– D3WA+ –
A Case Study of XAIP in a Model Acquisition Task for Dialogue Planning

Sarath Sreedharan[1*] · **Tathagata Chakraborti**[2*] · **Christian Muise**[3†]
Yasaman Khazaeni[2] · **Subbarao Kambhampati**[1]

[1]School of Computing, Informatics, and Decision Systems Engineering, Arizona State University · {ssreedh3, rao} @ asu.edu
[2]IBM Research AI · tchakra2@ibm.com, yasaman.khazaeni@us.ibm.com
[3]School of Computing, Queen's University · muise@cs.queensu.ca

Abstract

Recently, the D3WA system was proposed as a paradigm shift in how complex goal-oriented dialogue agents can be specified by taking a declarative view of design. However, it turns out actual users of the system have a hard time evolving their mental model and grasping the imperative consequences of declarative design. In this paper, we adopt ideas from existing works in the field of Explainable AI Planning (XAIP) to provide guidance to the dialogue designer during the model acquisition process. We will highlight in the course of this discussion how the setting presents unique challenges to the XAIP setting, including having to deal with the user persona of a domain modeler rather than the end-user of the system, and consequently having to deal with the unsolvability of models in addition to explaining generated plans.

Introduction

The state of the art (Sreedhar 2018) in the design of sophisticated *goal-directed* conversational agents – e.g. for applications such as customer support – requires the dialogue designer to either manually specify the entire dialogue plan (e.g. Google Dialogue Flow or Watson Assistant) or train end-to-end systems from existing logs of conversation. The former, of course, becomes intractable pretty soon which means conversational agents of any reasonable sophistication are still comfortably out of reach (Computer Generated Solutions 2018); while the latter provides no control over the emergent behavior of the agent, as seen in the infamous deployment of "Tay" (Metz 2018), and are thus unusable in the enterprise scene where customer experience has to be guaranteed to a reasonable certainty.

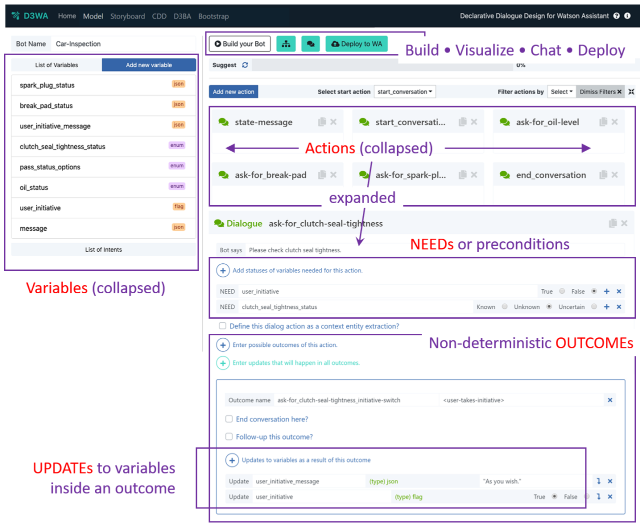
The topic of making the design of conversational agents easier for the dialogue designers – especially the task of making the specification of the conversation flow more tractable – remains of immense interest (Amazon 2019; Google 2019) to the AI community, and consequently to the world of automated planning as well due to its unique value proposition in the declarative specification of agents and the composition of higher order behavior from it. For example, recently authors of (Muise et al. 2019a; Botea et al. 2019) proposed a planning-based approach to bring down the effort in specification of such agents. These works present a paradigm shift in how goal-directed conversational agents can be designed using automated planning technology, demonstrating a tight synergy of symbolic and non-symbolic AI techniques to achieve a fully functional embodiment of reasoning, learning, and natural language processing under the same roof. We build on this work here and hence start with a brief description of the same below.

[*]Equal contributors.
[†]Work done while at [2]IBM Research (Cambridge).

A Brief History of D3WA

At the core of the declarative specification proposed in (Muise et al. 2019a) is an agent-centric view of the world – the dialogue designer specifies "capabilities" that are available to an agent and lets a non-deterministic planner generate (Muise, McIlraith, and Beck 2012) and execute (Muise et al. 2019b) the composed dialogue plan in the background. We demonstrated in ICAPS 2019 (Chakraborti et al. 2019b) how this can lead to an exponential scale-up from the size of the specification to the complexity of the composed agent, as an illustration of the exciting fusion of planning based technologies (especially non-deterministic planning) and the design of dialogue agents. This is especially useful in the design of certain kinds of conversations, especially ones with an underlying process – e.g. a business process (Chakraborti and Khazaeni 2020) – that drives the conversation. However, it turns out that while this provides a powerful tool for an experienced domain writer with expertise in planning, and declarative programming in general, for the uninitiated it presents too steep a learning curve. Since designers no longer explicitly compose the dialogue plan, they lose control over the composed agent if they do not grasp the *imperative consequences of their declarative specification.*

In this paper, we thus build on this work *with the aim of making the core domain authoring engine more amenable to dialogue designers who are usually outside the planning community and do not readily subscribe to the declarative mental model.* In order to do so, we build upon recent techniques from the explainable AI planning (XAIP) community (Chakraborti, Sreedharan, and Kambhampati 2020) to



Bot Name Car-Inspection

spark_plug_status	json
break_pad_status	json
user_initiative_message	json
clutch_seal_tightness_status	enum
pass_status_options	enum
oil_status	enum
user_initiative	flag
message	json

Select start action start_conversation
Filter actions by Select

state-message · start_conversati... · ask-for_oil-level · ask-for_break-pad · ask-for_spark-pl... · end_conversation

Dialogue ask-for_clutch-seal-tightness

Bot says Please check clutch seal tightness.

Add statuses of variables needed for this action.

NEED user_initiative True False [● False]
NEED clutch_seal_tightness_status Known Unknown Uncertain [● Unknown]

☐ Define this dialog action as a context entity extraction?

Enter possible outcomes of this action.
Enter updates that will happen in all outcomes.

Outcome name ask-for_clutch-seal-tightness_initiative-switch <user-takes-initiative>

☐ End conversation here?
☐ Follow-up this outcome?

Updates to variables as a result of this outcome

Update user_initiative_message (type) json "As you wish."
Update user_initiative (type) flag True False [● True]

Figure 1: Illustration of the salient aspects of declarative design of conversational agents on the D3WA interface, reproduced with permission from (Muise et al. 2019a).

bridge the gap with the end user. Before we get to the specific contributions of this paper, we start with a brief introduction to D3WA so as to make the rest of the presentation self-contained to the extent possible.

Actions, Outcomes and Context Variables The original system D3WA is illustrated in Figure 1. The interface surfaces two key elements to the dialogue designer: 1) **context variables** that model the agent's world; and 2) **actions** that are defined in terms of these variables. For a dialogue agent, these actions model the different capabilities available to it in terms of dialogue actions towards the end user, or internal system actions such as API calls or logical inferences.

Each action (Figure 1) has a set of NEEDs (or preconditions) and a set of OUTCOMEs which house a set of non-deterministic UPDATEs to a variable. The outcomes within an action are mutually exclusive and model how the user may respond or, in general, how the world may evolve in response to any action done by the agent. For example:

1. *Dialogue Action:* If the agent wants to ask the user their name, the corresponding dialogue action would have one outcome when the user responds with their name, and another one that models a digression in the conversation.
2. *System Action:* In order to make an API call, the agent would require as NEEDs, access to the link and the relevant payload. Two possible outcomes of the call may be a successful response (in which case the agent updates the values of the relevant variables it was looking for) or a 404 error (in which case the agent gets nothing).

A non-deterministic planner receives this specification, plans for all possible outcomes, and generates the resulting dialogue plans in Figure 1. This offline approach has two advantages: it allows the dialogue designer to inspect and sign off on the agent to be deployed, while also being able to support complex dialogues without having to plan and replan in real time. For more details on D3WA, and on how this specification is compiled to a planning problem in the background, we refer the reader to (Muise et al. 2019a).

D3WA + XAIP → D3WA+

The "explainable" version of D3WA – henceforth referred to as D3WA+ – developed in this paper provides a suite of debugging tools on top of the existing core model acquisition framework described above. This is aimed to make the dialogue designer more self-sufficient when they are faced with modeling errors. Specifically, based on difficulties observed during preliminary internal testing, we tackle two core issues faced frequently by dialogue designers grappling with the declarative paradigm:

- **Specification cannot be solved by the planner.** This is the case when the graph in Figure 2e does not appear at all, and the dialogue designer is left with an inscrutable "no solution found" message and nothing else to work with. Our goal here is to surface features from the current specification back to the designer so that they can fix identify the root cause of the unsolvability.
- **Solution does not match expectations.** Here, the problem is solvable but the solution does not match the designer's expectations – i.e. the graph in Figure 2e looks nothing like what they were aiming for. The goal here for us is to be able to respond to questions from the designer such as *Why is this a solution?* and *Why is this not a solution?*, so that they can modify the specification accordingly until they are satisfied with the outcome.

Some of these questions might look familiar with the line of investigation in (Smith 2012; Fox, Long, and Magazzeni 2017; Cashmore et al. 2019). However, the setting here involves the domain designer and not the end user. Thus the suite of challenges not only include the unsolvability question, not addressed in those works, but also the explanatory dialogue here is geared towards the model acquisition task rather than the exploration of the decision making process.

Contributions

- We formalize the XAIP problem for the model acquisition scenario and illustrate the salient challenges involved on a tool for the design of goal-directed conversational agents.
  - To this end, we motivate how the XAIP framework must also consider the unsolvability problem in addition to the explanation of generated plans, as has been mostly focused on in existing literature.
- We build on recent work in explaining unsolvability of classical planning problems (Sreedharan et al. 2019) and extend it to handle non-determinism as required by the particular tool under consideration.
- We demonstrate the usefulness of the approach via illustrations and empirical evaluations.

Note that, while unsolvability has been explored before, such as in the generation of excuses (Göbelbecker et al. 2010) (which focus on counterfactuals that are not necessarily useful for the domain acquisition task) or in the generation of certificates (Eriksson, Röger, and Helmert 2017) (which are mostly only machine consumable), here we are concerned primarily with assisting a domain writer who needs an easily understandable explanation of unsolvability rather than a certificate for verification or an excuse that does not provide any insight on how to fix their specification.

XAIP for Model Acquisition

We now formalize the notion of explainable planning in the context of model acquisition, particularly extended to non-deterministic planning used by the tool in our case study.

Planning Problems

A planning problem $\Pi : \mathcal{M} \mapsto \{\pi\}$ takes in as input a model $\mathcal{M} = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$ where $\mathcal{F}$ is a set of propositions that describe the state of the world and we will use $\mathcal{S}$ to represent the set of possible states that can be constructed from $\mathcal{F}$. $\mathcal{A}$ is a set of operators or actions available to an agent and produces a set of plans π that transform the initial state $\mathcal{I} \subseteq \mathcal{F}$ to the goal state $\mathcal{G} \subseteq \mathcal{F}$.

- For a deterministic model, each action produces a single next state: $\mathcal{A} \ni a : S \rightarrow S$. The solution to a deterministic problem are plans represented as a sequence of actions.
- The execution of a non-deterministic action $\mathcal{A} \ni a : S \rightarrow \{S\}$ can result in more than one possible state. The solution to a non-deterministic planning problem Π is thus a contingent plan which induces a set of plans. The behavior of the agent is described by one of those plans depending on which outcomes occur at the time of execution.

Since we are primarily concerned here with goal-directed agents (e.g. goal-directed conversations), we consider the space of behaviors represented by $\Pi(\mathcal{M}))$ as the solution space. The discussion generalizes to a "space of plans" enabled by a domain in the absence of a goal or initial state. Throughout the paper, we will refer to a non-deterministic problem as having a plan if a *weak solution* exists (Cimatti et al. 2003) – i.e., there is some sequence of actions and outcome selections that achieves the goal.

Transformations on $\mathcal{M}$

We now introduce a few operations in the space of models that we will deploy later to tackle the various challenges in model space reasoning for XAIP and model acquisition.

Model Edits Model edits $\delta : \mathcal{M} \mapsto \mathcal{M}'$ (Keren et al. 2017; Chakraborti et al. 2017) change one or more conditions in the model to generate a new model. For example, this could involve adding or removing a condition from the initial or goal state or from the set of preconditions and effects of an action. Search in the space of models propagates by the application of one or more such model edits. The size of a model edit is denoted by $|\delta|$.

Abstractions Model abstractions $Abs : \mathcal{M} \times \mathcal{P} \mapsto \mathcal{M}'$, on the other hand, simulate a collection model edits together that change one or more features of the model (Clarke et al. 2000). Here $\mathcal{P}$ is the set of variables that can be projected away. For example, authors in (Sreedharan, Srivastava, and Kambhampati 2018) use this technique of syntactic state variable projection to determine the right level of abstraction to present explanations in, while in (Sreedharan et al. 2019) authors use the same technique to explain unsolvability. We build on the latter in this paper, particularly extended in service of non-deterministic planning used in the tool under study. We will go into more details of this later.

Determinization Determinization is the process of turning a non-deterministic model $\mathcal{M}$ into a deterministic one $Det(\mathcal{M})$ – e.g. an "all-outcomes" determinization scheme (Yoon, Fern, and Givan 2007) transforms an action $a : S \rightarrow \{S_i\}$ into a set of actions $\forall i \; a_i : S \rightarrow S_i$ so that all the outcomes of an action with non-deterministic effects can be realized in the determinised model. With the fairness assumption (Cimatti et al. 2003), a solution to $Det(\mathcal{M})$ is also a valid behavior for $\mathcal{M}$, i.e. $\Pi(Det(\mathcal{M})) = \Pi(\mathcal{M})$.

Plan Preservation The model transformation $Obs : \mathcal{M} \times \{\pi\} \mapsto \mathcal{M}'$ receives a model and a (partial) sequence of actions (equivalent to a set of possible plans) and produces a compiled model where this sequence must be preserved, i.e. $\{\pi\} \subseteq \Pi(\mathcal{M}')$. In the past, such techniques have been used in the compilation of the goal recognition task into a classical planning problem (Ramírez and Geffner 2009) or in the construction of partial foils from the end user for the purposes of explanation (Sreedharan, Srivastava, and Kambhampati 2018). In the preservation technique used here for deterministic models, we allow for partial observation of non-determinism as well, in addition to the usual partial plans, in order to account for the specific needs of the tool under study. This means that an action may be specified in a plan but its outcome may be left unspecified.

The XMAS Problem

Explanations in the context of the model acquisition task provide a unique twist to a well known framework in XAIP – *model reconciliation* (Chakraborti et al. 2017). There, the task is to compute, given the agent model, the mental model, and a plan to explain, a set of updates that when applied to the mental model would render the given plan optimal (and hence without any foil) in the updated mental model.

Here also, we have two models – the one currently specified by the domain writer or *the revealed model*, and the one that they wanted to specify or *the mental model*. Furthermore, similar to the case of model reconciliation process, here too we have model differences – the revealed model and the mental model do not match due to mistakes made by the domain writer. However, unlike in the case of explanations in the model reconciliation framework, the target here is for the explainable AI system to transform the revealed model to the mental model, rather than updating the mental model to agree with the revealed model as focused on traditionally in the model reconciliation framework.

This is an iterative process with the human (in this case, the domain writer) firmly in the loop. This is because, unlike in standard closed world model estimation tasks where the features of the model are known, and a transition function between domain models can be specified (Bryce, Benton, and Boldt 2016; Keren et al. 2017), here the problem is open ended as the domain writer gradually builds their agent model. Thus, the model acquisition task is strictly *not one of estimation of the mental model*.

An Explainable Model Acquisition Setting XMAS is defined by the tuple $\Psi = \langle \mathcal{M}, \mathcal{M}^H \rangle$ where $\mathcal{M}$ is the revealed model and $\mathcal{M}^H$ is the mental model of the domain writer.

Since the eventual goal of writing a domain is to enable a desired set of behaviors, and there may be many ways to specify the same agent behavior, we condition the end goal of an XMAS in terms of the space of plans afforded by the agent model that is being specified. The solution to Ψ is a sequence of model updates $\Delta = \langle \delta_1, \delta_2, \dots \delta_n \rangle$ such that:

$$\Pi(\sum_i \delta_i(\mathcal{M})) = \Pi(\mathcal{M}^H)$$

This means that at the end of the model acquisition process the domain writer has successfully captured the space of behaviors of the agent they were trying to model. In contrast to the model reconciliation process, for the model acquisition task, these updates are, of course, generated by the domain writer. From the XAIP perspective, the task of the planner here is to empower the domain writer to come up with the most efficient Δ – e.g. $\min \sum_i |\delta_i|$ to reduce the overall complexity of the model acquisition process or just $\min |\Delta|$ to reduce the number of steps to reach the final model. The evaluation of the entire process requires research on the UX of abstractions, and is out of scope of this paper.

Instead, in the following discussion, we tackle key difficulties faced by domain writers for individual interactions during this process (as experienced in preliminary internal tests of D3WA). The focus of the proposed solutions here is to address the computational limitations of the domain writer using XAIP techniques like domain abstractions that have been shown to be useful in user studies (Sreedharan et al. 2019) as a vehicle for explanations in complex domains.

Q1. Why is there no solution?

This is the case when: $\Pi(\mathcal{M}^H) \neq \Pi(\mathcal{M}) = \emptyset$, i.e. the domain writer mistakenly thinks that they have a solvable model. As we mentioned before, for a model acquisition task, it is not enough to surface a cause for unsolvability but one must also make sure that the domain writer gets actionable information in order to remedy the situation. The directive from the planner thus has the following components:

1. **Minimal Unsolvability** We present to the domain writer the smallest possible abstraction $\mathit{Abs}(\mathit{Det}(\mathcal{M}), \mathcal{P})$ such that this model too does not have any solution.

$$\begin{aligned} \text{Find: } & \mathcal{P} \\ \text{s.t. } & \Pi(\mathit{Abs}(\mathit{Det}(\mathcal{M}), \mathcal{P} \cup G)) = \emptyset \\ \text{and } & \min |\mathcal{P}| \\ \text{Show: } & \mathit{Abs}(\mathit{Det}(\mathcal{M}), \mathcal{P} \cup G) \end{aligned}$$

   By the properties of the abstraction used, $\Pi(\mathcal{M}) \neq \emptyset$ only if $\Pi(\mathit{Abs}(\mathit{Det}(\mathcal{M}))) \neq \emptyset$. Thus the domain writer can fix the root cause of unsolvability in this simpler domain first. We will later show in the empirical evaluations how this approach can significantly reduce the size of the specification that the domain writer has to inspect in order to fix an unsolvable model. We include the goal in all abstractions – this fluent is not directly accessible to the designer.[1]

2. **Maximal Solvability & Exemplary Plan Failure** While the previous component of the explanation provides a simpler version of the model for the user to debug, it does not illustrate failures of any potential solutions to motivate fixes that the designer might attempt. The following complements it with an illustrative failure in the current model of a plan generated from the *maximally solvable* abstraction, while letting the domain writer continue fixing the issue on the minimally unsolvable one.[2]

$$\begin{aligned} \text{Find: } & \mathcal{P} \\ \text{s.t. } & \Pi(\mathit{Abs}(\mathit{Det}(\mathcal{M}), \mathcal{P} \cup G)) = \{\pi\} \neq \emptyset \\ \text{and } & \max |\mathcal{P}| \\ \text{Show: } & \Pi(\mathit{Obs}(\mathcal{M}, \{\pi\})) = \emptyset. \end{aligned}$$

   The point here is to find the most complete simplification of the model where a solution exists and illustrate – for example, using VAL (Howey, Long, and Fox 2004) – to the user why that solution does not apply to $\mathcal{M}$. The domain writer could also use this sample plan to explore where exactly a possible solution becomes invalid in the current specification, or even use it as a starting point to create more complex foils to investigate further.[3]

3. **Subgoals and Landmarks** In addition to presenting the abstractions and an exemplary plan failure, we can further help the domain writer debug the problem by presenting them with a subgoal that is necessary for achieving the goal in the current specification but can not be achieved. This is meant to serve as a prompt for the domain writer for a possible issue to focus on. We follow (Sreedharan et al. 2019) and use landmarks (Hoffmann, Porteous, and Sebastia 2004) for identifying such subgoals.

[1]This does not mean that the designer cannot specify a goal. They can specify any starting condition and point out which of the outcomes in one or more actions ends the conversation. The latter is then compiled internally to a single goal achieving condition. Thus the framework is quite generic for modeling any goal-directed process and not necessarily tied to any specific initial condition or goal state. For more details, please refer to (Muise et al. 2019a).

[2]Note that we cannot generate a plan from a minimal unsolvable abstractions since there are no solutions there. One possibility would be to choose a model more abstract than the minimally unsolvable one. Unfortunately, these model would ignore most model features and provide no useful plan for the user to debug.

[3]To improve the efficiency of the search, we will restrict the search for abstraction set over subsets of $(F \setminus P_{min}) \cup \mathcal{G}$ (where P_{min} is minimum abstracting set). Since the abstraction techniques followed here guarantees that abstractions formed from supersets of $P_{min} \cup \mathcal{G}$ will not result in solvable model. Moreover if P_{min} is the only subset of fluents leading to the unsolvability then $(F \setminus P_{min}) \cup \mathcal{G}$ should automatically be a solvable domain. So we will start our search from $(F \setminus P_{min}) \cup \mathcal{G}$ and will look at systematically relaxing the model until we get a solvable one. We evaluate this approximation against the exact approach in our evaluations.

The earlier work constructs landmarks from models that are more abstract than the minimally unsolvable model but this can lead to less informed subgoals as it may not be considering many of the state factors. We will instead extract landmarks directly from the the minimally unsolvable model using the following proposition:

Proposition 1 *If $\mathcal{M}$ is unsolvable, and $Abs(\mathcal{M}, \mathcal{P})$ is solvable while $Abs(\mathcal{M}, \mathcal{P} \cup \{f\})$ is not, then either the delete relaxation of $Abs(\mathcal{M}, \mathcal{P} \cup \{f\})$ is solvable or there must be an unmet landmark corresponding to f.*

This follows from the fact that in the abstraction scheme we follow, when we consider the delete relaxation of $Abs(\mathcal{M}, \mathcal{P} \cup \{f\})$ over that of $Abs(\mathcal{M}, \mathcal{P})$, the only possible differences can be that f may be part of preconditions and adds for an action. If this addition makes the delete relaxation unsolvable then that means that all relaxed plans for $Abs(\mathcal{M}, \mathcal{P})$ contained actions that required f as a precondition. The problem becomes a bit more involved when we allow for negative precondition as we need to test whether the possible landmark is f or $\neg f$.

Given this proposition, we first test if delete relaxation of $Abs(\mathcal{M}, \mathcal{P})$ is solvable so we can extract landmarks from it similar to (Sreedharan et al. 2019). Otherwise we test if addition of f in initial state makes the delete relaxation of $Abs(\mathcal{M}, \mathcal{P} \cup \{f\})$ solvable. If it does, then the landmark is f, else it is $\neg f$. While this can lead to more informed landmarks, for cases where the delete relaxation is unsolvable, we would not be able to leverage the ordering of the landmarks to find subgoals appearing earlier in the sequence.

Q2. Why is this not a solution?

This is the case when (for a set of plans $\{\pi\}$ presented by the domain writer): $\{\pi\} \subseteq \Pi(\mathcal{M}^H)$ but $\{\pi\} \not\subseteq \Pi(\mathcal{M})$. This is really a special case of **Q1** – we perform the following transformation to solve this:

Set: $\mathcal{M} \leftarrow Obs(\mathcal{M}, \{\pi\})$

If: $\Pi(Obs(\mathcal{M})) \neq \emptyset$ (the compilation is solvable) then demonstrate to the user that $\nexists \pi \in \Pi(Obs(\mathcal{M}, \{\pi\}))$ such that $cost(\pi) > cost(\pi_i) \; \forall \pi_i \in \{\pi\}$ – this means that the foil is not better than anything else already in the solution. At this point, the user can ask **Q3**.

Else: Follow **Q1** with $\mathcal{M}$.

Note that we cannot run VAL directly on the foil since: 1) It is very unlikely that the domain writer will provide full foils – this is largely due to the effort required in doing so but also uniquely infeasible for the current setting of dialogue design since internal system actions such as web calls and logical inferences are not part of logs that are used to stress test the design of the agent; and 2) in the case of a partial foil, $Obs(\mathcal{M})$ may not have a solution to run VAL with.

Q3. Why is this a solution?

This is the case when (for a set of plans $\{\pi\}$ presented by the domain writer): $\{\pi\} \not\subseteq \Pi(\mathcal{M}^H)$ but $\{\pi\} \subseteq \Pi(\mathcal{M})$. Here, the domain designer is surprised that a solution they did not expect is part of $\Pi(\mathcal{M})$. The provenance of actions along such a solution of $\mathcal{M}$ can be communicated to the designer through the visualization of the necessary causal links (Chakraborti et al. 2019a; Bercher, Behnke, and Biundo 2015). *We do not repeat this line of inquiry here.*

Non-deterministic XMAS

In this section, we go into specific details of the abstraction and landmark formulation for non-deterministic models that we had to develop specifically for the tool under study.

Each action a in the non-deterministic model is defined as: $a = \langle \text{prec}^a, \mathcal{O}^a \rangle$ where $\text{prec}^a \subseteq F$ is the set of preconditions that needs to be true for the action a to be executable and $\mathcal{O}^a = \{o_1^a, \ldots, o_k^a\}$ is the set of k possible outcomes for action a. Each outcome is further defined as the tuple $o_i^a = \langle \text{adds}_{o_i^a}, \text{dels}_{o_i^a} \rangle$, where $\text{adds}_{o_i^a}$ and $\text{dels}_{o_i^a}$ are the adds and deletes corresponding to the outcome. We now show that *syntactic projection in fact results in logically complete abstractions, i.e. they allow for more behavior and not less.*

Definition 1 *A model abstraction $\mathcal{M}' = Abs(\mathcal{M}, \mathcal{P})$ is logically complete if there exists a surjective mapping from states in $\mathcal{M}$ to $\mathcal{M}'$ and $\Pi(\mathcal{M}) \subseteq \Pi(\mathcal{M}')$.*

Now let us consider the syntactic projection function $Abs_p : \mathcal{M} \times 2^F \mapsto \mathcal{M}'$.

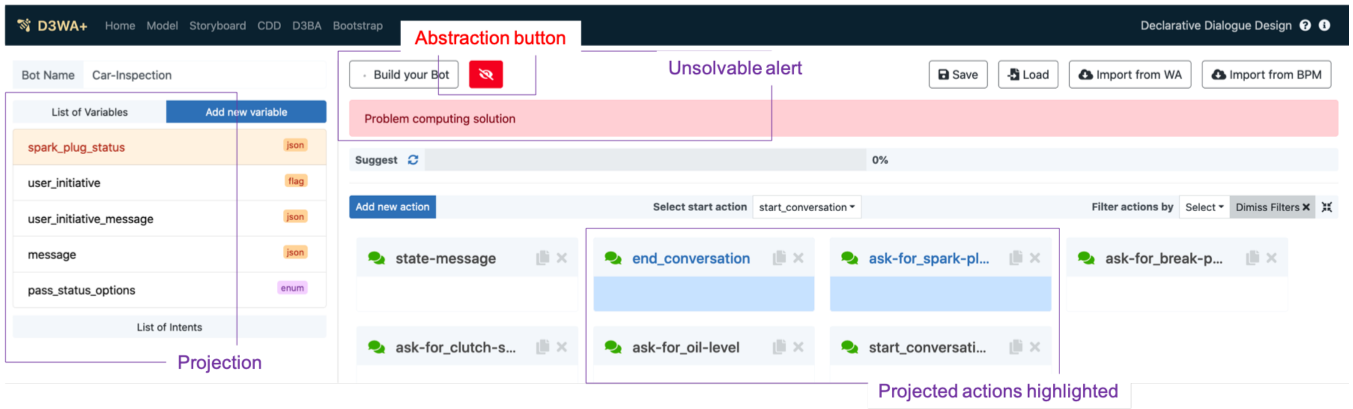
Definition 2 *Given a non-deterministc model $\mathcal{M}$ and a set of fluents $\mathcal{P}$: $Abs_p(\mathcal{M}, \mathcal{P}) = \langle F \setminus \mathcal{P}, \hat{A}, \mathcal{I} \setminus \mathcal{P}, \mathcal{G} \setminus \mathcal{P} \rangle$ where $\forall a \in A, \exists \hat{a} \in \hat{A}$ such that: $prec^{\hat{a}} = prec^a \setminus \mathcal{P}$ and every outcome becomes $\mathcal{O}_i^{\hat{a}} = \langle adds_{\mathcal{O}_i^{\hat{a}}} \setminus \mathcal{P}, dels_{\mathcal{O}_i^{\hat{a}}} \setminus \mathcal{P} \rangle$.*

This is a logically complete abstraction. Consider a transition $\langle S_i, a_i, S_{i+1} \rangle$ that is valid for $\mathcal{M}$, we can see that, $S_i \subseteq \text{prec}_{a_i}$ and there must exist an outcome $o_k^{a_i}$ for a_i such that $S_{i+1} = (S_i \setminus dels_{\mathcal{O}_k^{\hat{a}_i}}) \cup adds_{\mathcal{O}_i^{\hat{a}}}$. Then there must exist a corresponding transition $\langle S_i \setminus \mathcal{P}, \hat{a}_i, S_{i+1} \setminus \mathcal{P} \rangle$ that is valid for $Abs_p(\mathcal{M}, \mathcal{P})$ with an outcome $o_k^{\hat{a}_i}$.

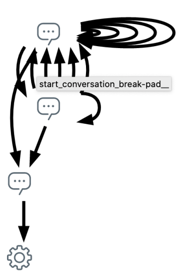
With these tools in place, we can generate explanations described in earlier sections by doing a search over the space of non-deterministic models. Though it would be easier if we can just focus on settings where we are testing solvability of classical planning model and extracting subgoals from these simpler models. We can in fact to do this, by showing that all of our explanation generation procedures can be performed on the determinised version of the original non-deterministic model. Specifically, we will show that the abstraction of a determinised model is the determinization of the abstracted non-deterministic model:

Proposition 2 *Given a non-deterministic model $\mathcal{M}$ and a set of fluents $\mathcal{P}$: $Det(Abs_p(\mathcal{M}, \mathcal{P})) = Abs_p(Det(\mathcal{M}), \mathcal{P})$.*

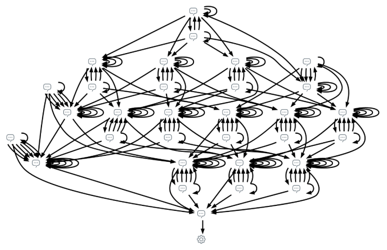
The determinised model will contain an action for each possible outcome, i.e. $\forall o_i^a, Det(\mathcal{M})$ contains an action $a_{o_i} = \langle \text{prec}^a, adds_{o_i^a}, dels_{o_i^a} \rangle$. So the projection of this determinised action will be $\langle \text{prec}^a \setminus \mathcal{P}, adds_{o_i^a} \setminus \mathcal{P}, dels_{o_i^a} \setminus \mathcal{P} \rangle$, and you would get the same action if you were determinising based on projected o_i^a.



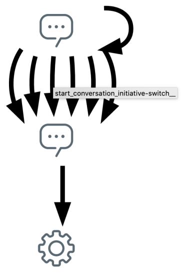
(a) Snapshot of D3WA+ illustrating model abstraction in response to unsolvability. Since the model has no solution, there is no generated dialogue plan and the canvas for visualization is empty – our user story starts with a white XMAS. The minimal unsolvable abstraction is presented to the designer so that they can devise a fix in the simpler model first, before translating that to the full specification. Note how some variables in the domain have disappeared in the minimal abstraction, in comparison to Figure 1. The designer can toggle back and forth between the full and projected models using the red button, until they are able to devise a fix. The highlight feature communicates to the user which of the projected fluents (orange hover) are associated with which of the actions (blue highlight).



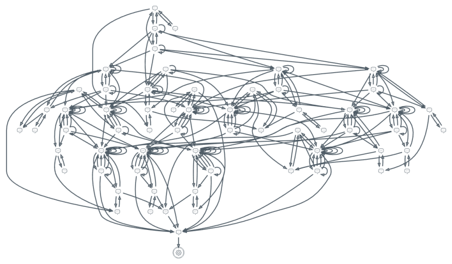
(b) XMAS Past: Solution of the minimal abstraction. Now that the designer was able to fix the simplified domain, they can try applying the fix to the full specification.



(c) XMAS Present: This is the solution when the fix to the minimal abstraction is applied to the full specification. Now it is at least solvable but something doesn't look quite right: the solution does not match the expectation of the domain designer. Now that they have a solution to work with, they now direct questions of the form of **Q2** to D3WA+. Figure 3 illustrates one such interaction.



(d) XMAS Present: The domain writer solves the minimal abstraction again. Applying this fix to the original specification takes the domain writer from Figure 2c to 2e.



(e) XMAS Future: This is the final (target) solution: a model that is equivalent to $\mathcal{M}^H$. The domain writer, of course, does not have access to this until they get there – and neither does D3WA+– but instead they continue working on their current specification and its solution (or the lack thereof) until they are satisfied. During the course of XMAS, the system was able to guide the designer along this journey by exposing the mistakes in their past models.

Figure 2: Examples of generated dialogue graphs illustrating the reconciliation of $\mathcal{M}^H$ to $\mathcal{M}$ during XMAS. The nodes in the graph stand for agent actions while the edges are the non-deterministic outcomes. The graph is meant to give the reader a sense of the sizes of the abstract solutions. Though the actual labels on the nodes and edges are not visible here, the system does allow the domain designer to drill down further as required.

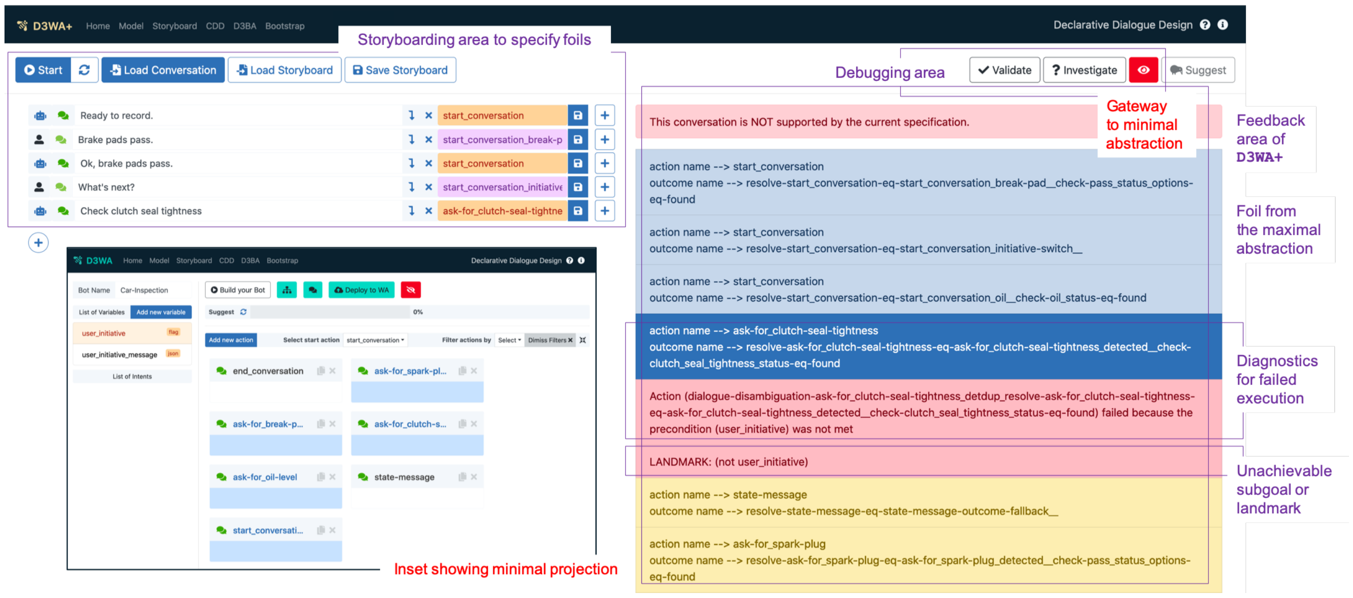
Illustrations on D3WA+

We will now illustrate each of the use cases covered above on our tool D3WA+. In the context of the design of dialogue agents using planning, each "solution" is a potential conversation path allowed by the agent design. Hence, the model acquisition process is one of the dialogue designer ensuring which conversations are allowed and which are not.

Demonstration While we attempt to illustrate as much of the use cases as possible in the limited space available here, please refer to ibm.biz/d3wa-xaip for more detailed walk-throughs. (Duration: 7min 55sec *excluding* explanation generation time reported in Table 1)

Car Inspection Bot For purposes of illustration, we consider the design of a conversational agent tasked with helping in the inspection of a car. This domain is adapted from (Muise et al. 2019a) as a typical demonstration of the design of conversational agents using automated planning techniques. The final dialogue plan, as seen in Figure 2e, has 63 nodes and 272 edges and is thus quite comfortably out of scope for the state of the art in dialogue design. The declarative specification as seen in Figure 1, on the other hand, has just 8 variables and 7 non-deterministic actions. Let us consider the following dialogue in this domain:

```
Bot: Ready to record.
```



Ready to record. — start_conversation
Brake pads pass. — start_conversation_break-p
Ok, brake pads pass. — start_conversation
What's next? — start_conversation_initiative
Check clutch seal tightness — ask-for_clutch-seal-tightne

This conversation is NOT supported by the current specification.

action name --> start_conversation
outcome name --> resolve-start_conversation-eq-start_conversation_break-pad__check-pass_status_options-eq-found

action name --> start_conversation
outcome name --> resolve-start_conversation-eq-start_conversation_initiative-switch__

action name --> start_conversation
outcome name --> resolve-start_conversation-eq-start_conversation_oil__check-oil_status-eq-found

action name --> ask-for_clutch-seal-tightness
outcome name --> resolve-ask-for_clutch-seal-tightness-eq-ask-for_clutch-seal-tightness_detected__check-clutch_seal_tightness_status-eq-found

Action (dialogue-disambiguation-ask-for_clutch-seal-tightness_detdup_resolve-ask-for_clutch-seal-tightness-eq-ask-for_clutch-seal-tightness_detected__check-clutch_seal_tightness_status-eq-found) failed because the precondition (user_initiative) was not met

LANDMARK: (not user_initiative)

action name --> state-message
outcome name --> resolve-state-message-eq-state-message-outcome-fallback__

action name --> ask-for_spark-plug
outcome name --> resolve-ask-for_spark-plug-eq-ask-for_spark-plug_detected__check-pass_status_options-eq-found

Figure 3: Snapshot of D3WA+ illustrating foil-based interactions when a model is solvable but does not match expectations. Note how small this projection is (inset), even smaller compared to the first unsolvable model in Figure 2a: the corresponding solution in Figure 2d is similarly smaller than the one in Figure 2b. This showcases an interesting property of XMAS– the size of abstractions is non-monotonic in the course of the model acquisition task. On the right, we can see here the foil generated by D3WA+ from the maximal abstraction with diagnostic information on how this fails in the current specification.

```
User: Break pads pass.
Bot: Ok, break pads pass.
User: What's next? <-- initiative switch!
Bot: Check the spark plugs.
User: What are the options.
...
Bot: Inspection complete!
```

The interesting part is the potential for *initiative switch* during the conversation – either the user can go through all the parts by themselves or hand over control to the bot to guide them, or a combination of both.[4] The salient feature of the specification is thus: there is a catch-all dialogue action to respond to the user when they have initiative and a set of actions to ask the user for information when the bot has initiative. There is one outcome in all these actions that switch initiative based on what the user has said, while the other outcomes update the state of the inspection by logging the correct variables based on the user utterance. Next, we follow the designer's journey to get to this specification.

***Q1.* Why is there no solution?** In our user story, the designer has forgotten to add a few critical domain conditions – the OUTCOME for the initiative switch is missing in the catch-all action while the UPDATE for spark plugs is also missing in the corresponding OUTCOME (refer back to the introduction to D3WA for a refresher of this modeling artifacts). As a result the model has become unsolvable – there is no way for either the inspector or the bot to drive the initiative and visit all the parts. In Figure 2a the user is presented with the minimal abstraction where the model is unsolvable.

[4]The specification is available at: ibm.biz/d3wa-inspection.

They fix this simpler specification to arrive at the solution in Figure 2b. The fix – adding the UPDATE for spark plugs – when applied to the original specification takes the designer to the current dialogue plan in Figure 2c. This interaction with D3WA+ allowed the user to find a fix for an unsolvable model by inspecting a *much* simpler model.

***Q2.* Why is this not a solution?** However, the missing OUTCOME for the initiative switch in the catch-all action is still missing.[5] As a result, all solutions right now only involve the user driving the conversation and the resulting solution in Figure 2c looks different from what the designer was expecting – i.e. it does not contain any conversation flow where the bot has initiative. The domain writer expects the sample conversation to be possible but do they know it's unsolvable, at all? In the spirit of XMAS, the domain writer gets to query D3WA+ with this foil – see Figure 3. The system again responds with a minimal abstraction to fix, along with a sample plan and an unachieved landmark in the maximally solvable abstraction illustrating why that foil fails in the current model. This not only explains the unsolvability but also provides powerful directive to fix the model.

Empirical Evaluations

We empirically investigate the properties of XMAS in terms of the size of the abstractions relative to the size of the original specifications, and the time taken to generate them. The

[5]We want to impress on the reader at this point that this discussion of "mistakes" or missing components are in hindsight – the target model does not exist until the domain designer gets there.

problem instance		maximal abstraction (approx)						maximal abstraction (exact)						minimal abstraction						
		size				time		size				time		size				time		
		$\lvert\mathcal{P}\rvert$	abstraction	size in %	plan	abstraction	plan	$\lvert\mathcal{P}\rvert$	abstraction	size in %	plan	abstraction	plan	$\lvert\mathcal{P}\rvert$	abstraction	size in %	plan	abstraction	plan	landmark
	p_1	21	286	86.405	5	0.882	0.956	22	307	92.749	5	5.711	0.987	2	47	14.199	1	111.363	1.014	1.103
Car	p_2	21	293	88.52	5	0.911	1.011	22	314	94.864	5	10.816	1.057	2	40	12.085	1	175.995	0.826	1.158
Inspection	p_3	21	286	86.405	5	1.069	0.967	22	307	92.749	5	13.561	1.117	2	47	14.199	1	208.862	0.863	1.088
	p_4	21	282	85.196	5	0.926	0.981	22	303	91.541	5	16.290	0.987	2	51	15.408	2	154.084	0.891	1.140
	p_5	21	286	86.405	5	0.995	0.978	22	307	92.749	5	16.351	1.058	2	47	14.199	1	237.173	0.984	1.041
	p_1	25	338	94.15	1	0.999	0.993	25	338	94.15	1	11.606	17.452	1	23	6.407	1	9.216	0.936	1.056
Data	p_2	25	338	94.15	1	1.288	1.162	25	338	94.15	1	14.279	1.157	1	23	6.407	1	13.302	0.971	1.196
Doppelganger	p_3	25	338	94.15	1	2.405	2.914	25	338	94.15	1	25.561	1.050	1	23	6.407	1	23.444	0.973	1.211
	p_4	25	338	94.15	1	0.834	0.924	25	338	94.15	1	11.035	0.935	1	23	6.407	1	11.115	0.923	1.000
	p_5	25	338	94.15	1	2.661	2.845	25	338	94.15	1	13.157	4.792	1	23	6.407	1	12.895	0.792	1.136
	p_1	75	2104	97.227	8	1.139	1.209	76	2161	99.861	8	31.396	1.202	2	61	2.819	2	1106.939	0.950	1.137
Credit	p_2	76	2161	99.861	7	1.113	1.070	76	2161	99.861	7	46.256	1.051	1	5	0.231	1	40.325	0.934	1.093
Card	p_3	76	2153	99.492	4	1.155	1.069	76	2153	99.492	4	33.903	1.105	1	12	0.555	1	29.498	0.935	1.165
	p_4	76	2160	99.815	7	1.040	1.149	76	2160	99.815	7	54.096	1.159	1	3	0.139	1	47.422	0.878	1.016

Table 1: Empirical properties of `XMAS` in three typical conversational domains modeled in `D3WA+`. Notice the massive reduction in size for the minimal abstraction, intended to make it easier for the domain writer to inspect issues with the domain. Also notice the large difference in size between the minimal and maximal abstractions, thereby indicating the need for a maximal abstraction to capture enough model information in order to produce a useful foil for the domain writer to fix.

size of the abstraction allows us to measure how much simplification of the model is achieved by our method, while the time taken is a measure of viability of our approach. We focus on **Q1** here since the properties of the solutions to **Q2** are derived from **Q1** while, as we mentioned before, **Q3** is already quite well understood in existing literature.

Test Domains To test out the empirical properties of our approach, we use two new domains, in addition to the car inspection domain used in the illustrative examples. The first of them – Data Doppelganger – is an assistant chat-bot that that helps a user perform variety of data science tasks, such as plotting a graph, given a data set. The other new domain – Credit Card Recommendation – is again adopted from (Muise et al. 2019a), and takes the user through choices of credit cards and their features until they make a selection.

To evaluate the effectiveness of `XMAS`, we needed to evaluate the systems on plausible mistakes on these test domain. For Table 1, we tried to create unsolvable problems for the first three domains by removing three model conditions at random. Specifically, we deleted adds and initial states from the model. Unfortunately, for the credit card domain randomly removing a subset of model components weren't yielding unsolvable problems. So instead, we went with a unique domain-specific mistake for each of the five cases. Each mistake was centered around the domain author missing adds for a specific outcome or initial state for a specific proposition. After identifying a reason for unsolvability, we further delete two additional adds per unsolvable instance. The solvability of the determinized problems was tested using FastDownward (Helmert 2006) implementation of A^* and LM-Cut (Helmert and Domshlak 2009). The landmarks were extracted using FastDownward implementation of (Keyder, Richter, and Helmert 2010) with $m = 1$. All experiments had a timeout of 60 mins.[6]

Model Compression Table 1 shows the amount of compression offered by the abstraction, against the size of the full models, for five randomly generated unsolvable instances. Here the size of each possible model is reported in

[6]p_5 for credit card domain timed out (removed from Table 1).

# edits	1			2			3		
problem	$\lvert\mathcal{P}\rvert$	size	time	$\lvert\mathcal{P}\rvert$	size	time	$\lvert\mathcal{P}\rvert$	size	time
p_1	2	47	49.477	2	51	87.685	✓	✓	✓
p_2	2	40	152.989	2	51	193.358	-	-	-
p_3	2	40	184.747	✓	✓	✓	✓	✓	✓

Table 2: Size of abstractions along the entire model reconciliation process during the model acquisition task.

terms of the number of non-goal fluents that are part of it ($|\mathcal{P}|$), the number of model conditions that are part of the problem (denoted as size in the table) and the percentage of model conditions remaining as compared to the original domain model (denoted as 'size in %' in the table). The larger the compression, the easier we make it for the domain writer to understand the cause of unsolvability, as has been established in existing literature (Sreedharan et al. 2019). Clearly, we are able to significantly reduce the size of the specification for this purpose using the proposed abstraction approach. For the credit card domain, where the original model contains close to 2k components, being able to focus on a subset containing only 5 conditions is a massive reduction.

Maximal versus Minimal Abstractions Table 1 also shows the difference in size of the plans in the minimal[7] and maximal abstractions. As we mentioned before, this was a specific design choice made so as to ensure that the domain writer has a reference point while inspecting an unsolvable model – they can use this either to debug the current model or to explore newer foils. The point of computing this reference point in the maximal as opposed to the minimal model (as evident from Table 1) is to provide more helpful debugging information to the domain writer – `XMAS` is not just about explaining unsolvability but also completing the model acquisition task. The reference point uses the maximal model in order to make sure maximal number of model features are considered so that the generated foil is as close

[7]Note that since the minimal model is also unsolvable, the plans used here for comparison are from the models that are one abstraction simpler than the minimal unsolvable model.

as possible to a plan the user might be looking for in $\mathcal{M}^H$. In Table 1, we refer to generating maximal model by starting from abstraction corresponding to $F \setminus P_{min}$ while the exact one refers to a systematic search starting from the most concrete domain to one where it is solvable. While search for maximal model was in general fast, we can see that the approximate method is much faster and finds models that are quite comparable to the exact minimal solution.[8]

Model Evolution In Table 2, we illustrate the evolution of the size of the abstractions as the domain writer progressively fixes the car inspection model. To simulate this, we needed to create models with potentially multiple causes for unsolvability. We generated nine pairs of model conditions (adds or initial state), each of whose removal can lead to an unsolvable problem. We then created three problems by removing three of those pairs. This means the model could have from one to potentially six unique causes for unsolvability. We passed each of these problems through our system, looked at the abstractions, made fixes and then tested if there were further issues. For p_1 and p_3, we were able to successfully make the problem solvable. While for p_2, we were able to fix two issues, but the explanation system timed out on the third trial. This illustrates the journey of the domain writer towards converging the space of solutions in $\mathcal{M}$ and $\mathcal{M}^H$ and the massive simplification of the model complexity offered by the proposed approach along this journey.

Computation Finally, we report the time taken to compute the different components of the planner's directive to the domain writer, in Table 1. We do not explore optimizations in this paper, but we are well within the bounds of real-time use even for the massive credit card domain.

Conclusion

In this paper, we formulated the explainable planning challenges in model acquisition tasks and demonstrated them in the context of a tool for the design of goal-oriented conversational agents using automated planning. Perhaps one of the most compelling aspects of our work is how powerful planning techniques and concepts themselves can be in helping with the acquisition of planning specifications to begin with.

Going forward, we intend to evaluate the approach with a wide base of seasoned dialogue designers. This is a significant undertaking since one must design the interface carefully in order to separate confounds that can mix up whether the designers could grasp the declarative mental model or not with indicators of whether the explanations themselves proved to be useful during debugging. While we laid the theoretical foundations of `XMAS` here, UX design is out of scope for this work. We hope to report on the results of investigation in that direction in the future.

Acknowledgments Kambhampati's research is supported in part by ONR grants N00014-16-1-2892, N00014-18-1-2442, N00014-18-1-2840, N00014- 19-1-2119, AFOSR grant FA9550-18-1-0067, DARPA SAIL-ON grant W911NF-19-2-0006, NSF grants 1936997 (C-ACCEL), 1844325, and a NASA grant NNX17AD06G.

[8]An interesting course of investigation in the future would be adopting the considerable body of work in the determination of dead ends during planning (Steinmetz and Hoffmann 2017; Muise 2014; Kolobov, Mausam, and Weld 2010) for `XMAS`. While our motivation here is user facing and is thus quite different to those works – i.e. we want to use abstractions to facilitate explanations, particularly in the model acquisition task, rather than speed up planning – it would be interesting to explore whether those techniques can speed up the explanation generation step in the future.

References

Amazon. 2019. Amazon Unveils Novel Alexa Dialog Modeling for Natural, Cross-Skill Conversations. Alexa Science Team Blog.

Bercher, P.; Behnke, G.; and Biundo, S. 2015. User-Centered Planning: A Discussion on Planning in the Presence of Human Users. In *International Symposium on Companion Technology (ISCT)*.

Botea, A.; Muise, C.; Agarwal, S.; Alkan, O.; Bajgar, O.; Daly, E.; Kishimoto, A.; Lastras, L.; Marinescu, R.; Ondrej, J.; Pedemonte, P.; and Vodolan, M. 2019. Generating Dialogue Agents via Automated Planning. *arXiv:1902.00771*.

Bryce, D.; Benton, J.; and Boldt, M. W. 2016. Maintaining Evolving Domain Models. In *IJCAI*.

Cashmore, M.; Collins, A.; Krarup, B.; Krivic, S.; Magazzeni, D.; and Smith, D. 2019. Towards Explainable AI Planning as a Service. *ICAPS Workshop on Explainable AI Planning (XAIP)*.

Chakraborti, T., and Khazaeni, Y. 2020. D3BA: A Tool for Optimizing Business Processes Using Non-Deterministic Planning. In *AAAI Workshop on Intelligent Process Automation (IPA-20)*.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *IJCAI*.

Chakraborti, T.; Fadnis, K. P.; Talamadupula, K.; Dholakia, M.; Srivastava, B.; Kephart, J. O.; and Bellamy, R. K. 2019a. Planning and Visualization for a Smart Meeting Room Assistant – A Case Study in the Cognitive Environments Laboratory at IBM T.J. Watson Research Center, Yorktown. *AI Communications*.

Chakraborti, T.; Muise, C.; Agarwal, S.; and Lastras, L. 2019b. D3WA: An Intelligent Model Acquisition Interface for Interactive Specification of Dialog Agents. In *AAAI & ICAPS System Demonstration Tracks*.

Chakraborti, T.; Sreedharan, S.; and Kambhampati, S. 2020. The Emerging Landscape of Explainable AI Planning and Decision Making. *arXiv:2002.11697*.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *Artificial Intelligence (AIJ)*.

Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; and Veith, H. 2000. Counterexample-guided Abstraction Refinement. In *International Conference on Computer Aided Verification*.

Computer Generated Solutions. 2018. Chatbots Deliver Speed, But Consumers Still Want Humans. Are We Moving too Quickly to Automation? CGS 2018 Consumer Customer Service Survey.

Eriksson, S.; Röger, G.; and Helmert, M. 2017. Unsolvability Certificates for Classical Planning. In *ICAPS*.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable Planning. *IJCAI Workshop on Explainable AI (XAI)*.

Göbelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming up With Good Excuses: What to do When no Plan Can be Found. In *ICAPS*.

Google. 2019. Introducing the Schema-Guided Dialogue Dataset for Conversational Assistants. Google AI Blog.

Helmert, M., and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *ICAPS*.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research (JAIR)*.

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered Landmarks in Planning. *Journal of Artificial Intelligence Research (JAIR)*.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning Using PDDL. In *ICTAI*.

Keren, S.; Pineda, L.; Gal, A.; Karpas, E.; and Zilberstein, S. 2017. Equi-reward Utility Maximizing Design in Stochastic Environments. In *IJCAI*.

Keyder, E.; Richter, S.; and Helmert, M. 2010. Sound and Complete Landmarks for AND/OR Graphs. In *ECAI*.

Kolobov, A.; Mausam; and Weld, D. S. 2010. SixthSense: Fast and Reliable Recognition of Dead Ends in MDPs. In *AAAI*.

Metz, R. 2018. Microsoft's Neo-Nazi Sexbot was a Great Lesson for Makers of AI Assistants. MIT Tech. Review.

Muise, C.; Chakraborti, T.; Agarwal, S.; Bajgar, O.; Chaudhary, A.; Lastras-Montano, L. A.; Ondrej, J.; Vodolan, M.; and Wiecha, C. 2019a. Planning for Goal-Oriented Dialogue Systems. *arXiv:1910.08137*.

Muise, C.; Vodolan, M.; Agarwal, S.; Bajgar, O.; and Lastras, L. 2019b. Executing Contingent Plans: Challenges in Deploying Artificial Agents. In *AAAI Fall Symposium*.

Muise, C.; McIlraith, S.; and Beck, C. 2012. Improved Non-Deterministic Planning by Exploiting State Relevance. In *ICAPS*.

Muise, C. 2014. *Exploiting Relevance to Improve Robustness and Flexibility in Plan Generation and Execution*. Doctoral Dissertation, University of Toronto.

Ramírez, M., and Geffner, H. 2009. Plan Recognition as Planning. In *IJCAI*.

Smith, D. E. 2012. Planning as an Iterative Process. In *AAAI*. Challenges in Planning, Spotlight Track.

Sreedhar, K. 2018. What it Takes to Build Enterprise-Class Chatbots. Chatbots.

Sreedharan, S.; Srivastava, S.; Smith, D.; and Kambhampati, S. 2019. Why Can't You Do That HAL? Explaining Unsolvability of Planning Tasks. In *IJCAI*.

Sreedharan, S.; Srivastava, S.; and Kambhampati, S. 2018. Hierarchical Expertise Level Modeling for User Specific Contrastive Explanations. In *IJCAI*.

Steinmetz, M., and Hoffmann, J. 2017. Search and Learn: On Dead-End Detectors, the Traps they Set, and Trap Learning. In *IJCAI*.

Yoon, S. W.; Fern, A.; and Givan, R. 2007. FF-Replan: A Baseline for Probabilistic Planning. In *ICAPS*.