Autonomous Agents and Multi-Agent Systems (2022) 36:29
https://doi.org/10.1007/s10458-022-09556-8

Explaining BDI agent behaviour through dialogue

Louise A. Dennis[1] · **Nir Oren**[2]

Accepted: 4 April 2022

Abstract
BDI agents act in response to external inputs and their internal plan library. Understanding the root cause of BDI agent action is often difficult, and in this paper we present a dialogue based approach for explaining the behaviour of a BDI agent. We consider two dialogue participants who may have different views regarding the beliefs, plans and external events which drove agent action (encoded via traces). These participants make utterances which incrementally reveal their traces to each other, allowing them to identify divergences in the traces, or to conclude that their traces agree. In practice, we envision a human taking on the role of a dialogue participant, with the BDI agent itself acting as the other participant. The dialogue then facilitates explanation, understanding and debugging of BDI agent behaviour. After presenting our formalism and its properties, we describe our implementation of the system and provide an example of its use in a simple scenario.

Keywords BDI · Dialogues · Explanation

1 Introduction

Belief, Desire Intention (BDI) based approaches to agent reasoning are very popular, with applications ranging from air traffic management [33], to e-Health [13]. The formal basis of BDI systems facilitate formal validation and verification, and provide guarantees as to their actions [12] and to the states the system will, or will not reach. However, understanding why a BDI-based system acted as it did is difficult, requiring working through plans and subplans while tracking the system's internal state.

Researchers have noted that dialogue is a potentially useful tool to explain the behaviour of complex AI artefacts [7], and in this paper we propose a dialogue based approach to reasoning about BDI system behaviour. As our departure point, we consider the case where two dialogue participants (which we may also refer to as agents) hold — possibly different — views about the content of a BDI program and the environment in which it

✉ Nir Oren
n.oren@abdn.ac.uk

Louise A. Dennis
louise.dennis@manchester.ac.uk

[1] University of Manchester, Manchester, UK

[2] University of Aberdeen, Aberdeen, UK

Published online: 06 May 2022

executes. Our dialogue is then designed to pinpoint where disagreement between dialogue participants exists. Such disagreement could, for example, lie in different views regarding what plans drive the BDI system; their priorities; or differences in inputs or initial beliefs of the system. Our dialogue then enables at least one dialogue participant to locate a disagreement (if one exists); and alternatively allows it to determine if no disagreement exists. We assume that in the case of multiple disagreements the dialogue can take place multiple times, with the dialogue participants' beliefs updated between each dialogue instance. Importantly, we do not consider how participants update their beliefs during or following a dialogue, with such belief revision lying outside the scope of the current work.

Our main contribution is the description and formalisation of the explanatory dialogue, enabling the identification of, and explanation for, the reasons why a BDI system behaved as it did. We focus on this formal aspect here and emphasise that natural language generation and utterance presentation from similar work (e.g., [7, 21]) lies outside the scope of this paper. Unlike such work, we do not provide a formal argumentation-based underpinning to our dialogue.

The remainder of the paper is structured as follows. In the next section, we discuss why explanation is necessary, and the benefits of explainability in AI systems. Section 3 introduces a simple BDI language and formalises our environment. Section 4 introduces the dialogue. We examine the properties of the dialogue in Sect. 5 and provide an illustrative example in Sect. 6. Section 8 contains detailed discussion, including a comparison with existing work. Section 9 concludes by considering avenues for future research.

2 Explainable systems

While there is significant interest in explainable AI systems [19], much such work revolves around explaining the inferences made by (otherwise opaque) machine learning systems. In this context, researchers argue that explainability has multiple benefits, including improving stakeholder trust in such systems [27], detecting biases [26], and ultimately improving their design [4].

Our focus here is instead on symbolic systems, and more specifically, BDI-based agent systems. The BDI-based approach to programming follows the declarative logic-programming tradition, based on a folk-psychology model of reasoning. In other words, one could ascribe that a system behaves as it does due to the beliefs, desires and intentions present within the system at some point in time. While BDI systems have other advantages (e.g., naturally reacting to changes in their environment), it has long been claimed that the intuitive understanding of these folk-psychological concepts make programming such systems simple, and that the explanation of behaviour of such systems can follow human intuitions. Implementations of BDI systems have shown that explanations of system behaviour are however not so easy to perform.

One reason for this difficulty — arguably — lies in the way BDI systems are implemented. As discussed in Sect. 3, most BDI systems consist of a set of beliefs (logical facts about the world), a set of desires (implemented as goals the system may wish to achieve) and a set of intentions, instantiated from a set of plans. Plan steps either affect the environment outside the BDI system (by undertaking physical actions in the environment), or affect the system's reasoning process by adding or removing beliefs, or identifying new goals that should be achieved. Given some context (i.e., a set of beliefs held by the system, current goals and possibly events from the environment), a subset of plans is selected to

achieve some goals, and the instantiation of these plans represent the intentions that the system will attempt to achieve.

Plans within the BDI system are usually hierarchical and nested; a high-level plan will specify some goals which lower level plans can achieve. Plans — especially at a low level — are often at too fine a level of granularity to be easily understood, and can also make use of non-intuitive programming language constructs to achieve their low level effects. The cognitive effort required to understand such plans is therefore potentially not much different to that required to understand and debug programs in other programming languages.

In contrast to the above, humans often use enthymemes in explanation to other humans, skip low level details that (they believe) the party they are providing explanation to already knows, and are able to focus on areas where explanation is actually necessary. A fundamental aspect of this is the presence of dialogue. The party obtaining the explanation is able to guide the explainer through the use of questions in the dialogue, homing in on disagreements, skipping points of agreement and what they already know. We therefore believe that the use of dialogue has the potential to drive explanation of BDI systems, both because they map onto human concepts, and — in the case of BDI systems — that such dialogue can allow for utterances about the intuitively understood concepts such as beliefs, goals, desires and intentions. Finally, dialogue also allows us to provide an explanation at the correct level, presenting both high-level beliefs and plans which explain the behaviour, and the lower-level beliefs which trigger or created the higher level beliefs, in response to the requirements of the party requiring explanation.

3 The SimpleBDI language

We begin this section by introducing a language which captures the fundamental features of more complex BDI languages. We then discuss the relationship between our simple language and more feature rich languages.

3.1 SimpleBDI

We introduce a very simple BDI language, SimpleBDI, which we will use to illustrate our ideas. SimpleBDI is designed to be as simple as possible, as its primary purpose is to demonstrate the feasibility of our approach and enable its formalisation. SimpleBDI contains the constructs which lie at the heart of more complex BDI languages, and is therefore an appropriate underlying representation.

A SimpleBDI program consists of a set of plans Π of the form $\pi_{id} : B \rightarrow I$ together with an ordering $\geq$ over these plans. B (the plan's *guard*) is a set of first order ground predicates over some language $\mathcal{L}$, and I is a $[U, do(a)]$ pair. In turn, U is a set of *belief updates* of the form $+b, -b$ where b is a ground first order predicate, and a is an action, again denoted using a ground first order predicate. Since some plans may only update beliefs rather than execute an action, we introduce a special symbol *null* to denote the lack of action. In addition, we assume the existence of an *empty plan* $\pi_{null} : [] \rightarrow [[], do(null)]$. Plans are assumed to be ordered by preference, and we write $\pi \geq \pi'$ if π is either equally or more preferred to π'. Unlike most BDI languages, SimpleBDI does not explicitly model goals. However, goals can be encoded through the introduction of a predicate of the form $goal(g)$, which is added and removed as a belief at appropriate times as part of plan execution.

When writing SimpleBDI programs (e.g., Listing 3), we denote plan orderings by associating a numeric priority with each plan. For example, the following plan has priority 3.

```
take_sample_message -(3)> -take_sample_message, +goal_at_location
```

If multiple plans can be executed, then plans with a higher priority are selected over plans with lower priority. We assume that the empty plan has lower priority than all other plans. While this presupposes a total ordering over plan preferences, we note that this choice is purely syntactic, and SimpleBDI's semantics can also deal with partial preference orderings over plans.

SimpleBDI programs execute plans based on beliefs and changes in the environment (percepts). The latter is captured by an *input trace* τ_e of events external to the agent. Each event is a list of belief updates, V, of the form $+b$ or $-b$ containing a single ground first order predicate. The list of belief updates for a single event cannot contain contradictory belief updates, i.e., $+b, -b \notin V$.

The executor[1] of a SimpleBDI program maintains a set of internal beliefs — denoted $\mathcal{B}$ — encoded as a set of ground first order predicates, and is formally represented at a point in time as a tuple

$$E = \langle \mathcal{B}, \Pi, \geq, \pi, \tau_e, a_{ex}, stage \rangle$$

Here, $\mathcal{B}$ is a set of the executor's beliefs; Π is its plan library and $\geq$ its preference relation over the plans within Π; π the current plan selected for execution; τ_e is the input trace; and a_{ex} the (external) action executed by the executor agent at that time. $stage \in \{\mathtt{s}, \mathtt{p}, \mathtt{e}\}$ captures the current state of the executor; SimpleBDI programs run through repeated perception (p), plan selection (s), and plan execution (e) stages. Given a set of plans Π representing a SimpleBDI program and an input trace τ_e the initial state of the executor is

$$E = \langle [], \Pi, \geq, null, \tau_e, null, \mathtt{p} \rangle$$

Figure 1 summarises the semantics of SimpleBDI describing how the tuple representing the executor evolves as a transition system (i.e., if $E_i \rightarrow E_{i+1}$ in Fig. 1 then E_i becomes E_{i+1} as the system executes). A *program execution trace* is then the sequence of tuples $[E_1, \ldots, E_n]$ where E_{i+1} is obtained by executing a program over the input trace found in E_i until the input trace is empty; the $\emptyset$ symbol denotes the end of program execution.

In the perception phase (p), the top of the input trace (τ_e) is consumed, updating the set of beliefs $\mathcal{B}$. The update itself is done through the *update* function, which takes a set of belief updates and a set of beliefs as input, and returns an updated set of beliefs. Note that during the remaining phases, no beliefs are consumed; a *null* perception is therefore consumed during these phases.

The plan selection phase (s) proceeds by selecting an *applicable* plan using the *select*, *gather* and *applicable* functions respectively. The *applicable* function determines whether a plan is applicable by checking whether the plan's beliefs do, or do not, appear in the belief base. All most preferred applicable plans are collected using the *gather* function, and a plan is then selected from these (via *select*). With no loss of generality, we assume

[1] We use the term "executor" rather than agent to differentiate this entity from the agents undertaking dialogue about the execution of the SimpleBDI program.

$$
\begin{aligned}
\langle \mathcal{B}, \Pi, \geq, \pi, [], a_{ex}, \mathtt{p} \rangle &\rightarrow \emptyset \\
\langle \mathcal{B}, \Pi, \geq, \pi, [V|\tau_e], a_{ex}, \mathtt{p} \rangle &\rightarrow \langle update(V, \mathcal{B}), \Pi, \geq, \pi, \tau_e, null, \mathtt{s} \rangle \\
\langle \mathcal{B}, \Pi, \geq, \pi, [null|\tau_e], a_{ex}, \mathtt{s} \rangle &\rightarrow \langle \mathcal{B}, \Pi, \geq, select(\mathcal{B}, \Pi), \tau_e, null, \mathtt{e} \rangle \\
\langle \mathcal{B}, \Pi, \geq, \pi_{id} : B \rightarrow [U, do(a)], [null|\tau_e], a_{ex}, \mathtt{e} \rangle &\rightarrow \langle update(U, \mathcal{B}), \Pi, \geq, null, \tau_e, a, \mathtt{p} \rangle
\end{aligned}
$$

$$
\begin{aligned}
update([], \mathcal{B}) &= \mathcal{B} \\
update([+b|B], \mathcal{B}) &= update(B, \mathcal{B} \cup \{b\}) \\
update([-b|B], \mathcal{B}) &= update(B, \mathcal{B} \backslash \{b\})
\end{aligned}
$$

$$
\begin{aligned}
select(\mathcal{B}, \Pi) &= \text{an element of } gather(\mathcal{B}, \Pi, \{\}) \\
gather(\mathcal{B}, [], G) &= \begin{cases} \{\pi_{null}\} & \text{if } G = \{\} \\ G & \text{otherwise} \end{cases} \\
gather(\mathcal{B}, [\pi : B \rightarrow I|\Pi], G) &= \begin{cases} gather(\mathcal{B}, \Pi, G) & \text{if } \neg applicable(\mathcal{B}, B) \text{ or there is a } \pi' : B' \rightarrow \_ \in \Pi \text{ s.t. } applicable(\mathcal{B}, B') \text{ and } \pi' \geq \pi \text{ and } \pi \not\geq \pi' \\ gather(\mathcal{B}, \Pi, G \cup \{\pi\}) & \text{otherwise} \end{cases} \\
applicable(\mathcal{B}, []) &= \top \\
applicable(\mathcal{B}, b : B) &= \begin{cases} applicable(\mathcal{B}, B) & \text{if } b \in \mathcal{B} \\ \bot & \text{otherwise} \end{cases} \\
applicable(\mathcal{B}, \neg b : B) &= \begin{cases} applicable(\mathcal{B}, B) & \text{if } b \notin \mathcal{B} \\ \bot & \text{otherwise} \end{cases}
\end{aligned}
$$

Fig. 1 SimpleBDI semantics. $\emptyset$ denotes termination of execution

that this plan is selected at random. If no applicable plan exists, then the empty plan π_{null} is returned. The selected plan is recorded, to be used in the next phase.

Finally, the plan execution phase (e) takes the selected plan and updates the belief base according to the plan's effects. In addition, any action a executed due to the plan is recorded. The cycle then begins again with a new perception phase.

Example 1 Listing 1 shows a simple program in SimpleBDI (i.e., the plans, Π, used by the program executor). In this program a robotic system (for instance a Mars Rover), must move from its starting position to a waypoint and then on to a final location to take a sample. It does this if it believes it has received a message take_sample. It then uses move1 to move from the starting point to the waypoint (if it believes the terrain is safe) and then uses move2 to move from the waypoint to the location where it should take the sample by drilling (again if it believes the terrain is safe). We omit "do(null)" for plans with no associated actions.

Code Listing 1

```
take_sample −(4)>                                                    1
    −take_sample, +goal_at_location                                  2
                                                                     3
safe_terrain, at_start, goal_at_location −(4)>                       4
    +at_waypoint, −at_start,                                         5
    do(move1)                                                        6
                                                                     7
safe_terrain, at_waypoint, goal_at_location −(2)>                    8
    +at_location, −at_waypoint,                                      9
    −goal_at_location, +goal_take_sample,                            10
    do(move2)                                                        11
                                                                     12
at_location, goal_take_sample −(1)>                                  13
    −goal_take_sample,                                               14
    do(drill)                                                        15
```

So for instance, the second plan (lines 4—6) states that if the agent perceives that it is at the start (belief `at_start`), and the terrain is safe (belief `safe_terrain`) and it has received a message telling it to take a sample (belief `goal_at_location`) then it will move to the waypoint (via the external action `move1`), adding the belief that it is at the waypoint (via `+at_waypoint`) to its knowledge base, and no longer believing that it is at the start location (due to the `-at_start` plan effect). At this point, if it no longer perceives the terrain is safe it will not move further. However if it continues to believe the terrain is safe it will move to the final location (using the plan in lines 8—11) and which in turn triggers the remaining plan in the program: to drill for a sample (lines 13—15).

Given a program and an initial state — which includes an input trace — the state of the executor at each step of the program execution trace describes the internal state of the executor and its effects (actions) on the environment. Table 1 illustrates an example trace obtained by running 1. To improve readability, we only show the head of the event stack at each time point.

3.2 Relationship of SimpleBDI to existing BDI languages

SimpleBDI lacks a number of features that are common in existing BDI languages such as Jason [6] and 2APL [11]. Key among these are the existence of goals as first class objects, the sequencing of multiple actions within plans, and the use of multiple intentions (allowing, for instance, multiple goals to be pursued at once).

We argue that these features are important for programming and engineering purposes, but they complicate the formalism, and their behaviour can be reproduced in SimpleBDI—though at the expense of more verbose and opaque programs. This means that languages containing such features can still employ the dialogue mechanism proposed here in the context of SimpleBDI. We present below an informal transformations from programs with these features into programs without them. We note that, in practice, adapting a language with such features to produce dialogues *meaningful to a user or programmer* by transforming programs and traces in such a language into ones equivalent to SimpleBDI programs and traces will be non-trivial.

Table 1 The trace obtained when executing Code Listing 1

Time	Stage	Event stack head	Beliefs	Current plan	Action
0	p	+take_sample, +at_start, +safe_terrain	Ø	null	null
1	s	null	{at_start, take_sample, safe_terrain}	null	null
2	e	null	{at_start, take_sample, safe_terrain}	take_sample-(4)> -take_sample, +goal_at_location	null
3	p	null	{at_start, safe_terrain, goal_at_location}	null	null
4	s	null	{at_start, safe_terrain, goal_at_location}	null	null
5	e	null	{at_start, safe_terrain, goal_at_location}	goal_at_location, at_start, safe_terrain-(4)> +at_waypoint, -at_start,do(move1)	null
6	p	null	{at_waypoint, safe_terrain, goal_at_location}	null	move1
7	s	null	{at_waypoint, safe_terrain, goal_at_location}	null	null
8	e	null	{at_waypoint, safe_terrain, goal_at_location}	at_waypoint, safe_terrain, goal_at_location-(2)> +at_location, +goal_take_sample, -at_waypoint, -goal_at_location, do(move2)	null
9	p	null	{at_location, goal_take_sample, safe_terrain}	null	move2
10	s	null	{at_location, goal_take_sample, safe_terrain}	null	null
11	e	null	{at_location, goal_take_sample, safe_terrain}	at_location, goal_take_sample-(1)> do(drill), -goal_take_sample	null
12	p	null	{at_location, safe_terrain}	null	drill

3.2.1 Goals

As can be seen in our sample programs we advocate representing goals as beliefs in SimpleBDI. BDI theory categorises goals as *perform goals*, *achievement goals* and *maintenance goals* each with different behaviours. In general maintenance goals are not widely implemented as first class objects so we omit discussion of them here. Languages vary over whether they implement perform and/or achievement goals. A perform goal exists as the trigger for the application of a plan but there is no check on whether the goal itself is achieved after the plan has been executed. The behaviour of this type of goal is easily reproduced using beliefs since the belief can be used in a SimpleBDI plan guard and the belief itself then removed by the plan's belief updates. An achievement goal persists until some state of the world is bought about – typically given some logical term t a goal to achieve t persists until a belief t is acquired. In this case, again, we can model the goal as belief but instead of removing the goal in the belief update of any plan that includes the goal in its guard, we instead have a separate plan which removes the belief representing the goal when the desired state of the world comes about. An example of how SimpleBDI implements perform goals is shown in the last plan of code listing 1, where `goal_take_sample` is removed when drilling occurs, while an achievement goal is captured in the third plan of code listing 1, which checks whether one has reached the goal location before removing it form the belief base.

3.2.2 Action sequences

A plan that contains a sequence of actions can be implemented by chaining together a set of SimpleBDI plans. For instance, suppose in some situation, represented by a plan guard, G, we want to execute a_1 followed by a_2 and, potentially make some set of belief updates, U. This means we want a plan of the form $\pi : B \rightarrow [U, do(a_1);do(a_2)]$ - we represent this in SimpleBDI as $\pi_1 : B \rightarrow [+b_1, do(a_1)]$ (a plan to perform the first action and then add some "placeholder" belief b_1) and then a second plan $\pi_2 : [b_1] \rightarrow [U \cup \{-b_1\}, do(a_2)]$ (a plan that performs the second action, removes the placeholder belief and performs the desired plan update). Assuming an appropriate priority ordering on plans this will have the same behaviour as a single plan that sequences the two actions.

3.2.3 Multiple intentions

Multiple intentions are needed in the context of sequences of actions in plans where there may be occasions where interleaved execution of actions between two plans is required. For instance, the program might have a plan that moves a robot to some location and performs a measurement, but also needs to have a second plan that can execute an emergency halt or take additional measurements in response to specific stimuli while the first movement action is executing. Since we do not have sequencing of actions within plans, we do not actually need multiple intentions to reproduce this behaviour but can instead rely on appropriate priorities between single action plans. This is illustrated in code listing 2 where movement will occur until the system detects it is in danger, and then prioritise corrective action over further movement.

Code Listing 2

```
not_at_goal -(1)> do(move)                                    1
in_danger -(2)> +stopped, do(emergency_stop)                  2
stopped -(2)> do(remove_danger), -stopped                     3
```

The ease with which goals and multiple intentions can be encoded within SimpleBDI leads us to believe that most programs in well-known BDI languages can easily be transformed to SimpleBDI programs and, similarly, that their traces can be transformed into SimpleBDI traces for the purposes of generating explanatory dialogues. The formal parts of our dialogue mechanism therefore do not need to explicitly handle these features though we note that the presentation of utterances to users in a way that is meaningful given their understanding of an agent's program will be non trivial in more complex languages; we leave further investigation of this to future work.

4 Dialogues

The semantics of SimpleBDI allow us to determine how a program will execute (for a given initial state). However, systems executing such programs are often opaque, and understanding why some behaviour occurred with only partial knowledge of an execution's inputs and internal workings may be difficult, requiring (at best) tracing through multiple layers of plans, and (at worst) guessing as to what some beliefs and system events were. The aim of this paper is to help facilitate an understanding of program behaviour in such situations.

To this end we consider a dialogue between two participants who may have partial access to the program execution trace, and have their own model of the executing system. Our dialogue seeks to identify differences between the participants' models so as to identify disagreements. Such differences could arise due to differences in the plans the dialogue participants believe the executor has; a divergence with regards to the beliefs they believe the executor holds; or different beliefs they have with regards to the various traces. If one of the participants is the program executor (whose trace is correct), and another is a human or system trying to understand the executor's behaviour, then identifying a disagreement means that an error in the latter's assumptions or reasoning has been identified, and doing so serves as a form of explanation of the executor's behaviour.

We begin by providing the intuition behind our dialogue, after which we describe a model of the dialogue participants. Finally, we formalise the dialogue by describing the *utterances* or *moves* participants may make in the dialogue (c.f., dialogue games [32]).

4.1 Dialogue — Intuitions

When applying the semantics correctly, differences between execution traces between dialogue participants arise due to differing plans within agent plan libraries or plan precedence, due to different perceptions from the environment, or due to different initial beliefs, as these drive the execution of the system and the resultant execution trace. The only externally visible effects of a running system are the actions it executes within the environment,

and our dialogue therefore begins by having one participant asking the other why, or why not, an action was performed at some time.

Let us consider the evolution of a possible explanatory dialogue. If a dialogue participant asks another why an action did not take place, the latter can respond by asking the former why they believe an action did take place. If on the other hand, a participant asks why an action did take place, the explanation (i.e., response) involves identifying the (executed) plan which triggered the action. When asked why a plan was executed, the response involves demonstrating that the set of beliefs which triggered the plan held. When asked why some belief held, a response involves either presenting the percept which caused the belief, or the plan which led to the belief being adopted. In the latter case, the dialogue can continue by providing an explanation for the plan.

When an assertion regarding a belief is presented it is also possible for a disagreement to occur with the other dialogue participant asserting that the belief does not hold at the relevant point in time. In such a situation, the dialogue can continue with the presentation of a plan or percept which removes the belief. In the former case, the dialogue can continue by providing an explanation for the plan. In the latter case, the presentation of the percept should identify a disagreement between the dialogue participants.

The above paths through the dialogue help us identify natural points of dialogue termination. When a percept justifying a belief is presented, no further explanation is possible, as such a percept originates from outside the BDI system. When stating that a plan was executed, if the other dialogue participant is not aware of the plan (i.e., the plan is not present in their plan library), or if they believe that a higher precedence plan exists, then a disagreement has been identified which cannot be resolved by further discussion regarding system execution. However, if an alternative plan is advanced, then the dialogue can continue either by explaining why the plan should not have been executed (e.g., due to it having a lower priority), or the agent querying what beliefs hold for the alternative plan to have run. In the latter case, the dialogue participants reverse their roles, but the dialogue can continue. Finally, If one dialogue participant asks another why an action took place (or didn't take place), and the latter believes that the action didn't take place (did take place), then no further discussion is possible. Figure 2 graphically describes this ordering of utterances with regards to other utterances.

4.2 Dialogue participant model

A dialogue participant is a tuple $\langle \mathcal{M}, \mathcal{O}, \overline{\mathcal{O}} \rangle$ where $\mathcal{M}, \mathcal{O}$ and $\overline{O}$ are program execution traces of a BDI program, and $|\mathcal{M}| = |\mathcal{O}| = |\overline{\mathcal{O}}|$.

Informally $\mathcal{M}$ represents the participants model of what *should* have happened – i.e., the program execution trace they believe to be correct, $\mathcal{O}$ represents their (partial) understanding of what the other participant's trace looks like. $\overline{\mathcal{O}}$ then captures commitments or constraints that emerge on the other participant due to their utterances – specifically plans the other participant has explicitly committed to *not* having been selected; beliefs explicitly committed to *not* having been perceived on the input trace; and actions explicitly committed to *not* having been performed[2].

We index a specific time point within the execution trace using array notation (e.g., $\mathcal{M}[5]$). Where the context is clear, we index individual portions of a BDI executor's

[2] We observe that some elements of the tuples stored within $\overline{O}$ are not referred to by the dialogue.

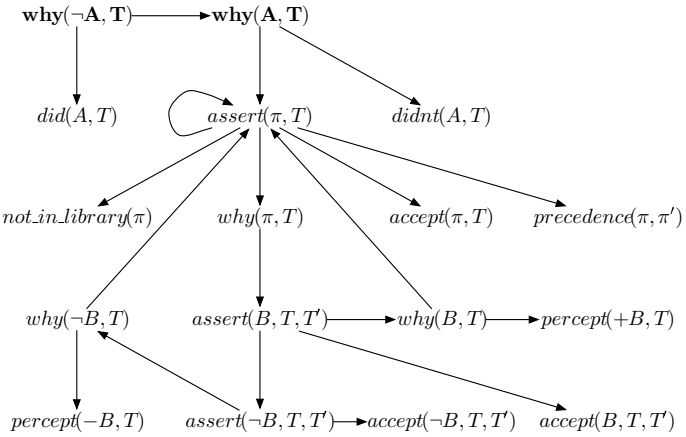

Fig. 2 A graphical description of which moves can follow which moves in the dialogue. Bold text identifies the two possible starting moves

state at a specific time in the same manner, identifying the program with a superscript. For example, $a_{ex}^{M}[5]$ refers to a_{ex} of BDI program execution trace M at time 5. Equivalently, if — for example — some $x \in \tau_e^M[5]$, we may say that x holds in τ_e^M at time 5. We refer to elements within $\overline{\mathcal{O}}$ as $\overline{\mathcal{B}}, \overline{\pi}$ etc, indexing individual entries by time. We note that — in the present system — the plan library Π does not change, and therefore abuse notation by referring to it without identifying a specific time point; we assume that any operations on $\Pi^{\mathcal{M}}, \Pi^{\mathcal{O}}$ and $\overline{\Pi}$ apply to all time indices. Finally, we also assume that $stage^{\mathcal{M}}[T] = stage^{\mathcal{O}}[T] = stage^{\overline{\mathcal{O}}}[T]$ for all $0 \leq T < |\mathcal{M}|$.

Utterances made by one dialogue participant can affect the other participant's view of the utterer. Therefore, given one dialogue participant $\langle \mathcal{M}, \mathcal{O}, \overline{\mathcal{O}} \rangle$, we refer to the other dialogue participant as $\langle \mathcal{M}', \mathcal{O}', \overline{\mathcal{O}}' \rangle$. We can, for example, index the other dialogue participant's view of its own input trace at time T as $\tau_e'^{\mathcal{M}}[T]$.

The purpose of our dialogue is to allow a participant to identify disagreements or inconsistencies between itself and the other participant. Such disagreements can be recognised as occurring between the $\mathcal{M}$ and $\mathcal{O}$ traces, or between the $\mathcal{M}$ and $\overline{\mathcal{O}}$ traces.

- A disagreement can be detected when there is a difference in beliefs. That is, when one dialogue participant is aware that some of the other participant's beliefs are not a subset of its own, or that the other participant is committed to facts that this participant knows do not hold. Formally, for an index T, if $\mathcal{B}^{\mathcal{O}}[T] \not\subseteq \mathcal{B}^{\mathcal{M}}[T]$, or if $\overline{\mathcal{B}}[T] \cap \mathcal{B}^{\mathcal{M}}[T] \neq \emptyset$ then a disagreement in belief has been identified. More specifically, the disagreement rests on beliefs $(\mathcal{B}^{\mathcal{O}}[T] \backslash \mathcal{B}^{\mathcal{M}}[T]) \cup (\overline{\mathcal{B}}[T] \cap \mathcal{B}^{\mathcal{M}}[T])$.
- Another source of disagreement occurs when plan libraries differ between participants, or plan priorities differ. That is, if $\Pi^{\mathcal{O}} \not\subseteq \Pi^{\mathcal{M}}$, or if $\pi > \pi'$ according to $\Pi^{\mathcal{M}}$, and $\pi \not> \pi'$ according to $\Pi^{\mathcal{O}}$, or if there is some $\pi \in \overline{\Pi}, \Pi^{\mathcal{M}}$ then a disagreement w.r.t. the plan library has been identified. Such a dispute revolves around plans π, π' or π respectively.
- Disagreements can also occur when the participants disagree about which plan is executed at some point in time. For an index T, if $\pi^{\mathcal{O}}[T], \overline{\pi}[T] \neq null$ and $\pi^{\mathcal{M}}[T] \neq \pi^{\mathcal{O}}[T]$

or $\pi = \overline{\pi}[T]$ then a disagreement in the executing plan has been identified. This dispute centers on plan π.

- Another source of disagreement can arise if the participants believe that the physical environment (in terms of percepts arriving to the system) differ. That is, if at any time T the head of $\tau_e^{\mathcal{M}}$ differs from the head of $\tau_e^{\mathcal{O}}$ and $\tau_e^{\mathcal{O}} \neq null$, then a disagreement in perception (w.r.t. the respective heads of the lists) has been identified.
- Finally, a disagreement can revolve around what action the participants believe has been executed by the system. This occurs if at any time T, $a_{ex}^{\mathcal{M}}[T] \neq a_{ex}^{\mathcal{O}}[T] \neq null$ or $a_{ex}^{\mathcal{M}}[T] = \overline{a_{ex}}[T]$, then a disagreement in action has been identified, based on the actions identified.

For a disagreement to occur, the relevant element of $\mathcal{O}$ and $\overline{\mathcal{O}}$ must not be $\emptyset$. This reflects the fact that as the dialogue progresses, $\mathcal{O}$ and $\overline{\mathcal{O}}$ are updated and a disagreement only occurs when an explicit difference is found.

4.3 Dialogue initiation and termination

At the start of a dialogue all elements of $\mathcal{O}$ and $\overline{\mathcal{O}}$ at all times are set to *null* (or $\emptyset$ for beliefs), reflecting a lack of knowledge a dialogue participant has about the other participant.

The dialogue begins when one dialogue participant makes a $why(A, T)$ or $why(\neg A, T)$ move, asking why an action, A was, or was not performed at time T.

The dialogue continues as additional utterances are made by the participants in response to previous utterances. Utterances are *open* until their *closure condition* occurs in the dialogue, at which point they are *closed*. The dialogue terminates when no open utterances exist, i.e., when there is no legal move that any dialogue participant can make. Once the dialogue terminates, disagreement(s) can be identified using the procedure described in the previous section[3].

4.4 Utterances

During a dialogue, participants make different utterances. Table 2 describes these, when they can be made (i.e., what move must be open for the utterance to be made), and their intuitive meaning, aligning with the high level dialogue description provided in Sect. 4.1 and Fig. 2. Note that $why(A, T)$ can be used to initiate the dialogue, or made in response to a $why(\neg A, T)$ move. In other words, if a participant asks "Why did action A not occur?", asking "Why do you think action A should have occurred?" is a valid response, as it will allow for a disagreement in views to be detected. Also note that the assertion of a plan ($assert(\pi, T)$) can be made in response to asking why an action took place, why a belief was instantiated, or in response to the claim that some other plan should have been executed. The intuition behind the latter is that a dialogue participant suggests that another plan should have been executed. The dialogue can then continue to investigate why this is the case.

[3] The dialogue itself can be viewed as a collection of utterances. The status of each utterance can easily be computed based on other utterances present within the dialogue.

Table 2 Legal dialogue utterances and what moves they follow, as well as their intuitive meaning

Utterance	Follows	Intuition
$why(\neg A, T)$	None	Asks why action A did not take place at time T.
$why(A, T)$	None $why(\neg A, T)$	Asks why action A took place at time T.
$did(A, T)$	$why(\neg A, T)$	Asserts that action A took place at T. Ends the dialogue.
$didnt(A, T)$	$why(A, T)$	Asserts that action A did not occur at time T. Ends the dialogue.
$assert(\pi, T)$	$why(A, T+1)$ $why(B, T+1)$ $why(\neg B, T+1)$ $assert(\pi', T)$	Asserts that plan π was selected for execution at time T in response to a question regarding why an action or belief held, or to counter a claim that another plan was executed at time T.
$not\_in\_library(\pi)$	$assert(\pi, T)$	States that the dialogue participant is not aware of plan π. Ends the dialogue.
$precedence(\pi, \pi')$	$assert(\pi, T)$	Follows a second $assert(\pi', T)$ move. States that plan π has equal or greater precedence than π'. Ends the dialogue.
$accept(\pi, T)$	$assert(\pi, T)$	Accepts that plan π was selected for execution at time T.
$why(\pi, T)$	$assert(\pi, T)$	Asks why plan π was selected for execution at time T.
$assert(B, T, T')$	$why(\pi, T'+1)$	Asserts that belief B exists at all times between T and T' (inclusive).
$assert(\neg B, T, T')$	$assert(B, T'', T')$	Asserts that B did not exist at all times between T and T'.
$accept(B, T, T')$	$assert(B, T, T')$	Accepts that B holds between T and T'.
$accept(\neg B, T, T')$	$assert(\neg B, T, T')$	Accepts that B does not hold between T and T'.
$why(B, T)$	assert(B, T, T')	Asks why belief B exists at time T.
$why(\neg B, T)$	assert$(\neg B, T, T')$	Asks why belief B does not exist at time T.
$percept(+B, T)$	$why(B, T+1)$	Explains that B was perceived at time T in response to asking why it held at the next time point.
$percept(-B, T)$	$why(\neg B, T+1)$	Explains that B was perceived being removed at time T.

While Table 2 specifies what utterance can be made in response to a move, the contents of a legal utterance are further constrained. Table 3 provides a semi-formal description of each utterance, stating when a move can be made (the move condition), the move's closure condition, and move's effect on dialogue participants. Within the table, _ is used, as in Prolog, to indicate that any instantiation of the relevant value may exist.

We assume that the same move cannot be repeated. A dialogue D is then a sequence of moves $[D_1, \ldots, D_n]$ obeying all dialogue constraints (i.e., "Follows" requirements of Table 2, and "Move" and "Closure" conditions of Table 3). Note that we do not specify an explicit turn taking mechanism. Rather, dialogue participants make utterances in response to an open move subject to the move conditions. Different instantiations of the dialogue are therefore possible whereby, for example, an agent can respond to a question about why a plan holds by responding with a single belief assertion at a time, or by asserting all elements of the plan's guard simultaneously. While this may have an impact on dialogue understanding and dialogue length (which we will categorise as part of future work), the entire dialogue family will yield equivalent results in terms of the dialogue's goals (i.e., in identifying disagreements).

Moves such as $accept(\pi, T)$ which always close a dialogue branch either explicitly indicate agreement or disagreement with an utterance previously made by the other dialogue participant. In the latter case, they typically end the dialogue. Other moves are closed when the appropriate closure move exists. This closure move either identifies the disagreement, or refines where scope for disagreement exists. For example, when one participant asserts a plan was executed, and the other responds by asserting that some other plan was executed, participants no longer needs to discuss the former plan to identify disagreement. Instead, identifying *why* the latter plan was (believed to be) executed is enough to identify the disagreement.

The *precedence* utterance sets a constraint between π and π'. We assume in addition that the effect maintains a total ordering over plans in Π. We omit the requirement that $\pi' > \pi$ appear in $\overline{\Pi'}$ as the utterance's effect is sufficient to detect disagreement.

Note that there is an asymmetry with regards to closure conditions between $assert(B, T, T')$ and $assert(\neg B, T, T')$. The former can be closed by the latter, but not the other way around. The intuition behind this is that the assertion of a belief must identify the maximal interval during which the belief held. Providing an overlapping interval where it does not hold counters the assertion, but the new assertion must be explained (via a $why(\neg B, T)$ move) rather than requiring another assertion for the belief holding.

Finally, note that *why* moves have no effect on the dialogue participants, as such moves simply request more information without committing the utterer to any specific stance. However, the condition for uttering such a why move requires that the utterer have appropriate beliefs (e.g., for $why(B, T)$, the utterer has to believe that belief B held at time T. We do not impose a similar constraint when asking why an action did/didn't take place. Such utterances initiate the dialogue and requires a participant to believe that the other believes the action did/didn't take place but places no requirements on the utterer (i.e., constraints on $\mathcal{M}$), and without a response, does not constrain the other (i.e., does not constrain $\mathcal{O}$).

Table 3 Preconditions for an utterance; requirements to label the move closed; and utterance effects on dialogue participants

Utterance	Move condition	Closure condition	Effect
$why(\neg A, T)$	$stage^{\mathcal{M}}[T] = \mathtt{e}$ and there is no other $why(\neg A, \_)$ utterance in the dialogue.	$did(A, T)$ or $why(A, T)$ in the dialogue	None
$why(A, T)$	$stage^{\mathcal{M}}[T] = \mathtt{e}$ and there is no other $why(A, \_)$ utterance in the dialogue.	$didnt(A, T)$ or $assert(\pi, T-1)$ s.t. $\pi = \_ \rightarrow [\_, do(A)]$ in the dialogue.	None
$why(\pi, T)$	$\pi = \pi^{\mathcal{O}}[T]$	If π is of the form $[b_1, \ldots, b_n] \rightarrow [U, \_]$ then there is a move $assert(b_i, \_, T-1)$ for all $i = 1, \ldots, n$.	None
$why(B, T)$	$B \in \mathcal{B}^{O}[T]$	When there is a move $percept(+B, T-1)$ or a move $assert(\pi, T-1)$ such that π is of the form $\_ \rightarrow [U, \_]$ and $+B \in U$.	None
$why(\neg B, T)$	$B \notin \mathcal{B}^{\mathcal{O}}[T]$	When there is a move $percept(-B, T-1)$ or a move $assert(\pi, T-1)$ such that π is of the form $\_ \rightarrow [U, \_]$ and $-B \in U$.	None
$assert(\pi, T)$	$stage^{\mathcal{M}}[T] = \mathtt{s}$ and $\pi = \pi^{\mathcal{M}}[T]$. If following $why(A, T+1)$ (equivalently $why(B, T+1)$ or $why(\neg B, T+1)$) then π is of the form $\_ \rightarrow [\_, do(A)]$ (equivalently $\_ \rightarrow [[\ldots, +B, \ldots], \_]$ or $\_ \rightarrow [[\ldots, -B, \ldots], \_]$).	Dialogue contains one of the following: $why(\pi, T)$ $assert(\pi', T)$ $accept(\pi, T)$ $not\_in\_library(\pi)$ $precedence(\pi, \pi')$	$\pi'^{\mathcal{O}}[T] = \pi$. If following an $assert(\pi', T)$ then π is added to $\overline{\pi}'[T]$.
$assert(B, T, T')$	$B \in \mathcal{B}^{\mathcal{M}}[i]$ such that $T \leq i \leq T'$	Dialogue contains $why(B, T')$ or $accept(B, T, T')$ or $assert(\neg B, T'', T')$ (where $T'' \leq T'$).	B is added to all $\mathcal{B}'^{\mathcal{O}}[i]$ for all $T \leq i \leq T'$
$assert(\neg B, T, T')$	$B \notin \mathcal{B}^{\mathcal{M}}[i]$ such that $T \leq i \leq T'$	Dialogue contains $why(\neg B, T')$ or $accept(\neg B, T, T')$	B is added to $\overline{\mathcal{B}'}[i]$ for all $T \leq i \leq T'$
$did(A, T)$	$A = a_{ex}^{\mathcal{M}}[T]$	Always closed	$a'^{\mathcal{O}}_{ex}[T] = A$
$didnt(A, T)$	$A \neq a_{ex}^{\mathcal{M}}[T]$	Always closed	$\overline{a'_{ex}}[T] = A$
$not\_in\_library(\pi)$	$\pi \notin \Pi^{\mathcal{M}}$	Always closed	π is added to $\overline{\Pi'}$
$precedence(\pi, \pi')$	Dialogue has $assert(\pi, T)$, $assert(\pi', T)$. $\pi, \pi' \in \Pi^{\mathcal{M}}$ and $\pi \geq \pi'$ in $\Pi^{\mathcal{M}}$	Always closed	$\pi > \pi' \in \Pi'^{\mathcal{O}}$
$accept(\pi, T)$	$\pi = \pi^{\mathcal{M}}[T]$	Always closed	$\pi'^{\mathcal{O}}[T] = \pi$
$accept(B, T, T')$	$B \in \mathcal{B}^{\mathcal{M}}[i]$ for all $T \leq i \leq T'$	Always closed	B is added to $\mathcal{B}'^{\mathcal{O}}[i]$ for all $T \leq i \leq T'$
$accept(\neg B, T, T')$	$B \notin \mathcal{B}^{\mathcal{M}}[i]$ for all $T \leq i \leq T'$	Always closed	B is added to $\overline{\mathcal{B}'}[i]$ for all $T \leq i \leq T'$

Table 3 (continued)

Utterance	Move condition	Closure condition	Effect
$percept(+B, T)$	$stage^{\mathcal{M}}[T] = \mathtt{p}$ and $+B$ is contained within the set at the head of $\tau_e^{\mathcal{M}}[T]$	Always closed	B is added to ${\mathcal{B}'}^{\mathcal{O}}[T+1]$
$percept(-B, T)$	$stage^{\mathcal{M}}[T] = \mathtt{p}$ and $-B$ is contained within the set at the head of $\tau_e^{\mathcal{M}}[T]$, or $T = 0$ and $B \notin \mathcal{M}[0]$	Always closed	B is added to $\overline{\mathcal{B}'}[T+1]$

5 Dialogue properties

Having described the utterances dialogue participants can make, as well as how a dialogue is initiated and terminates, we now turn our attention to the properties of the dialogue.

The first property we consider reflects the fact that the model held by one dialogue participant of the other always reflects the latter's true internal state if it did so previously.

Proposition 1 *If, before move D_i, for all indexes T, ${\mathcal{B}'}^{\mathcal{O}}[T] \subseteq \mathcal{B}^{\mathcal{M}}[T]$, ${\Pi'}^{\mathcal{O}} \subseteq \Pi^{\mathcal{M}}$, ${\pi'}^{\mathcal{O}}[T] \in \{\emptyset\} \cup \{\pi^{\mathcal{M}}[T]\}$, ${\tau'_e}^{\mathcal{O}}[T] \in \{\emptyset\} \cup \{\tau_e^{\mathcal{M}}[T]\}$, and ${a'_{ex}}^{\mathcal{O}}[T] \in \{\emptyset\} \cup \{a_{ex}^{\mathcal{M}}[T]\}$, then this will also be the case following the move.*

Proof We note that *why* moves do not affect the traces, and therefore only consider the remaining move types.

Those moves which update an element of $\mathcal{O}'$ do so in a way consistent with $\mathcal{M}$, giving us the desired result. □

Note that the update procedure described above also updates the constraints in $\overline{\mathcal{O}'}$ in a manner consistent with $\mathcal{M}$. Therefore, $\mathcal{O}', \overline{\mathcal{O}'}$ are consistent with the program execution trace $\mathcal{M}$.

Corollary 1 *Given two dialogue participants $\langle \mathcal{M}, \mathcal{O}, \overline{\mathcal{O}} \rangle$, $\langle \mathcal{M}', \mathcal{O}', \overline{\mathcal{O}'} \rangle$, if $\mathcal{M}$ and $\mathcal{O}'$ contain no contradictions prior to move D_i, then they will contain no contradictions following it.*

The next property we demonstrate is that a dialogue participant can always respond to an open move. We do this by considering each utterance individually.

Proposition 2 *Given a dialogue and an open move, a response to the open move is always possible.*

Proof We provide an exhaustive proof. Consider two dialogue participants $\langle \mathcal{M}, \mathcal{O}, \overline{\mathcal{O}} \rangle$, $\langle \mathcal{M}', \mathcal{O}', \overline{\mathcal{O}'} \rangle$, and a dialogue D with some open move m. With no loss of generality, assume that there is an open utterance made by the first agent. We begin by noting that if this move is one of $did(A, T), didnt(A, T), not\_in\_library(\pi), accept(\pi, T)$, $precedence(\pi, \pi')$, $percept(\_B, T)$ or $accept(\_B, T, T')$ then these are closed, and therefore do not consider these moves further. For the remaining moves, we show that there is always at least one legal response.

$why(\neg A, T)$ Here, A is either in the second participant's trace at time T, or not. If it is, then $did(A, T)$ is a possible response. However, even if this is not the case, $why(A, T)$ is always a legal response.

$why(A, T)$ If A is not in the second participant's trace at time T, then $didnt(A, T)$ is a possible move. If it is in the trace, then there must have been a plan which caused the action to take place within the trace, which was selected at the previous time point for execution, meaning that $assert(\pi, T-1)$ can be uttered.

$assert(\pi, T)$ If the plan is not in the second participant's plan library, they can respond with $not\_in\_library(\pi)$. If the plan appears in their own trace at time T, they can accept it. If another plan π' appears in their trace at time T, and π can also be selected for execution,

they can utter $precedence(\pi, \pi')$. Finally, if they are aware of the plan, they can also always utter $why(\pi, T)$.

$why(\pi, T)$ For this to have been uttered by the first participant, the second participant must have uttered $assert(\pi, T)$, and the plan must have been selected for execution within their trace. This means that the beliefs which allowed the plan to be triggered must be present, meaning that these can be asserted (via $assert(B, T, T')$ until the $why(\pi, T)$ move is closed.

$assert(B, T, T')$ If the belief holds between time T and T' in the second participant's trace, then they can utter $accept(B, T, T')$. Alternatively, they can always question $why(B, T)$, or — if there is a time-point in their trace $\mathcal{M}'$ where B does not hold, they can respond with $assert(\neg B, T, T')$.

$why(B, T)$ This was uttered in response to an $assert(B, T, T')$ which means that the belief appeared in the utterer's trace at time T. This was either due to a percept occurring at the previous time-point within that trace, allowing for a $percept(+B, T-1)$ to be uttered, or alternatively, as the effect of a plan executed in the previous time point, allowing $assert(\pi, T-1)$ to be uttered.

$assert(\neg B, T, T')$ If the belief is not in the respondent's trace, they can$accept(\neg B, T, T')$[4]. The only other response is asking $why(\neg B, T)$ which is always a valid response.

$why(\neg B, T)$ This was uttered in response to an $assert(\neg B, T, T')$ which means the belief disappeared from the utterer's trace at time T. This was either due to a percept removing the belief (allowing $percept(-B, T-1)$[5] to be uttered), or due to a plan removing the belief, allowing $assert(\pi, T-1)$ to be uttered. □

Next, we demonstrate that our dialogues always terminate.

Proposition 3 *Given a finite set of plans Π with a finite set of propositions in their guards, G, then any dialogue starting with a why question on an action will terminate.*

Proof We know from proposition 2 that a response is always possible. We need to show that this response must either close the dialogue, or (eventually) make moves which only refer to previous time points. Since the lowest possible time is 0, and no move may refer to time points before this, this is enough to demonstrate dialogue termination. We consider possible moves individually.

$why(\neg A, T)$ A $did(A, T)$ response closes the dialogue immediately. $why(A, T)$ is the only other possible response.

$why(A, T)$ A $didnt(A, T)$ closes the dialogue immediately. The only other response, $assert(\pi, T-1)$ refers to a previous time point.

$assert(\pi, T)$ is closed immediately by an *accept*, *not_in_library* or *precedence* move. We therefore need only consider $why(\pi, T)$ or $assert(\pi', T)$. We note that in the latter case, given the move's conditions, a third *assert* cannot occur, meaning that $why(\pi', T)$ will be asked in response, or a closing move will be made. We therefore need only consider $why(\pi, T)$.

[4] Note that this should never occur as this assertion follows an $assert(B, T, T')$ from the other dialogue participant, requiring the belief to have held in their trace.

[5] Note that if $T = 0$, this utterance can be made if $B \notin \tau_e^{\mathcal{M}}[0]$.

why(π, T) The only legal response is a set of *assert*$(B, T', T-1)$ such that $T' \leq T-1$. Therefore, each consider a time before T.

assert(B, T', T) An *accept*(B, T', T) will close this move. We must therefore consider *assert*$(\neg B, T'', T)$ and *why*(B, T). We show that responses to these will either close the dialogue or deal with earlier time points.

assert$(\neg B, T', T)$ An *accept*$(\neg B, T', T)$ will close this move, meaning we need to show that the only other legal response — *why*$(\neg B, T)$ ends up considering time points previous to T.

why(B, T) and *why*$(\neg B, T)$. Both of these moves are closed by appropriate *percept* moves. The only alternative is an *assert*$(\pi, T-1)$ move, which considers time $T-1$.

It is also clear that given the requirements that *stage*$[0] = \mathrm{p}$, and that *percept* moves are the only ones which can refer to this time, that no dialogue will refer to an earlier time point than $T = 0$. □

Turning to the question of dialogue complexity, we demonstrate that the worst case length of a dialogue depends on the number of plans in the plan library and the size of plan guards for each plan.

Corollary 2 *The complexity of creating a dialogue is polynomial in the size of the plan library and plan's guard.*

Proof Let k be the total number of plans in the plan library. Since moves cannot be repeated the maximum number of (plan) asserts which can take place in any branch of the tree is k. Furthermore, the dialogue can branch whenever a *why*(π, T) is asked, with a factor equal to the number of beliefs in the guard of the plan (call these g). Finally, from the previous theorem, we know that a dialogue can take place for at most T time points, meaning that the upper bound for the number of moves is $O(g^{Tk})$. □

The following proposition states that if a disagreement within the dialogue participant's views (M) exists, its root cause — the difference in plans, perceptions or beliefs which led to it — can be detected by the dialogue, assuming that some aspect of the disagreement was already known to the dialogue participants (e.g., a difference in perceived action).

Proposition 4 *Given two agents for which* $\mathcal{M} \neq \mathcal{M}_{\simeq}$ *and for which* $a_{ex}^{\mathcal{M}}[T] \neq a_{ex}^{\mathcal{M}'}[T]$, *there is a dialogue which terminates with a not_in_library, precedence or percept move.*

Proof We know from Proposition 3 that all dialogues terminate. We show that there is at least one belief or plan that is not *accept*ed.

Note that since there is a disagreement in actions, the dialogue can initiate by asking why the disagreed upon action was executed, meaning that we can ignore *did*/*didnt* moves.

Assume the proposition is false. This would mean that there is agreement on which plan was executed. But there is a disagreement in actions, which means that $\mathcal{M}$ believes a plan with head $a_{ex}^{\mathcal{M}}[T]$ was executed, while $\mathcal{M}'$ believes a plan with head $a_{ex}^{\mathcal{M}_{\simeq}}[T]$. Therefore this is a contradiction. The only way to close the dialogue is either with a *not_in_library*, or *precedence* move, asking *why* the plan was selected, or for another plan to be asserted leading to the same arguments as above. Therefore only the question regarding *why* a plan was chosen does not (eventually) close the move. In turn, this leads to a disagreement about

beliefs, which can only be resolved via a *percept* disagreement, or by asking about further plans, over which there must (as above) be a disagreement. □

It follows trivially that at least one dialogue participant will be able to identify the disagreement by examining their $\mathcal{M}$ and $\mathcal{O}$.

The results above suggests a simple strategy for identifying the root cause of a disagreement, namely to never *accept* when a disagreement exists, and always ask *why* about such disagreements. Such *disagreement* dialogues can be contrasted from *confirmatory* dialogues, where one participant may wish to confirm that the other's internal trace matches their own. A simple strategy for such confirmatory dialogues involves always asking *why* where possible, *accepting* only when no other move exists.

Finally it can be easily shown that if both $\mathcal{M}$ and $\mathcal{M}'$ are identical, all dialogues will terminate with *did*/*didnt* moves (if the initial *why* asks about a move that did/didn't occur), *accept*, or *percept* moves. In other words, no disagreement will be identified.

6 Implementation

We have implemented SimpleBDI and our dialogue explanation system in Python[6]. In our system two agents execute the program. Perceptions are supplied to each agent individually at certain time steps – allowing for differences in execution to occur because of differing perceptions. Once execution is completed, a trace of the actions performed by the two agents is used to detect points where their behaviour diverged and these points can be used to start a dialogue. For convenience the first agent in the dialogue is referred to as the human, though it should be noted that our dialogues are in fact generated by two software agents conversing.

Figure 3 shows a sample dialogue generated by our system for Example 1. In this example when the robot reached the waypoint it perceived that the terrain was no longer safe and so did not move to the final location. The human asks why it did not make this move. The robot and human agree that it was at the waypoint, and that it had a goal to move to the final location, but they realise they disagree that the terrain was safe and the robot explains that it no longer believed the terrain to be safe from time point 15.

Figure 4 shows a dialogue for a different example. In this example the robot is charged with performing routine remote inspections of some site (e.g., a nuclear waste storage facility). When it performs its daily inspection it should inspect the walls of the facility (if they are scheduled for inspection) and the stored barrels (if they are stored for inspection). In this scenario, both tasks have equal precedence. This simple program is shown in Listing 3.

Code Listing 3

```
daily_inspection, barrels_scheduled -(2)>                    1
                    do(inspect_barrels), -barrels_scheduled  2
daily_inspection, wall_scheduled -(2)>                       3
                    do(inspect_wall), -wall_scheduled        4
```

[6] Source code can be found at https://github.com/jhudsy/BDIexplanation.git.

```
human: Why Not move2 at 17
robot: Why move2 at 17
human: Selected at_waypoint,goal_move_to_location,safe_terrain, -(2)>
          do(move2),-goal_move_to_location,-at_waypoint,+at_location,+goal_take_sample, at 16
robot: Why select at_waypoint,goal_move_to_location,safe_terrain, -(2)>
          do(move2),-goal_move_to_location,-at_waypoint,+at_location,+goal_take_sample, at 16
human: +at_waypoint at time 14 and it remained so until at least 16
robot: I agree +at_waypoint between 14 and 16
human: +goal_move_to_location at time 11 and it remained so until at least 16
robot: I agree +goal_move_to_location between 11 and 16
human: +safe_terrain at time 6 and it remained so until at least 16
robot: -safe_terrain at time 15 and it remained so until at least 16
human: Why -safe_terrain at 15
robot: I perceived -safe_terrain at 15
```

Fig. 3 Sample Dialogue for Example 1

```
human: Why Not inspect_wall at 14
robot: Why inspect_wall at 14
human: Selected wall_scheduled,daily_inspection, -(2)> do(inspect_wall), at 13
robot: Selected barrels_scheduled,daily_inspection, -(2)> do(inspect_barrels), at 13
human: wall_scheduled,daily_inspection, -(2)> do(inspect_wall), has precedence in my plan library
```

Fig. 4 Sample Dialogue for Plan Priority Example

Figure 4 shows a dialogue generated for an instance where the human believes that the wall inspection should have priority over barrel inspection. The human asks why the robot did not inspect the wall, the robot counters by asking why the human thought it should inspect the wall. They both explain the plan they thought applicable at that point and the human asserts that they thought the wall inspection plan had priority. Since both plans had equal priority, the robot is now aware of where the source of disagreement occurs. Communicating this to the human is outside the scope of the dialogue, but is easily done.

7 Dialogue strategies

Our current dialogue mechanism can produce dialogues with many seemingly redundant digressions (at least to human eyes). While the dialogues we show in Figs. 3 and 4 are compact these were selected from a large number of potential dialogues we generated and were selected specifically since they intuitively "make sense". However at many points in a dialogue a player has multiple options – for instance asking about any of the components of the guard on a plan, whether they agree or disagree with it. We therefore need to consider a strategy for the dialogues, and to this end, we must distinguish between *confirmatory dialogues* and *disagreement dialogues*. In the first case the participants are interested in establishing only that they have reasoned in the same way, while in the second case the participants are seeking to identify some cause of disagreement. Clearly, these different approaches require different *dialogue strategies* in which some set of heuristics may guide the choice of utterances during the dialogue in order to achieve the overarching dialogue goal more efficiently.

We focus here on defining a strategy for disagreement dialogues with a view to identifying at least one difference in the participants' plans or observations. Our strategy depends upon the assumption that some difference has been observed in the traces of the two participants. The dialogue opens when one participant asks either *why*(A, T)

Table 4 Additional constraints on why and why not questions introduced by the strategy

Utterance	Move condition	Additional strategy condition
$why(\neg A, T)$	$stage^{\mathcal{M}}[T] = \mathtt{p}$	$a^{\mathcal{M}}_{ex}[T] = A$
$why(A, T)$	$stage^{\mathcal{M}}[T] = \mathtt{p}$	$a^{\mathcal{M}}_{ex}[T] \neq A$
$why(\pi, T)$	$\pi = \pi^{\mathcal{O}}[T]$	$\pi \neq \pi^{\mathcal{M}}[T]$
$why(B, T)$	$B \in \mathcal{B}^{\mathcal{O}}[T]$	$B \notin \mathcal{B}^{\mathcal{M}}[T]$
$why(\neg B, T)$	$B \notin \mathcal{B}^{\mathcal{O}}[T]$	$B \in \mathcal{B}^{\mathcal{M}}[T]$

or $why(\neg A, T)$ (i.e., why, or why not, does some action appear in the trace at time T). Unlike the basic dialogue described in Tables 2 and 3, we now introduce the additional move condition that $a^{\mathcal{M}}_{ex}[T] = A$ (resp. $\neq A$) — i.e., if the participant asks $why(A, T)$ then it is because they expected to observe A (or resp. did not expect to observe A).

The strategy imposes extra conditions on several other dialogue moves – for instance $why(B, T)$ now requires that the agent asking this did not believe B at time T. These additional conditions are shown in Table 4. Closure conditions and effects remain the same. Given we have imposed additional constraints on the dialogue we need to show that it remains the case that if there is an open move, a response to the open move continues to be possible.

Proposition 5 *Given a dialogue using our disagreement strategy and an open move, a response to the open move that is consistent with the strategy is always possible.*

Proof We refer back to the proof of Proposition 2. We need only consider the cases from that proof where the possible response was one of the moves which have an additional condition under the strategy. These are:

$why(\neg A, T)$ In the proof of Proposition 2, $why(A, T)$ was needed to provide a possible response in the situation were action A was not in the second participant's trace at time T. In this situation $why(A, T)$ is also legal under the strategy.

$assert(\pi, T)$ In the proof of Proposition 2, $why(\pi, T)$ was needed to provided a possible response if the plan was in the second participant's library but could not be selected for execution at time T. In this situation $why(\pi, T)$ is also legal under the strategy.

$assert(B, T, T')$ In the proof of Proposition 2, $why(B, T)$ was needed to provide a possible response if the belief B did not hold at time T. In this situation $why(B, T)$ is also legal under the strategy.

$assert(\neg B, T, T')$ In the proof of Proposition 2, $why(\neg B, T)$ was needed to provide a possible response if B appeared in the second respondent's trace at time T. In this situation $why(\neg B, T)$ is also legal under the strategy. □

It is interesting to note that we do not need to concern ourselves with the applicability of $why(\neg A, T)$ in the above proof. This move only ever initiates a dialogue (see Table 2) and our strategy restricts this to only initiating disagreement dialogues rather than confirmatory dialogues.

The final component of the disagreement strategy involves a dialogue participant making a *why* move whenever it can according to the rules of the dialogue. If multiple

such moves are possible, one is selected at random. The effect of this is that dialogue participants will always query actions, plans or beliefs they disagree with, homing in on points where their traces do not align.

Unlike disagreement strategies, confirmatory strategies would potentially require the dialogue participants to validate their full traces; identifying heuristics for this class of strategies (for example based on opponent modelling) is left for future work.

8 Discussion and future work

We noted in Sect. 2 that the advantages of human-to-human explanatory dialogue over more static explanation techniques (such as, for example, written text or a one-shot utterance) include the ability to focus the explanation on the relevant portions of the domain; skip unnecessary detail; and make use of background information already known to the other party. Our proposed dialogue provides some of these advantages. More specifically, the selective use of *why* questions allows one dialogue participant to focus in only on those areas of discussion where they disagree with the other party, and ignore those portions where they believe an explanation is not needed (though, by leaving that branch of the dialogue open, they can return to it if necessary). However, our dialogue does not assume any domain specific knowledge held by other parties, and follows a very prescriptive structure, requiring an explanation to always start at an action before moving down to plans, beliefs, and then back to plans and/or percepts.

A dialogue participant can make multiple utterances in some stages of the dialogue. For example, a possible response to a $why(\pi, T)$ move, could be a single $assert(b_1, \_, T-1)$ move followed by a sub-dialogue to close this assertion, after which a second $assert(b_2, \_, T-2)$ move can be made followed by another sub-dialogue. Alternatively, a response consisting of multiple moves of the form $assert(b_1, \_, T-1), \ldots, assert(b_n, \_, T-1)$ could be made, closing the original *why* move, but leaving all the assertions open until dealt with. Rules covering turn taking (for example) would then instantiate specific dialogues, but this would not affect the dialogue properties described previously. In other words, our formalism describes a family of possible dialogues, and experiments investigating which of these is most suitable for providing explanation is left as an avenue for future work.

Our work makes an important assumption, namely that both dialogue participants apply the SimpleBDI semantics correctly to their internal version of the BDI program. In other words, the disagreements we identify come about from omissions or differences in the plan library, in the initial set of beliefs held by the dialogue participants, or differences in beliefs regarding the input trace τ_e. Extending the dialogue to deal with fallible participants who may simply forget a belief or who do not apply an applicable rule is another strand of future work, as doing so will provide for a dialogue more suited to humans acting as dialogue participants.

At worst, our dialogue identifies only a single disagreement between participants. We assume that between dialogues, participants update their beliefs about the program execution trace and therefore, on rerunning the dialogue would identify different disagreements. Determining how such belief updates should take place is outside the scope of the paper, but serves as another important avenue of future work. Related to this, allowing the participants to update their $\mathcal{M}$ models during the trace (with concomitant effects on $\mathcal{O}'$ and $\overline{\mathcal{O}'}$) would enable more disagreements to be discovered during single instance of the dialogue. Such work would require, at the very least, the addition of moves to retract beliefs [32].

An orthogonal direction of future work involves extending our approach to a multi-agent system, rather than considering an individual agent in isolation. The presence of an agent communication language (such as KQML or FIPA ACL) whose semantics are described in terms of impacts on agent beliefs, goals and the like means that our dialogue could be extended to include the effects of such communication events in a natural manner.

Explanation has become an important area of AI research. Much of the work in the domain focuses on the explainability of machine learning systems [1], but several recent papers consider explanation related to planning systems. For example, there has been work on argument based planning [23], and explaining automated planning [8, 18], with the latter extending an idea by Caminada *et al.* of using a proof dialogue [3] to explain the behaviour of an automated planner [7, 21]. We note that the latter works also consider how to translate formal utterances (as per our dialogue) into natural language, and believe that this will be an interesting and fruitful avenue of research.

Several other argumentation based approaches to explanation have been proposed in the literature (e.g., [9, 28, 36]). While these approaches could be adapted to explain the behaviour of BDI systems, we are unaware of such adaptations, which would — at least — require instantiating BDI specific concepts related to time, beliefs, goals, and the like as rules, which could then be combined into arguments, and over which explanation dialogues could then operate. It is worth noting that in the context of BDI specifically, several researchers have investigated how argumentation can be applied to drive the reasoning cycle. [20] makes use of abstract argumentation semantics to select goals within a BDI system, while [22] extends Jason [5] to perform reasoning using argumentation schemes [31]. The arguments generated by these and similar systems can potentially be used to drive explanation, as per the proof dialogue approach used by [7].

In contrast, our current work does not utilise an argumentation-theoretic semantics to underpin it. Instead, it could be viewed as a dialogue game built using argument schemes and critical questions [32] created for the BDI domain, in the tradition of work in informal logic [31] and practical reasoning [2].

Winikoff [34] and Hindriks [15] both consider providing explanations for BDI languages in the context of debugging. Hindriks' work was later expanded by Koeman *et al.* [16]. These systems all generate explanations using a formal semantics over a trace of program execution. Harbers [14] generates explanations for BDI systems using goal hierarchy paired with a behaviour log. Winikoff *et al.* [35] uses a concept of preferences to help produce explanations from BDI program execution traces. While all of these systems — like us — use execution traces to provide explanations for BDI program, none compare conflicting traces through dialogue to guide the generation of the explanation towards the concerns of the user.

Sreedharan *et al.* [29] consider the question of explanations in the context of AI Planning and, like us, explicitly identify the need to reconcile the human mental model with execution to generate an explanation. They pre-generate a set of explanations that are intended to reveal specific aspects of the Planning system's model (for instance that a particular location must be visited in particular circumstances) and then use machine learning to determine which explanations are most likely to explain which observable transitions in the system behaviour. These are then presented to users when they label some particular transition as inexplicable. More recent work [30] examines how plans can be explained through abstraction (i.e., by ignoring some elements of an environmental state). Applying this idea of abstraction to our work may result in a useful dialogue strategy, and we intend to investigate this possibility as future work. In the context of planning, [17] describes how hypothetical plans can be generated which can be compared to the original plan, serving

a similar function to our two dialogue participants. However, these approaches ignore the dialogical aspects of our solution and are grounded in planning, reducing the importance of concepts such as percepts.

Apart from the research mentioned above, there are several additional strands of future work we intend to explore. Perhaps the most obvious involves a detailed empirical evaluation of the dialogue across multiple domains, and with users having different levels of expertise. Based on the results noted by [21, 36], we believe that our system will be shown to have a positive effect on user understanding of system behaviour. Another direction of research involves investigating the link between dialogue and formal argumentation (noted by Caminada *et al.* [7]). Move sequences such as $assert(\pi, T)$, $assert(\pi', T)$ imply a contradiction in the dialogue participant's views which — through the dialogue — are instantiated into attacking arguments. We therefore intend to investigate an argument-theoretic semantics for the dialogue presented in this paper, potentially allowing for stronger links with other explainable AI approaches underpinned by argumentation [3, 10, 21], aiming to enable more efficient dialogues through the introduction of concepts such as burden of proof [24]. We also intend to further investigate the effects of strategy on dialogue properties. While our results provide worst-case upper bounds for dialogue length, and propose a simple strategy for disagreement dialogues, more complex strategies which consider what the other participant's knowledge and goals are [25] may — at least in the average case — significantly reduce the number of moves that need to be made. Finally, extending SimpleBDI may result in more complex dialogues. Allowing, for example, a non-strict ordering over plans could allow participants to argue about the unobserved effects of plans, requiring looking forwards as backwards over time, and such enrichment could be a fruitful direction of future work.

9 Conclusions

We presented a family of dialogues allowing two dialogue participants to identify if, and where, a divergence of views exists between them with regards to a BDI agent's operation. Our dialogue aims to be general enough to capture two external observers discussing the behaviour of a (third) BDI agent, but we believe that in practice, one of the dialogue participants will be the BDI agent, seeking to explain its actions to the second participant, typically a human. Such explanations then focus on divergences in the views of the participants with regards to the perceptions, plans and underlying beliefs of the BDI system, and we show that when a divergence exists with regards to what action should have taken place, the dialogue enables the root cause of the divergence to be detected.

Acknowledgements This work arose out of conversations at a Lorentz Workshop on the Dynamics of Multi-Agent Systems (2018). Thanks are due Koen Hindriks and Vincent Koeman for their input. The work was supported by the UKRI/EPSRC RAIN [EP/R026084], SSPEDI [EP/P011829/1] and FAIR-SPACE [EP/R026092] Robotics and AI Hubs and the Trustworthy Autonomous Systems Verifiability Node [EP/V026801/1].

Author contribution Both authors contributed equally to the work, and author names are listed in alphabetical order.

References

1. Adadi, A., & Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access, 6*, 52138–52160.
2. Atkinson, K., & Bench-Capon, T. (2007). Practical reasoning as presumptive argumentation using action based alternating transition systems. *Artificial Intelligence, 171*(10), 855–874. https://doi.org/10.1016/j.artint.2007.04.009.
3. Baroni, P., Gabbay, D., Giacomin, M., & van der Torre, L. (2018). Handbook of Formal Argumentation. College Publications
4. Belle, V., & Papantonis, I. (2021). Principles and practice of explainable machine learning. *Frontiers in Big Data, 4,* 39.
5. Bordini, R., Hübner, J., & Wooldridge, M. (2007). *Programming multi-agent systems in AgentSpeak Using Jason*. UK: Wiley.
6. Bordini, R. H., Hübner, J. F., & Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology)*. Hoboken, NJ, USA: John Wiley and Sons Inc.
7. Caminada, M.W., Kutlak, R., Oren, N., & Vasconcelos, W.W. (2014). Scrutable plan enactment via argumentation and natural language generation. In: Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems, pp. 1625–1626. International Foundation for Autonomous Agents and Multiagent Systems (2014)
8. Collins, A., Magazzeni, D., & Parsons, S. (2019). Towards an argumentation-based approach to explainable planning. In: Proceedings of the 2nd ICAPS Workshop on Explainable Planning (XAIP-2019), p. 5
9. Cyras, K., Fan, X., Schulz, C., & Toni, F. (2017) Assumption-based argumentation: Disputes, explanations, preferences. FLAP **4**(8) . http://www.collegepublications.co.uk/downloads/ifcolog00017.pdf
10. Čyras, K., Letsios, D., Misener, R., & Toni, F. (2019). Argumentation for explainable scheduling. In: Proceedings of the AAAI Conference on Artificial Intelligence, 33: 2752–2759
11. Dastani, M. (2008). 2apl: A practical agent programming language. *Autonomous Agents and Multi-Agent Systems, 16*(3), 214–248. https://doi.org/10.1007/s10458-008-9036-y
12. Dennis, L., Fisher, M., Webster, M., & Bordini, R. (2012). Model Checking Agent Programming Languages. *Automated Software Engineering, 19*(1), 5–63.
13. Garcia, E., Tyson, G., Miles, S., Luck, M., Taweel, A., Van Staa, T., & Delaney, B. (2013). Analysing the Suitability of Multiagent Methodologies for e-Health Systems. In J. P. Müller & M. Cossentino (Eds.), *Agent-Oriented Software Engineering XIII* (pp. 134–150). Berlin Heidelberg, Berlin, Heidelberg: Springer.
14. Harbers, M. (2011). Explaining agent behaviour in virtual training. Ph.D. thesis, SIKS Dissertation Series . No. 2011-35
15. Hindriks, K. V. (2012). Debugging is explaining. In I. Rahwan, W. Wobcke, S. Sen, & T. Sugawara (Eds.), *PRIMA 2012: Principles and Practice of Multi-Agent Systems* (pp. 31–45). Berlin Heidelberg, Berlin, Heidelberg: Springer.
16. Koeman, V., Dennis, L.A., Webster, M., Fisher, M., & Hindriks, K. (2019) The "Why did you do that?" Button: Answering Why-questions for end users of Robotic Systems. In: Proceedings of the 7th International Workshop in Engineering Multi-Agent Systems. Montreal, Canada . http://cgi.csc.liv.ac.uk/~lad/emas2019/accepted/EMAS2019_paper_27.pdf
17. Krarup, B., Cashmore, M., Magazzeni, D., & Miller, T. (2019) Model-based contrastive explanations for explainable planning. In: ICAPS 2019 Workshop on Explainable AI Planning (XAIP)
18. Mahesar, Q., & Parsons, S. (2021) Argument schemes and dialogue for explainable planning. CoRR **abs/2101.02648** . arxiv:2101.02648
19. Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence, 267*, 1–38. https://doi.org/10.1016/j.artint.2018.07.007

20. Morveli Espinoza, M., Possebom, A.T., & Tacla, C.A. (2019). Argumentation-Based Agents that Explain Their Decisions. In: Proceedings of the 8th Brazilian Conference on Intelligent Systems (BRACIS), pp. 467–472. IEEE, Salvador, Brazil . https://doi.org/10.1109/BRACIS.2019.00088
21. Oren, N., van Deemter, K., & Vasconcelos, W.W. (2020). Argument-Based Plan Explanation, pp. 173–188. Springer International Publishing, Cham . https://doi.org/10.1007/978-3-030-38561-3_9.
22. Panisson, A.R., Engelmann, D.C., & Bordini, R.H. (2021). Engineering explainable agents: An argumentation-based appraoch. In: Proceedings of the 9th International Workshop on Engineering Multi-Agent Systems
23. Pardo, P., & Godo, L. (2018). A temporal argumentation approach to cooperative planning using dialogues. *Journal of Logic and Computation, 28*(3), 551–580.
24. Prakken, H., Reed, C., & Walton, D. (2005). Dialogues about the burden of proof. In: Proceedings of the 10th International Conference on Artificial Intelligence and Law, ICAIL '05, pp. 115–124. ACM, New York, NY, USA . https://doi.org/10.1145/1165485.1165503.
25. Rienstra, T., Thimm, M., & Oren, N. (2013). Opponent models with uncertainty for strategic argumentation. In: Twenty-Third International Joint Conference on Artificial Intelligence
26. Samek, W., Montavon, G., Vedaldi, A., Hansen, L.K., & Müller, K. (eds.): (2019).Explainable AI: Interpreting, Explaining and Visualizing Deep Learning, *Lecture Notes in Computer Science*, 11700. Springer
27. Shin, D. D. (2021). The effects of explainability and causability on perception, trust, and acceptance: Implications for explainable AI. *International Journal of Human-Computer Studies, 146,* 102551.
28. Sklar, E.I., & Azhar, M.Q. (2018). Explanation through argumentation. In: Proceedings of the 6th International Conference on Human-Agent Interaction, HAI '18, p. 277-285. Association for Computing Machinery, New York, NY, USA . https://doi.org/10.1145/3284432.3284470.
29. Sreedharan, S., Olmo, A., Mishra, A.P., & Kambhampati, S. (2019) Model-free model reconciliation. In: IJCAI
30. Sreedharan, S., Srivastava, S., & Kambhampati, S. (2021). Using state abstractions to compute personalized contrastive explanations for AI agent behavior. *Artificial Intelligence, 301,* 103570.
31. Walton, D. (2008). Informal Logic: A Pragmatic Approach, 2 edn. Cambridge University Press . https://doi.org/10.1017/CBO9780511808630
32. Walton, D., & Krabbe, E.C. (1995). Commitment in dialogue: Basic concepts of interpersonal reasoning. SUNY press
33. Weigang, L., de Souza, B. B., Crespo, A. M. F., & Alves, D. P. (2008). Decision support system in tactical air traffic flow management for air traffic flow controllers. *Journal of Air Transport Management, 14*(6), 329–336.
34. Winikoff, M. (2017). Debugging agent programs with Why? questions. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS '17, pp. 251–259. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC
35. Winikoff, M., Dignum, V., & Dignum, F. (2016). Why bad coffee? explaining agent plans with valuings. In: A. Skavhaug, J. Guiochet, E. Schoitsch, F. Bitsch (eds.) SAFECOMP, *LNCS*, vol. 9923, pp. 521–534. Springer
36. Čyras, K., Rago, A., Albini, E., Baroni, P., & Toni, F. (2021). Argumentative xai: A survey. In: Z.H. Zhou (ed.) Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, pp. 4392–4399. International Joint Conferences on Artificial Intelligence Organization . https://doi.org/10.24963/ijcai.2021/600. Survey Track