                                  */
    Make:  csToUV : Cs ⇀ {r ∈ ℝ | r >= 0}
    Initialized so that ∀c ∈ Cs. cvToUV(c) = 0.0;
    TV = 0.0 ; // total volume of all the boxes
    for thisB ∈ B do
        v ← computeVolume(thisB);
        TV ← TV + v;
        csCoveringThisB ← bsToCs(thisB);
        for thisC ∈ csCoveringThisB do
            csToTV(thisC) ← csToTV(thisC) + v;
            if |csCoveringThisB| == 1 then
                csToUV(thisC) ← csToUV(thisC) + v;
            end
        end
    end
    if TV == 0.0 then
        TV = 1.0;
    end
    Make:  csToTV_2 : Cs ⇀ {r ∈ ℝ | r >= 0}
    s.t. ∀c ∈ domain(csToTV). csToTV_2(c) = csToTV(c) / TV;
    Make:  csToUV_2 : Cs ⇀ {r ∈ ℝ | r >= 0}
    s.t. ∀c ∈ domain(csToUV). csToUV_2(c) = csToUV(c) / TV;
    return (csToTV_2 , csToUV_2);
```

Algorithm 9: Pseudocode for determining volume information.

```
196 Function removePredicatesImpliedByOthers(Cs, csTobs, csToUniqVols):
197     Bs ← ∪_{c∈Cs} csTobs(c);
198     boxCs ← {c ∈ Cs|c is a box-range predicate};
199     conjunctionCs ← {c ∈ Cs|c is a conjunction of atomic predicates};
200     othersCs ← Cs \ (boxCs ∪ conjunctionCs);
201     orderToConsider ←<>;// Note: <> is a sequence
202     /* Below: iterate from left-most (0-index) to right-most (max index), in order
           */
203     for s ∈ <boxCs, conjunctionCs, othersCs> do
204         // Below: cons(<A, B>, C) evaluates to <A, B, C>, i.e., append to back
205         orderToConsider ← cons( orderToConsider, formSortedList( listToSort ← s, sortKey ←
              csToUniqVols, order ← 'ascending') );
206     end
207     Cs_2 ← Cs;
208     bsToAlwaysCheck ← {};
209     // Below: as before, iterate from 0-index to max-index
210     for thisList ∈ orderToConsider do
211         for thisC ∈ thisList do
212             restOfCs ← Cs_2 \ {thisC};
213             bsToCheckHere ← bsToAlwaysCheck ∪ csTobs(thisC);
214             removeThisC ← ⊤ ;
215             for thisB ∈ bsToCheckHere do
216                 if ¬ disjunctCoversBox(thisB, restOfCs) then
217                     removeThisC ← ⊥ ;
218                     break ;
219                 end
220             end
221             if removeThisC then
222                 Cs_2 ← restOfCs ;
223                 bsToAlwaysCheck ← bsToCheckHere ;
224             end
225         end
226     end
227     return Cs_2 ;
```

Algorithm 10: Pseudocode to remove description elements that are implied by other parts of the description. Note that a particular predicate may cover a box that no other *individual* predicate covers - but the appropriate disjunction of other predicates might; this is exactly what we check for here. Some code for efficiency evoked prior to this stage - such as sound feasibility checks and faster-but-incomplete redundancy-removal functions (incomplete in the sense that not all redundancy can be addressed by them) - are not shown for clarity.

```
Function disjunctCoversBox(b, Cs):
    /* first, for efficiency, some random sampling prior to a formal check        */
    nSamples ← configurationFile.nSamplesPriorToFormalCheck ;
    for i ← 1; i <= nSamples; i ← i + 1 do
        v ← getRandomVectorInBox(b) ;
        noCHolds ← ⊤ ;
        for thisC ∈ Cs do
            if thisC(v) then
                noCHolds ← ⊥;
                break;
            end
        end
        if noCHolds then
            return ⊥
        end
    end
    // formal check with SAT-Solver
    formulaToCheck ← (∀v ∈ b. ∨_{thisC∈Cs} thisC(v));
    return satSolverCheck(formulaToCheck);
```

Algorithm 11: Psuedocode for helper-function to algorithm 10

```
Function handleNewInstancesOfBoxRangePred(coveringCs, Cs,
    dictMappingConditionToBoxesItIsConsistentWith):
    listOfCandidateBoxes ← {thisC ∈ coveringCs|thisC is a box-range predicate};
    Cs_2 ← Cs \ listOfCandidateBoxes ;
    coveringCs_2 ← coveringCs \ listOfCandidateBoxes ;
    newMergedBoxes ← mergeBoxes( ∪_{thisC∈listOfCandidateBoxes}{thisC.box} ,
     state.parametersForBoxMerging);
    newConditionsToAddIn ← ∪_{thisBox∈newMergedBoxes}{createBoxPredicate(thisBox)};
    for thisC ∈ newConditionsToAddIn do
        Cs_2 ← Cs_2 ∪ {thisC};
        coveringCs_2 ← coveringCs_2 ∪ {thisC};
        /* Below is a bit of a sub-preferred and slow way to do things - ideally we
           would keep track of this information when we did the original merges - but
           as-written, it is reasonable way to write it to get across the idea.
           Further, there would ideally not be too many boxes here, as hopefully most
           boxes would be addressed by named-predicates. Either way, the ideal way
           would be to have mergeBoxes returned the merged boxes and a set of tuples
           of form (original box, box in the new merger) to avoid the repeated work
           of the below line.                                                        */
        dictMappingConditionToBoxesItIsConsistentWith(thisC) ←
         {b ∈ listOfCandidateBoxes|firstBoxContainsSecondBox(thisC.box, b)} ;
    end
    return (coveringCs_2, Cs_2, dictMappingConditionToBoxesItIsConsistentWith);
```

Algorithm 12: Pseudocode for handling boxes that fail to be described by conditions (named-predicates and conjunctions of named-predicates) earlier in the process. In particular, we attempt to merge any boxes remaining at this phase prior to displaying to the user.

G Extended Comment on the Difficulties of Pretheoretical Notions of Abstractness

As commented on in Section 2.5, criteria to judge degree-of-abstractness in the lay sense are often difficult to capture, if even existent generally. For instance, one may ponder which is more abstract: a chair or the collection of atoms that constitute it? Does it matter if we consider only one atom in the chair as opposed to all the atoms? It is debatable which, if either, should be considered more abstract than the other in any absolute sense. The history of science suggests one ordering, while a casual, crude consideration of Platonism crossed with parts-of-whole relations suggests another way. We spare further consideration of this complexity to stave-off sidetracking too far in this work. At the risk of possibly invoking a platitude and sounding as though pragmatic answers are the ultimate answer,[43] we have already highlighted in Section 2.5 works that do point to some clear schemas and demonstrate functioning utilizations of *implementations of the concept* in computer science work. Regrettably, we will leave the discussion here as to: (1) whether a single notion of abstractness is applicable across all contexts, desires, collections of objects, and aspects of human life, (2) whether the chair-versus-atoms example may be inappropriate due to (a) mixing types or (b) context / subject matter irrelevant for the situations we consider in this work.

[43] A position we are not explicitly committing to or against at this stage.