Resolving Misconceptions about the Plans of Agents via Theory of Mind (Appendix)

To provide structure for this document, we enumerate the different appendices below:

- **Appendix A:** we provide background on PEKBs and RMLs.
- **Appendix B:** we provide background on classical planning.
- **Appendix C:** we provide a proof of Theorem 1.
- **Appendix D:** we describe in detail the various domains used in our evaluation.
- **Appendix E:** we detail some of the domain and problem files used in the evaluation described in Section 5. We moreover provide a number of examples of the goals given to the classical planner and the generated discrepancy resolving plans.
- **Appendix F:** we provide details about the method and results of our user study.

A Proper Epistemic Knowledge Bases (PEKBs)

A proper epistemic knowledge base (PEKB) is a set of restricted formulae, called restricted modal literals (RMLs) (Lakemeyer and Lespérance 2012). An RML is obtained from the following grammar:

$$\phi ::= p \mid B_i\phi \mid \neg\phi$$

where $p \in \mathcal{P}$ and $i \in Ag$. $\mathcal{P}$ and Ag are sets of atoms and agents, respectively, as defined in Section 2. The maximum number of nested belief modalities determines the depth of an RML. For instance, $B_i\phi$ has a depth of 1 in addition to the depth of ϕ, where ϕ may hold additional belief modalities. Formally, the depth of an RML is defined by Muise et al. (Muise et al. 2015b) as: $depth(p) = 0$ for $p \in P$, $depth(\neg\phi) = depth(\phi)$ and $depth(B_i\phi) = 1 + depth(\phi)$. In addition, a conjunction of RMLs is defined as a set of RMLs, where the set of all RMLs with bounded depth d for a group of agents Ag is denoted as $\mathcal{L}^{Ag,d}_{RML}$. The state of the world is represented by some set of RMLs.

B Classical Planning

Automated planning is the task of selecting a goal leading plan based on a high-level description of the world (Ghallab, Nau, and Traverso 2004). While there are many flavors of automated planning, *classical planning* involves deterministic actions and a complete and fully observable environment. Here we are interested in classical$^+$ planning problems which are classical planning problems augmented with ADL (Pednault 1989) features, most notably here including conditional effects and a disjunctive goal. We adapt Muise et al.'s definition of a classical planning problem (Muise et al. 2021), which includes conditional effects, to also include disjunctive goals.

Definition 10 (Classical$^+$ Planning Problem (building on (Muise et al. 2021))) *A **classical$^+$ planning problem** consists of a tuple $\langle \mathcal{F}, \mathcal{I}', G', O \rangle$ where $\mathcal{F}$ is a set of fluent atoms, $\mathcal{I}$ is the initial state, G is the goal, and O is a set of operators. A complete state s is a subset of the fluents $\mathcal{F}$ with the interpretation that fluents not in s are false (equivalently, this can be seen as a conjunction between the literals found in s and the negated literals not found in s). A partial state s is similarly a subset of the fluents $\mathcal{F}$, but has the interpretation that literals not found in s can take on any value (thus equivalent to a conjunction of just the literals in s). $\mathcal{I}$ is a complete state while G is a disjunction of partial states (possibly a single disjunct, in which case a goal simply a partial state, as in a 'regular' classical planning problem). Every operator $o \in O$ is a tuple $\langle Pre_o, \mathit{eff}^+_o, \mathit{eff}^-_o \rangle$, and we say that o is applicable in the state s iff $Pre_o \subseteq s$. The set eff^+_o (resp. eff^-_o) contains conditional effects describing the fluent atoms that should be added (resp. removed) from the state when applying the operator. Finally, every conditional effect in eff eff^+_o or eff^-_o is of the form $(C \rightarrow l)$ where C is the condition for the effect and l is a fluent that is the result of the effect. The condition C consists of a tuple $\langle C^+, C^- \rangle$ where C^+ is the set of fluents that must hold and C^- the set of fluents that must not hold. A conditional effect $(\langle C^+, C^- \rangle \rightarrow l)$ fires in state s if $C^+ \subseteq s$ and $C^- \cap s = \emptyset$. Assuming o is applicable in s, and $\mathit{eff}^+_o(s)$ (respectively, $\mathit{eff}^-_o(s)$) are the positive (resp. negative) conditional effects that fire in state s, the state of the world s' after applying o is defined as follows:*

$$s' = s \setminus \{l \mid (C \rightarrow l) \in \mathit{eff}^-_o(s)\} \cup \{l \mid (C \rightarrow l) \in \mathit{eff}^+_o(s)\}$$

Given a classical$^+$ planning problem $\langle \mathcal{F}, \mathcal{I}', G', O \rangle$, a solution is a sequence of actions (a plan) $\pi = a_1 \dots a_n$, $a_i \in O$, where π is applicable in $\mathcal{I}'$ (i.e., a_1 is applicable in $\mathcal{I}'$, a_2 is applicable in the state of the world after applying a_1 and so on) and there exists a disjunct G_i of G (where G_i is a partial state) such that $G_i \subseteq s_\pi$ where s_π is the state of the world after sequentially applying the operators in π. We use $\textsc{prog}_c(\pi, \mathcal{I}')$ to denote the state of the world after sequentially applying the operators in π. $\textsc{prog}_c(\pi, \mathcal{I}')$ is not defined if π is not applicable in $\mathcal{I}'$.

C Proof of Theorem 1

To prove Theorem 1, we first formulate a number of definitions and lemmas. We start by defining (following (Muise et al. 2021)) a restricted perspectival multi-agent epistemic planning problem (RP-MEP problem) for depth bound d and the root agent $\star \in Ag$ as a MEP problem with the additional restriction that every RML is from the perspective of the root agent – i.e., it is from the following set:

$$\{B_\star \phi \mid \phi \in \mathcal{L}_{RML}^{Ag,d}\}$$

Next, we define the classical encoding of an RP-MEP problem (following (Muise et al. 2021)).

Definition 11 (Classical Encoding of an RP-MEP Problem (following (Muise et al. 2021))) *Let $\mathcal{B}_i$ and $\mathcal{N}_i$ be functions that map agent i's positive and, respectively, negative beliefs from a PEKB P to the respective fluents in the classical$^+$ planning domain:*

$\mathcal{B}_i(P) = \{l_\phi \mid B_i\phi \in P\}$
$\mathcal{N}_i(P) = \{\neg l_\phi \mid \neg B_i\phi \in P\}$

Given an RP-MEP problem, $\langle\langle \mathcal{P}, \mathcal{A}, Ag\rangle, \mathcal{I}, G\rangle$, we define the propositional encoding as the tuple $\langle \mathcal{F}, \mathcal{I}', G', O\rangle$ such that:

$\mathcal{F} \overset{def}{=} \{l_\phi \mid \phi \in \mathcal{L}_{RML}^{Ag,d}\}$
$\mathcal{I}' \overset{def}{=} \mathcal{B}_\star(\textsc{Closure}(\mathcal{I}))$
$G' \overset{def}{=} \mathcal{B}_\star(G)$

where $\star \in Ag$ is the root agent. Planning in RP-MEP is done from the perspective of the root agent. Closure *is a closure procedure that deduces a new set of RMLs from an existing one under the KD_n axioms (Muise et al. 2021, Definition 7). For every action $\langle Pre_a, \langle \mathit{eff}_a^+, \mathit{eff}_a^-\rangle\rangle$ in $\mathcal{A}$, we have a corresponding operator $\langle Pre_o, \langle \mathit{eff}_o^+, \mathit{eff}_o^-\rangle\rangle$ in O such that:*

$Pre_o \overset{def}{=} \mathcal{B}_\star(Pre_a)$
$\mathit{eff}_o^+ \overset{def}{=} \{(\langle \mathcal{B}_\star(\gamma_i), \overline{\mathcal{N}_\star(\gamma_i)}\rangle \rightarrow l_\phi) \mid (\gamma_i, \mathcal{B}_\star\phi) \in \mathit{eff}_a^+\}$
$\mathit{eff}_o^- \overset{def}{=} \{(\langle \mathcal{B}_\star(\gamma_i), \overline{\mathcal{N}_\star(\gamma_i)}\rangle \rightarrow l_\phi) \mid (\gamma_i, \neg\mathcal{B}_\star\phi) \in \mathit{eff}_a^-\}$

Definition 12 (Classical encoding function) *Given an RP-MEP problem $\langle\langle \mathcal{P}, \mathcal{A}, Ag\rangle, \mathcal{I}, G\rangle$, a root agent $\star \in Ag$, the corresponding set of all RMLs with bounded depth d, $\mathcal{L}_{RML}^{Ag,d}$, and an RML $\phi \in \mathcal{L}_{RML}^{Ag,d}$, we say that $\mathcal{C}(\phi) = \mathcal{B}_\star\phi$*

Definition 13 (Classical decoding function) *Given an RP-MEP problem $\langle\langle \mathcal{P}, \mathcal{A}, Ag\rangle, \mathcal{I}, G\rangle$, a root agent $\star \in Ag$, the corresponding classical$^+$ planning problem per Definition 11, $\langle \mathcal{F}, \mathcal{I}', G', O\rangle$, the corresponding set of all RMLs with bounded depth d, $\mathcal{L}_{RML}^{Ag,d}$, and a propositional fluent $l_\phi \in \mathcal{F}$, we say that $\mathcal{D}(l_\phi) = \phi$, $\phi \in \mathcal{L}_{RML}^{Ag,d}$.*

Lemma 1 *Suppose $Q = \langle \mathcal{P}, \mathcal{A}, Ag\rangle$ is an RP-MEP domain, $\star \in Ag$ is the root agent, $\mathcal{I}$ an initial state, and the classical encoding of those (according to* ClassicallyEncodeMEP*) is $\langle \mathcal{F}, \mathcal{I}', O\rangle$, where $\mathcal{F}$ is a set of propositional fluents representing each RML in the domain Q, $\mathcal{I}'$ is the classically encoded initial state $\mathcal{I}$, O is a set of classically encoded operators corresponding to the set of actions $\mathcal{A}$ in Q, and $\mathcal{L}_{RML}^{Ag,d}$ is the corresponding set of all RMLs with bounded depth d. Then for every RML $\ell \in \mathcal{L}_{RML}^{Ag,d}$ of the form $B_\star\phi$ and sequence of actions π from $\mathcal{A}$,*

$$\textsc{prog}(\pi, \mathcal{I}) \models \ell$$

if and only if

$$\textsc{prog}_c(\pi', \mathcal{I}') \models \mathcal{C}(\ell)$$

where π' is the sequence of actions in O corresponding to π.

Proof 1 *This follows from (Muise et al. 2021, Theorem 2) and Definition 12.*

Lemma 2 *Suppose $Q = \langle \mathcal{P}, \mathcal{A}, Ag\rangle$ is an RP-MEP domain, $\star \in Ag$ is the root agent, $\mathcal{I}$ an initial state, and the classical encoding of those (according to* ClassicallyEncodeMEP*) is $\langle \mathcal{F}, \mathcal{I}', O\rangle$, where $\mathcal{F}$ is a set of propositional fluents representing each RML in the domain Q, $\mathcal{I}'$ is the classically encoded initial state $\mathcal{I}$, O is a set of classically encoded operators corresponding to the set of actions $\mathcal{A}$ in Q, and $\mathcal{L}_{RML}^{Ag,d}$ is the corresponding set of all RMLs with bounded depth d. For every RML $\phi \in \mathcal{L}_{RML}^{Ag,d}$ of the form $B_\star\ell$ where $\mathcal{C}(B_\star\ell) = \ell'$, $\ell' \in \mathcal{F}$, and sequence of actions π from $\mathcal{A}$,*

$$\textsc{prog}(\pi, \mathcal{I}) \models B_\star\mathcal{D}(\ell')$$

if and only if

$$\text{PROG}_c(\pi', \mathcal{I}') \models \ell'$$

where π' is the sequence of actions in O corresponding to π.

Proof 2 *This follows from (Muise et al. 2021, Theorem 2) and Definitions 12 and 13.*

We can extend $\mathcal{D}()$ to apply it to not just fluents, but any boolean combination of fluents:

$$\begin{aligned}\mathcal{D}(\phi \vee \psi) &= (\mathcal{D}(\phi) \vee \mathcal{D}(\psi))\\ \mathcal{D}(\phi \wedge \psi) &= (\mathcal{D}(\phi) \wedge \mathcal{D}(\psi))\\ \mathcal{D}(\neg\phi) &= \neg\mathcal{D}(\phi)\end{aligned}$$

Definition 14 *Given a PEKB S, the classical encoding S' of S is*

$$S' = \textsc{Closure}(\mathcal{B}_\star(S) \cup \mathcal{N}_\star(S))$$

In our implementation, the discrepancy-resolving agent, agent i, is always the root agent. Since we wish to talk about the root agent's beliefs about other agents' beliefs about $\text{VAL}(\pi, G)$, we define $\text{VAL}(\pi, G)$ for the RP-MEP setting.

Definition 15 *Given an RP-MEP problem $\langle\langle\mathcal{P}, \mathcal{A}, Ag\rangle, \mathcal{I}, G\rangle$ (where $\star \in Ag$ is the root agent) and plan π, we define*

$$\text{VAL}(\pi, G) = \mathcal{D}(\text{VAL}_c(\pi_c, G_c))$$

That is,

$$\text{VAL}(\pi, G) = \mathcal{D}(\text{REG}_c(\pi_c, G_c)),$$

where VAL_c is the validity formula in a classical$^+$ planning setting.

That is, $\text{VAL}(\pi, G)$ is a formula with the same structure as $\text{VAL}_c(\pi_c, G_c)$, but which replaces the propositional fluents in it with the corresponding RMLs in the MEP domain. Recall that $\text{VAL}_c(\pi_c, G_c)$ is the classical$^+$ planning validity formula, which ϕ is set to in Line 4 of Algorithm 1.

Theorem 1 *Suppose that a plan π' is returned by Algorithm 1, given a tuple $R = \langle\langle\mathcal{P}, \mathcal{A}, Ag\rangle, \mathcal{I}, i, j, \vec{v}, \pi, G\rangle$. Then π'' is a discrepancy resolving plan for R, where π'' is the plan comprising actions from $\mathcal{A}$ corresponding to the classically encoded operators in π'.*

Proof 3 *We want to show that if a plan π' is returned by* ResolveDiscrepancy *given the tuple $\langle\langle\mathcal{P}, \mathcal{A}, Ag\rangle, \mathcal{I}, i, j, \vec{v}, \pi, G\rangle$, where $\langle\langle\mathcal{P}, \mathcal{A}, Ag\rangle, \mathcal{I}, G\rangle$ is an RP-MEP problem (where $\star \in Ag$ is the root agent), then the plan π'' corresponding to the plan π' is a discrepancy resolving plan for $\langle Q, \mathcal{I}, i, j, \vec{v}, \pi, G\rangle$. The plan π' is returned in Line 6 by the classical planner and therefore solves $\langle\mathcal{F}, \mathcal{I}', G', O\rangle$, where G' is*

$$\bigvee_{\phi_d \in DNF(\phi)} \left(\bigwedge_{\phi_{dc} \in \phi_d} \mathcal{C}(B_{i,j,\vec{v}}\mathcal{D}(\phi_{dc})) \wedge \mathcal{C}(B_{i,\vec{v}}\mathcal{D}(\phi_{dc})) \right) \vee$$
$$\bigvee_{\phi_d \in DNF(\neg\phi)} \left(\bigwedge_{\phi_{dc} \in \phi_d} \mathcal{C}(B_{i,j,\vec{v}}\mathcal{D}(\phi_{dc})) \wedge \mathcal{C}(B_{i,\vec{v}}\mathcal{D}(\phi_{dc})) \right),$$

where $\phi = \text{VAL}_c(\pi_c, G_c)$. and π_c and G_c correspond to the plan and goal π and G in the tuple given to Algorithm 1. π_c contains operators from O, and G_c is expressed using propositional fluents from $\mathcal{F}$. By Lemma 2, we have that

$$\text{PROG}(\pi'', \mathcal{I}) \models \bigvee_{\phi_d \in DNF(\phi)} \left(\bigwedge_{\phi_{dc} \in \phi_d} B_{i,j,\vec{v}}\mathcal{D}(\phi_{dc}) \wedge B_{i,\vec{v}}\mathcal{D}(\phi_{dc}) \right) \vee$$
$$\bigvee_{\phi_d \in DNF(\neg\phi)} \left(\bigwedge_{\phi_{dc} \in \phi_d} B_{i,j,\vec{v}}\mathcal{D}(\phi_{dc}) \wedge B_{i,\vec{v}}\mathcal{D}(\phi_{dc}) \right).$$

It follows that

$$\textsc{prog}(\pi'', \mathcal{I}) \models [B_{i,j,\vec{v}}\mathcal{D}(\phi) \wedge B_{i,\vec{v}}\mathcal{D}(\phi)] \vee$$
$$[B_{i,j,\vec{v}}\mathcal{D}(\neg\phi) \wedge B_{i,\vec{v}}\mathcal{D}(\neg\phi)].$$

Since $\phi = \textsc{val}_c(\pi_c, G_c)$*, that can be rewritten as*

$$\textsc{prog}(\pi'', \mathcal{I}) \models [B_{i,j,\vec{v}}\mathcal{D}(\textsc{val}_c(\pi_c, G_c)) \wedge B_{i,\vec{v}}\mathcal{D}(\textsc{val}_c(\pi_c, G_c))] \vee$$
$$[B_{i,j,\vec{v}}\mathcal{D}(\neg\textsc{val}_c(\pi_c, G_c)) \wedge B_{i,\vec{v}}\mathcal{D}(\neg\textsc{val}_c(\pi_c, G_c))]$$

Using the definition of $\mathcal{D}()$*, we can move some negation signs around:*

$$\textsc{prog}(\pi'', \mathcal{I}) \models [B_{i,j,\vec{v}}\mathcal{D}(\textsc{val}_c(\pi_c, G_c)) \wedge B_{i,\vec{v}}\mathcal{D}(\textsc{val}_c(\pi_c, G_c))] \vee$$
$$[B_{i,j,\vec{v}}\neg\mathcal{D}(\textsc{val}_c(\pi_c, G_c)) \wedge B_{i,\vec{v}}\neg\mathcal{D}(\textsc{val}_c(\pi_c, G_c))]$$

Since we defined $\textsc{val}(\pi, G) = \mathcal{D}(\textsc{val}_c(\pi_c, G_c))$*, we are done.*

D Domain Descriptions

In what follows, we describe the various domains (and problems within those domains) used in our experiments.

D.1 BW4T

Johnson et al. (2009) presented a multi-agent simulation platform, BlocksWorld for Teams (BW4T), which is an abstraction of a myriad of application domains such as search & rescue. Typically in this domain, there are a number of rooms and a drop zone, where each room contains a number of colored blocks. In the application domains, blocks may represent survivors of a disaster or medical kits, and the various agents may be humans or robots with different roles and capabilities. We cast blocks as medical kits.

We modelled various instances of the BW4T domain by varying the number of rooms, medical kits, and types of medical kits, totalling 10 unique problem instances. Moreover, in each instance we modelled a number of scenarios involving perceived discrepancies about plan validity. Common to all scenarios and instances is the following: there are three agents in the environment (Alice, Mary, and Bob); all discrepancies are perceived by Alice and resolved by her; and all plans (except for the discrepancy resolving plans) are executed by Bob. In all scenarios, Alice can (truthfully) inform other agents of either (agents' beliefs about) the whereabouts of various medical kits or the status of Mary's communication device. Moreover, Alice can move medical kits between different locations in the environment. Our BW4T domain was adapted from the BW4T domain in the RP-MEP repository found in https://bit.ly/3lWjJ3b by reducing the number of actions found in the original domain. The domain is shown in more detail in Appendix E.2.

Dude, Where's my Medical Kit? This scenario (and the next) builds on our running example. Bob's goal is to get a particular medical kit to the drop zone and he believes that it is in some room. Alice believes that Bob holds a false belief pertaining to the location of the medical kit. Two tuples are created and given to Algorithm 1: one with π_{AliceBob} from our example in Section 3 (which Bob believes to be valid and Alice believes to not be valid, based on her belief about the location of the medical kit) and one with $\pi_{\text{AliceMaryBob}}$ (which Bob believes to not be valid and Alice believes to be valid). $\vec{v}$ is empty in these tuples. Possible discrepancy resolving plans that may be computed by the planner are π' and π'', as discussed in Section 3.

Where Does he Think he's Going? In this scenario, the setup is the same but Mary is involved and Alice believes that Mary falsely believes that Bob does not have a false belief about the location of the medical kit (similarly to our running example). Two tuples are created and given to Algorithm 1: one with π_{AliceBob} and one with $\pi_{\text{AliceMaryBob}}$ (about both of which there is a discrepancy between Alice's beliefs about Mary's beliefs about Bob's beliefs and Alice's beliefs about Bob's beliefs). $\vec{v}$ is $\langle$Bob$\rangle$ in both tuples. A possible discrepancy resolving plan that may be computed by the planner for π_{AliceBob} is π''', as discussed in Section 3.

To evaluate the impact of d, we also create a variant of this scenario with $d = 5$. That is, in addition to Mary, Alice, and Bob, we have two additional agents: Charlie and Rose. In this case, Alice believes that Mary falsely believes that Charlie believes that Rose believes that Bob does not have a false belief about the location of the medical kit. Thus, Alice must resolve a discrepancy between

$$B_{\text{Alice}}B_{\text{Mary}}B_{\text{Charlie}}B_{\text{Rose}}B_{\text{Bob}}\textsc{val}(\pi, G)$$

and

$$B_{\text{Alice}}B_{\text{Charlie}}B_{\text{Rose}}B_{\text{Bob}}\textsc{val}(\pi, G).$$

$\vec{v}$ is $\langle$Charlie, Rose, Bob$\rangle$ in the tuple given to Algorithm 1.

Can You Hear Me?? Here, Bob has the goal of getting a particular medical kit to the drop zone and notifying his teammate, Mary, that he has done so (i.e., at(Kit1,Hall) $\wedge$ B_{Mary}at(Kit1,Hall)). This time, Bob has a correct belief about the medical kit's location. However, while Alice believes that Mary's communication device is not working properly (perhaps she met Mary in passing and was told by her), she also believes that Bob falsely believes that it is working properly. Alice therefore believes that Bob's plan (which involves sending a message to Mary) will fail to achieve the *epistemic* component of his goal (i.e., for Mary to believe that a medical kit is now at the drop zone). A tuple is created and given to Algorithm 1 where $\vec{v}$ is empty and the plan π is

$$\begin{aligned}[&\textit{move}(\text{Bob,Hall,RoomB}),\\ &\textit{pckUp}(\text{Bob,Kit1,RoomB}),\\ &\textit{move}(\text{Bob,RoomB,Hall}),\\ &\textit{dropOff}(\text{Bob,Kit1,Hall}),\\ &\textit{sendComm}(\text{Bob,Mary,at(Kit1,Hall)}))].\end{aligned}$$

A possible discrepancy resolving plan involves Alice informing Bob that Mary's communication device is not working (see Appendix E.2 for details).

D.2 Corridor (Epistemic Planning Benchmark)

In our modified[6] version of the *Corridor* domain, there are n agents in various rooms connected to a long corridor. A single acting agent, Bob, holds a secret and can move along the corridor, enter different rooms, and announce his secret. When announcing the secret, all agents in the room with the announcer, as well as all agents in the adjacent rooms (when the door between the rooms is open), now believe the secret. The encoding of the *shareSecret* action in PDKBDDL appears in Appendix E.1.

Bob may have different epistemic goals, including a universal or selective spread of his secret to the other agents in the environment. For example, let us assume that there are two agents in the environment in addition to Bob (k and l) and that one of Bob's epistemic goals is B_k(BobSecret) $\wedge$ $\neg B_l$(BobSecret). That is, Bob wants agent k to believe his secret, but does not wish for agent l to believe it. We create 10 instances of this domain by varying the number of rooms, agents, and false beliefs held by Bob about the locations of each agent, and whether or not the doors between the different rooms are open or closed. For each of the generated instances, a tuple is created and given to Algorithm 1 where $\vec{v}$ is empty, agent i is Alice (who is an agent in the environment who holds correct beliefs about agent locations and about Bob's beliefs) and agent j is Bob. Alice can inform Bob of agents' locations and can also open and close doors in the environment.

As usual, we are interested in discrepancies pertaining to the validity of Bob's plan, as perceived by Alice. There are two reasons for Alice to believe that Bob's plan is not valid while believing that Bob believes it is valid:

- Bob's goal is for agent k to believe his secret but Alice believes that Bob falsely believes that k is in some room r (i.e., Alice believes that agent i is not in room r) and will plan to head to room r to share his secret with agent k. In some problem instances Bob's goal is for a number of agents to believe his secret (e.g., B_k(BobSecret) $\wedge$ B_l(BobSecret) $\wedge$ B_k(BobSecret)). In these problem instances, Alice may believe that Bob holds a false belief about the location of some or all of the agents.
- Bob only wants agent k to believe his secret without agent l believing it (i.e., B_k(BobSecret) $\wedge$ $\neg B_l$(BobSecret)). Alice believes that Bob falsely believes that agent l is neither in the room with agent k nor in the adjacent rooms or correctly believes that l is in an adjacent room but falsely believes that the door between the rooms is closed. Therefore, Bob will plan to go to the room in which he believes k to be and share his secret with her. However, this plan will fail to achieve his goal since either agent l is in the room with agent k; or agent l is in the adjacent room and will also come to believe Bob's secret since the door between the rooms is actually open.

As discussed in more detail in Appendix E.1, Alice can resolve discrepancies regarding the validity of Bob's plans by either (1) informing Bob of agents' locations or the (open or closed) position of doors, or (2) by closing or opening doors in the environment.

D.3 IPC Domains

Inspired by 7 IPC domains[7] – Depots, Driverlog, Gripper, Rovers, Logistics, Zeno Travel, and Satellite – we modelled 7 MEP domains with agents Alice, Bob, Mary, Charlie, and Rose. In total, we generated 70 problem instances (10 from each domain) by creating false beliefs for agents (e.g., causing an agent to hold a false belief about the location of an object in the Driverlog domain) and varying the domain parameters (e.g., number of objects in the domain). As before, Bob is the acting agent. For each problem instance, we varied the depth of nested belief d to create 3 cases where Alice (who perceives all discrepancies) resolves a discrepancy between her beliefs and the beliefs of other agents about the validity of Bob's plan. Appropriate tuples containing Bob's plan and goal were created and given to Algorithm 1.

[6]The unmodified corridor domain can be found in https://bit.ly/3tXsi0r.

[7]Which can be found in http://editor.planning.domains/

- For $d = 2$, Algorithm 1 resolves discrepancies between $B_{\text{Alice}}\text{VAL}(\pi, G)$ and $B_{\text{Alice}}B_{\text{Bob}}\text{VAL}(\pi, G)$.
- For $d = 3$, Algorithm 1 resolves discrepancies between $B_{\text{Alice}}B_{\text{Bob}}\text{VAL}(\pi, G)$ and $B_{\text{Alice}}B_{\text{Mary}}B_{\text{Bob}}\text{VAL}(\pi, G)$.
- For $d = 5$, Algorithm 1 resolves discrepancies between $B_{\text{Alice}}B_{\text{Mary}}B_{\text{Charlie}}B_{\text{Rose}}B_{\text{Bob}}\text{VAL}(\pi, G)$ and $B_{\text{Alice}}B_{\text{Charlie}}B_{\text{Rose}}B_{\text{Bob}}\text{VAL}(\pi, G)$.

In each IPC domain, Alice has at her avail appropriate communicative and ontic actions. For example, in the Driverlog domain Alice may either inform Bob that he holds a false belief about the location of an object; or move the object to where Bob believes it to be. See Appendix E.3 for details on the encoding of the DriverLog domain and an example of a discrepancy resolving plan computed by the planner.

E Exemplary Domains

Here we detail some of the domain and problem files used in the evaluation described in Section 5. All files are in the PDKBDDL format which is a variant of PDDL. For the sake of readability, the domains seen here are partial and contain a subset of the actions found in the domains used in our evaluation. Moreover, to better illustrate the PDKBDDL format, some actions are grounded versions of the actions used in our experiments.

E.1 Corridor

domain-corridor.pdkbddl

```
(define (domain corridor)

    ; This specifies the agents in the set of agents Ag
    (:agents a b c)

    (:types loc door)

    (:predicates
        (secret ?agent)
        (at ?agent - agent ?l - loc)
        (connected ?l1 ?l2 - loc)
        (door_open ?d - door)
        (dummy)
    )

    (:action shareSecret
        :derive-condition   (at $agent$ l1)
        :parameters         (?a ?as - agent)
        :precondition       (and (at ?a l1) [?a](secret ?as))
        :effect             (and
                                (forall ?a2 - agent
                                    (when    (and (door_open dl1l2) (at ?a2 l2))
                                         [?a2](secret ?as))
                                         )

                                (forall ?a3 - agent
                                    (when    (and (at ?a3 l1))
                                         [?a3](secret ?as))
                                         )
                            )
    )

    (:action informDoorOpen
      :derive-condition   (always)
      :parameters         (?a1 - agent ?d - door)
      :precondition       (and (door_open ?d))
      :effect             (and
                               [?a1](door_open ?d))

    )
    (:action closeDoor
      :derive-condition   never
      :parameters         (?d - door)
      :precondition       (and (dummy))
      :effect             (and

                                (!door_open ?d))

    )

)
```

Since planning in RP-MEP is from the perspective of a single agent[8], all fluents are implicitly preceded by the beliefs of that agent. In our evaluation, a single agent Alice resolves all discrepancies and so planning in RP-MEP is done from her perspective. For instance, in the *informDoorOpen* action in the Corridor domain, the effect [?a1](door_open ?d) is implicitly preceded by [Alice] such that the effect of the action is that Alice believes that agent a1 believes that door d is open.

The value of derive-condition in the *shareSecret* action, (at $agent$ l1), specifies the condition for mutual awareness. In this case, agents are aware of the *shareSecret* action when they are at location l1, where this *grounded*[9] action is performed. For instance, if the specified depth of nested belief in the problem file is 2, then if agents b and c are in l1 and agent a is sharing her secret, then agent b will believe the secret and will believe that agent c believes the secret. In addition, agent c will believe the secret and will believe that agent b believes the secret (and so on and so forth for all agents in l1).

In Line 3 of Algorithm 1 we use RP-MEP's machinery to classically encode the MEP domain and initial state. Here is the partial classically encoded corridor domain:

```
domain-corridor.pddl
(define (domain corridor)

    (:requirements :strips :conditional-effects :disjunctive-preconditions)

    (:predicates
        (not_at_a_l1)
        (not_at_a_l2)
        (not_at_b_l1)
        (not_at_b_l2)
        (not_at_c_l1)
        (not_at_c_l2)
        (not_connected_l1_l1)
        (not_connected_l1_l2)
        (not_connected_l2_l1)
        (not_connected_l2_l2)
        (not_door_open_dl1l2)
        (not_dummy)
        (not_secret_a)
        (not_secret_b)
        (not_secret_c)
        (Ba_not_at_a_l1)
        (Ba_not_at_a_l2)
        (Ba_not_at_b_l1)
        (Ba_not_at_b_l2)
        (Ba_not_at_c_l1)
        (Ba_not_at_c_l2)
        (Ba_not_door_open_dl1l2)
        (Ba_not_dummy)
        (Ba_not_secret_a)
        (Ba_not_secret_b)
        (Ba_not_secret_c)
```

[8]This agent is also called the *root agent* by Muise et al. (Muise et al. 2015b, 2021).

[9]For the lifted action, the derive-condition would be (at $agent$?l), where l is the location where the secret is being shared.

```
(Ba_at_a_l1)
.
.
.

(:action share_a_a_l1
:precondition (and (at_a_l1)
                   (Ba_secret_a)
              )
:effect (and
             ; #16607: origin
             (when (and (at_c_l1))
                   (Bc_secret_a))

             ; #29379: <==closure== 39863 (pos)
             (when (and (at_b_l1))
                   (Pb_secret_a))

             ; #30779: <==closure== 16607 (pos)
             (when (and (at_c_l1))
                   (Pc_secret_a))

             ; #32347: <==closure== 89586 (pos)
             (when (and (and (at_a_l2)
                        (door_open)))
                   (Pa_secret_a))

             ; #39863: origin
             (when (and (at_b_l1))
                   (Bb_secret_a))

             ; #49490: origin
             (when (and (at_a_l1))
                   (Ba_secret_a))

             ; #51412: <==closure== 73425 (pos)
             (when (and (and (door_open)
                        (at_b_l2)))
                   (Pb_secret_a))

             ; #54730: <==closure== 49490 (pos)
             (when (and (at_a_l1))
                   (Pa_secret_a))

             ; #55625: origin
             (when (and (and (door_open)
                        (at_c_l2)))
                   (Bc_secret_a))

             ; #73425: origin
             (when (and (and (door_open)
                        (at_b_l2)))

            ; #89586: origin
             (when (and (and (at_a_l2)
                        (door_open)))
                   (Ba_secret_a))

             ; #93603: <==closure== 55625 (pos)
```

```
                        (when (and (and (door_open)
                                        (at_c_l2)))
                              (Pc_secret_a))

                        ; #10374: <==uncertain_firing== 49490 (pos)
                        (when (and (not (not_at_a_l1)))
                              (not (Pa_not_secret_a)))

                        ; #14825: <==unclosure== 33844 (neg)
                        (when (and (at_b_l1))
                              (not (Bb_not_secret_a)))

                        .
                        .
                        .
```

Notice the propositional fluents created during the classical encoding process to represent every RML in the MEP domain. Moreover, note the ancillary conditional effects that are automatically added during the encoding process to enforce closure under the KD45 axioms.

In this problem instance, π is the single action shareSecret(a,a) and G is [b](secret a) $\wedge$![c](secret a). We included here the grounded shareSecret action that only allows the agent to share her secret in location l1. Therefore the action shareSecret(a,a) means that agent a is sharing agent a's secret in l1. As described in Appendix D.2, agent a (Bob) has the epistemic goal that agent b believes his secret, with agent c **not** believing his secret. Thus, the regression formula $\phi = VALID(\pi_c, G_c)$, computed from the classically encoded domain, is the following (converted to DNF by the Python library sympy):

```
Regression formula for corridor problem
(Ba_secret_a & at_a_l1 & at_b_l1 & ~at_c_l1 & ~at_c_l2) |
(Ba_secret_a & at_a_l1 & at_b_l1 & ~at_c_l1 & ~door_open_dl1l2) |
(Ba_secret_a & at_a_l1 & at_b_l2 & door_open_dl1l2 & ~at_c_l1 & ~at_c_l2)
```

For instance, the first disjunct means that a necessary and sufficient condition for the achievement of G by π is for i. agent a to believe a's secret, ii. agent a to be in l1, iii. agent b to be in l1 and iv. agent c to **not** be in either l1 or l2. If this disjunct is satisfied, then following the shareSecret action, agent b will come to learn a's secret and agent c will not, which satisfies agent a's epistemic goal [b](secret a) $\wedge$![c](secret a). Finally, the following is the negation of the regression formula above, converted to DNF by sympy:

```
Negation of regression formula for corridor problem converted to DNF
at_c_l1 | ~Ba_secret_a | ~at_a_l1 | (at_c_l2 & door_open_dl1l2) |
(~at_b_l1 & ~at_b_l2) | (~at_b_l1 & ~door_open_dl1l2)
```

For instance, if agent c is at l1 then the goal cannot be satisfied by π since agent c will come to learn agent a's secret. Moreover, the fourth disjunct says that if agent c is at l2 and the door between l1 and l2 (*dl1l2*) is open, then the plan is not valid since agent c will come to learn agent a's secret. Both of these conditions of course make the single action plan shareSecret(a,a) not valid.

The following is the partial (classically encoded) initial state and the goal given to the classical planner:

```
prob-corridor.pddl
(define (problem corridor-prob)

    (:domain corridor)

    (:init
        (door_open_dl1l2)
        (Ba_not_door_open_dl1l2)

        (at_b_l1)
        (Ba_at_b_l1)

        (not_at_c_l1)
        (Ba_not_at_c_l1)

        (at_c_l2)
        (Ba_at_c_l2)

        .
        .
        .

    (:goal
        (or
            (and (Ba_Ba_secret_a) (Ba_secret_a) (Ba_at_a_l1) (at_a_l1) (Ba_at_b_l1)
            (at_b_l1) (Ba_not_at_c_l1)
            (not_at_c_l1) (Ba_not_at_c_l2) (not_at_c_l2) )

            (and (Ba_Ba_secret_a) (Ba_secret_a) (Ba_at_a_l1) (at_a_l1) (Ba_at_b_l1)
            (at_b_l1) (Ba_not_at_c_l1)
            (not_at_c_l1) (Ba_not_door_open_dl1l2) (not_door_open_dl1l2) )

            (and (Ba_Ba_secret_a) (Ba_secret_a) (Ba_at_a_l1) (at_a_l1) (Ba_at_b_l2)
            (at_b_l2)
            (Ba_door_open_dl1l2) (door_open_dl1l2) (Ba_not_at_c_l1) (not_at_c_l1)
            (Ba_not_at_c_l2) (not_at_c_l2) )

            (and (Ba_at_c_l1) (at_c_l1) )

            (and (Ba_not_Ba_secret_a) (not_Ba_secret_a))

            (and (Ba_not_at_a_l1) (not_at_a_l1))

            (and (Ba_at_c_l2) (at_c_l2) (Ba_door_open_dl1l2) (door_open_dl1l2) )

            (and (Ba_not_at_b_l1) (not_at_b_l1) (Ba_not_at_b_l2) (not_at_b_l2) )

            (and (Ba_not_at_b_l1) (not_at_b_l1) (Ba_not_door_open_dl1l2)
            (not_door_open_dl1l2) )

        )
    )
)
```

We can see that while Alice believes that the door between l1 and l2 is open, she also believes that agent a (Bob) believes that it is not open. This is a discrepancy! Moreover, notice the *disjunctive* goal given to the classical planner that comprises the disjuncts of DNF(ϕ) and DNF($\neg\phi$) (where ϕ is the regression formula $VALID(\pi_c, G_c)$, as described in Section 4).

Given this disjunctive goal (encoded in prob-corridor.pddl), there are two possible optimal plans. Here is one of these optimal discrepancy resolving plans, produced by Fast Downward with an admissible heuristic:

```
Discrepancy Resolving Plan #1
1. closedoor_dl1l2
```

That is, Alice will close the door which will resolve the discrepancy by making Bob's (agent a's) plan valid.

This plan satisfies the disjunct

```
(and (Ba_Ba_secret_a) (Ba_secret_a) (Ba_at_a_l1) (at_a_l1) (Ba_at_b_l1) (at_b_l1)
    (Ba_not_at_c_l1) (not_at_c_l1) (Ba_not_door_open_dl1l2) (not_door_open_dl1l2) )
```

since agent a already believes that the door is closed and after closing the door, Alice will also believe that the door is closed. The other conjuncts in this disjunct are already satisfied in the initial state.

The other optimal discrepancy resolving plan that Fast Downward produced is the following:

```
Discrepancy Resolving Plan #2
1. informdooropen_a_dl1l2
```

That is, Alice will inform Bob (agent a) that the door is open which will resolve the discrepancy by making Bob aware that his plan is not valid. This plan satisfies the disjunct

```
(and (Ba_at_c_l2) (at_c_l2) (Ba_door_open_dl1l2) (door_open_dl1l2) )
```

Recall that we create three versions of each problem instance. When we remove a subset of communicative actions from the domain, the first discrepancy resolving plan will be computed, involving Alice closing the door. When we remove a subset of ontic actions (i.e., the closing/opening of doors), the second discrepancy resolving plan will be computed, involving Alice informing Bob that the door is open. If we do not modify the domain, since both plans are of equal length, we observed that Fast Downward chose the action that appeared ‘earlier’ in the domain file.

E.2 BW4T

```
domain-bw4t.pdkbdl

(define (domain bw4t)

  (:agents a b)
  (:requirements :strips :typing )

  (:types color room place id)

  (:predicates
                (at ?ag - agent ?place - place)
                (block ?colorid - color ?id - id ?placeid - room)
                (connected ?r1 - place ?r2 - place)
                (connected ?r1 - place ?r2 - room)
                (connected ?r1 - room ?r2 - place)
                (blockin ?colorid - color ?id - id ?dropzone - room)
                (in ?ag - agent ?roomid - room)
                (holding ?ag - agent ?colorid - color ?id - id ?placeid - room)
                (atblock ?colorid - color ?id - id ?placeid - room)
                (droplocation ?loc - room)
                (handempty)
   )



 (:action goto
            :derive-condition   always
       :parameters (?ag - agent ?placeid - place ?currplace - room)
       :precondition (and (in ?ag ?currplace) (handempty)
                     (connected ?currplace ?placeid) )
       :effect
       (and (not(in ?ag ?currplace)) (at ?ag ?placeid)
        )
  )


  (:action gotodrop
            :derive-condition   always
         :parameters (?ag - agent ?placeid - place ?currplace - room)
         :precondition (and (in ?ag ?currplace) (not (handempty))
                              (connected ?currplace ?placeid) )
         :effect
         (and (not(in ?ag ?currplace)) (at ?ag ?placeid))
  )


 (:action pckUp
       :derive-condition   (at $agent$ ?placeid)
       :parameters         (?ag - agent ?colorid - color ?id - id ?placeid - room)
       :precondition       (and (in ?ag ?placeid)
                                (block ?colorid ?id ?placeid)
                                (atblock ?colorid ?id ?placeid)
                                (handempty)
              (not (droplocation ?placeid))
         )
       :effect
                          (and (holding ?ag ?colorid ?id ?placeid)
            (not (handempty)) (not (atblockbot))))




  (:action putdown
       :derive-condition   (at $agent$ ?placeid)
       :parameters        (?ag - agent ?colorid - color ?id - id ?placeid - room
                           ?dropzone - room)
       :precondition      (and (holding ?ag ?colorid ?id ?placeid)
                          (in ?ag ?dropzone) (droplocation ?dropzone))


       :effect
            (and (not (holding ?ag ?colorid ?id ?placeid))
                 [?ag](!block ?colorid ?id ?placeid)
                 [?ag](block ?colorid ?id ?dropzone)
                 (blockin ?colorid ?id ?dropzone)
                 (handempty)
           )
   )




  (:action informAboutBlockLocation
      :derive-condition   always
      :parameters         (?a1 - agent ?colorid - color ?id - id ?placeid - room)
      :precondition       (and (atblock ?colorid ?id ?placeid))
      :effect             (and
                               [?a1](atblock ?colorid ?id ?placeid))

   )

 (:action informAboutBlockNOTLocation
      :derive-condition   always
      :parameters         (?a1 - agent ?colorid - color ?id - id ?placeid - room)
      :precondition       (and (!atblock ?colorid ?id ?placeid))
      :effect             (and
                               [?a1](!atblock ?colorid ?id ?placeid))

   )
```

```
)
```

The last two actions can be used by Alice to inform any agent that a certain block is either in *placeid* or is not in *placeid*. The following (partial) problem file describes one of the *Dude, Where's my Medical Kit?* problem instances where agent b (Bob) holds a false belief about the location of the medical kit (blue block #1). As a reminder, since [Alice] implicitly precedes all fluents, we can see that Bob's beliefs are false from Alice's perspective since she believes that the block is in room b2 and she also believes that Bob believes that the block is in room c3.

```
problem-bw4t.pdkbdl

(define (problem prob4)
  (:domain bw4t)

 (:objects
                    blue red - color
                    1 2 - id
                    roomc1 roomc2 roomc3 roomb1 roomb2 roomb3 dropzone - room
                    frontdropzone righthalld lefthalld - place
                    frontroomc1 frontroomc2 frontroomc3 lefthallc righthallc - place
                    frontroomb1 frontroomb2 frontroomb3 lefthallb righthallb - place
 )

 (:depth 2)
 (:init
    .
    .
    .
        (!atblock blue 1 roomc3)
        (atblock blue 1 roomb2)
   [b](atblock blue 1 roomc3)
   [b](!atblock blue 1 roomb2)
)


 ))
```

To model the *Can You Hear Me??* scenario described in Appendix D.1, we introduce the action *sendComm*:

```
domain-epistemic-goal-bw4t.pdkbdl
.
.
.
(:action sendComm
      :derive-condition   always
      :parameters         (?a1 - agent ?a2 - agent ?colorid - color ?id - id
                           ?loc - loc)
      :precondition       ([?a1](blockin ?colorid ?id ?loc))
      :effect             (and

                               (when
                                    (workingCommDevice ?a2)

                                  [?a2](blockin ?colorid ?id ?loc)
```

```
                                          )

                                     )

)
.
.
.
```

As can be seen from the following partial initial state, Bob (agent b) initially believes that Mary's (agent c) communication device is working properly while Alice believes that it is not.

problem-epistemic-goal-bw4t.pdkbdl

```
.
.
.
(:init

        (!workingCommDevice c)
        [b](workingCommDevice c)
.
.
.
```

Recall that Bob's plan π is to deliver the block (medical kit) to the drop zone and then send a communication to Mary:

[*move*(Bob,Hall,RoomB),
pckUp(Bob,Kit1,RoomB),
move(Bob,RoomB,Hall),
dropOff(Bob,Kit1,Hall),
sendComm(Bob,Mary,at(Kit1,Hall)))],

To resolve the discrepancy about Bob's plan, Alice can inform Bob that Mary's communication device is not working. This is the discrepancy resolving plan computed by the classical planner given the disjunctive goal based on the computed regression formula:

Discrepancy Resolving Plan (BW4T epistemic goal)

```
1. informcommdevicenotworking_b_c
```

That is, agent b (Bob) is informed that agent c's (Mary's) communication device is not working. Then, Bob will believe that his plan is not valid.

E.3 DriverLog

In the DriverLog domain, Alice is assumed to be an embodied robot that can act in the environment. She can therefore pick up packages and move them to other locations in order to resolve discrepancies. Here is a partial encoding of the domain in PDKBDDL with the 'pick up package' action that can be executed by Alice the robot:

domain-driverlog.pdkbdl

```

(define (domain driverlog)

  (:agents a b c d)
  (:requirements :strips :typing )

  (:predicates

        (OBJ ?obj)
            (TRUCK ?truck)
        (LOCATION ?loc)
```

```
                (driver ?d)
                (at ?obj ?loc)
                (in ?obj1 ?obj)
                (driving ?d ?v)
                (link ?x ?y) (path ?x ?y)
                (empty ?v)
        (robotholdingpackage ?obj)
        (disc_resolution)
        (dummy)
        (atRobot ?loc)
)


    (:action ROBOT-PICK-UP-PACKAGE
:derive-condition   always
  :parameters
   (?obj
    ?loc)
  :precondition
   (and (OBJ ?obj) (atRobot ?loc) (LOCATION ?loc)
    (at ?obj ?loc))
  :effect
   (and (not (at ?obj ?loc)) (robotholdingpackage ?obj))

   )

.
.
.
)
```

In one of the problem instances, we have that Alice (the robot) believes that package1 is at location s1 while also believing that Bob (agent a) believes that the package is at location s0.

```
domain-driverlog.pdkbdl
.
.
.
(:init
        (at package1 s1)
        (!at package1 s0)
        [a](at package1 s0)
        [a](!at package1 s1)
.
.
.
```

Alice is initially at s1. And so, when the disjunctive goal is given to the classical planner, the planner comes up with the following plan:

```
Discrepancy Resolving Plan (DriverLog)
1. robot-pick-up-package_package1_s1
2. robot-move_s1_s0
3. robot-drop-off-package_package1_s0
```

That is, to resolve the discrepancy, Alice the robot picked up the package from s1 and delivered it to s0.

Higher order discrepancies Finally, let us see what discrepancy resolution looks like when $d = 5$. As mentioned in D.3, for $d = 5$, Algorithm 1 resolves discrepancies between $B_{\text{Alice}}B_{\text{Mary}}B_{\text{Charlie}}B_{\text{Rose}}B_{\text{Bob}}\text{VAL}(\pi, G)$ and $B_{\text{Alice}}B_{\text{Charlie}}B_{\text{Rose}}B_{\text{Bob}}\text{VAL}(\pi, G)$.

For instance, in one of our problem instances we have that Alice believes that Mary (agent a) falsely believes that Charlie (agent b) believes that Rose (agent c) believes that Bob (agent d) believes that package1 is at location s0. At the same time, Alice believes that Charlie believes that Rose believes that Bob believes that package1 is at location s1. This is encoded as follows in PDKBDDL:

Partial initial state in Driverlog domain
```
[a][b][c][d](at package1 s0)
[a][b][c][d](!at package1 s1)
[b][c][d](!at package1 s0)
[b][c][d](at package1 s1)
```

In the domain file, we have the following inform action that can inform some agent ?a1 about the beliefs of agent ?a2 about the beliefs of agent ?a3 about the beliefs of agent ?a4 about the location of some object.

Inform action in Driverlog domain
```
(:action informAboutHigherOrderBeliefObjLoc
      :derive-condition   (always)
      :parameters         (?a1 - agent ?a2 - agent ?a3 - agent ?a4 - agent
                           ?obj  ?loc)
      :precondition       (and [?a2][?a3][?a4](at ?obj ?loc) (OBJ ?obj)
                               (LOCATION ?loc) )
      :effect             (and
                               [?a1][?a2][?a3][?a4](at ?obj ?loc))

    )
```

In the classically encoded problem, one of the disjuncts in the disjunctive goal is:

```
    (and (Ba_Bb_Bc_Bd_at_package1_s1) (Bb_Bc_Bd_at_package1_s1))
```

And so, in order to satisfy the disjunctive goal the classical planner generates the following discrepancy resolving plan:

Discrepancy Resolving Plan for higher order discrepancy (DriverLog)
```
1. informAboutHigherOrderBeliefObjLoc_a_b_c_d_package1_s1
```

That is, Alice informs Mary (agent a) that Charlie (agent b) believes that Rose (agent c) believes that Bob (agent d) believes that package1 is at location s1.

F User Study

As discussed in Section 6, we set out to test the following hypotheses in our study:

H1: Participants will be more likely to generate a valid plan to achieve their goal when presented with information *derived from a discrepancy resolving plan*, compared to the likelihood of generating a valid plan prior to receiving the information.

H2: Participants will be more likely to correctly predict another agent's plan when presented with information *from a discrepancy resolving plan*, compared to their prediction prior to receiving the information.

To test these hypotheses, participants were asked to imagine that they are part of an emergency response team whose members must communicate with one another and obtain various items (such as medical kits). Participants were told that they are partnered with a virtual assistant meant to provide decision support, and were presented with two scenarios, mirroring two of the BW4T scenarios used in our evaluation and discussed in Appendix D.1. Initially, participants were given incomplete or incorrect information, causing discrepancies. We had a total of 40 participants who were recruited via Amazon Mechanical Turk and were paid upon completing the questionnaire via an online platform[10]. Participants had no prior knowledge about the study.

[10]https://www.surveymonkey.com/

F.1 Testing H1

In the first scenario (mirroring the *'Dude, Where's my Medical Kit?'* scenario discussed in Appendix D.1), participants were told that their goal is to acquire a medical kit from the supply tent of the base. Subsequently, participants were given *incorrect* information about the location of the supply tent. Participants were then asked where they would go in order to obtain a medical kit and were given 4 options from which to choose: the west, east, north, and south ends of the base. This is a proxy for participants' beliefs about plan validity. In other words, if participants believe that one of the 4 possible plans is valid (i.e., going to a certain location in the base) then we assume that participants will follow that plan and head to the respective location.

Next, participants were informed by their virtual assistant of the true location of the supply tent and were asked, again, where they would go in order to obtain a medical kit. The assistant's communication is a natural language representation of the inform actions in the discrepancy resolving plan generated by RP-MEP. That is, the content of an inform action is processed using a number of simple natural language templates. For example, *at*(SupplyTent,BaseWestEnd) is converted to *"SupplyTent is at BaseWestEnd"*. The discrepancy resolving plan resolves a discrepancy perceived by the virtual assistant (who is assumed to hold correct beliefs) between its beliefs and its beliefs about the participant's beliefs, pertaining to the validity of the participant's plan.

F.2 Testing H1 - Results

Prior to being informed by the virtual assistant about the true location of the supply tent, 0 participants generated a valid plan to obtain the medical kit. After being informed by the virtual assistant about the true location of the medical kit, all participants correctly generated a valid plan to obtain the medical kit, which indicates that their misconception regarding plan validity was resolved.

These results are consistent with **H1** since participants were more likely to generate a valid plan to achieve their goal *after* being presented with information derived from a discrepancy resolving plan, compared to the likelihood of generating a valid plan before receiving the information.

F.3 Testing H2

In the second scenario (a slight simplification of the *'Where Does he Think he's Going?'* scenario discussed in Appendix D.1), participants were told that they have a human teammate whose goal it is to acquire a medical kit from the supply tent. Participants were led to *incorrectly* believe that their teammate, like the participants before being informed by the virtual assistant, believes that the supply tent is in the old location. Participants are told that their plan is to send a message to the teammate's communication device. The participants, however, do not know that the teammate's communication device is faulty. Participants were then asked to predict the location to which their teammate will go in order to obtain the medical kit, after the participant's plan of sending the message is executed. Once again, participants were given 4 options from which to choose: the west, east, north, and south ends of the base. This is a proxy for participants' beliefs about their teammate's beliefs about plan validity. In other words, if participants believe that their teammate believes that one of the 4 possible plans is valid (i.e., going to a certain location in the base) then we assume that participants will predict that their teammate will follow that plan and head to the respective location.

Next, participants were informed by their virtual assistant that their teammate's communication device is not working and were asked, again, to predict the teammate's plan following the execution of the participant's message-sending plan. As before, the assistant's communication is a natural language representation of the inform actions in the discrepancy resolving plan generated by RP-MEP. This time, the communication pertains to the teammate's communication device not working. The discrepancy resolving plan resolves a discrepancy perceived by the virtual assistant (who is assumed to hold correct beliefs) between its beliefs about the status of the teammate's communication device and its beliefs about the participant's beliefs about the status of the teammate's communication device. See also Appendix E.2 for the domain encoding and the generated discrepancy resolving plan.

F.4 Testing H2 - Results

Prior to being informed by the virtual assistant about the status of their teammate's communication device, 0 participants correctly predicted their teammate's plan. After being informed by the virtual assistant about the status of their teammate's communication device, 95% of participants correctly predicted their teammate's plan which indicates that their misconception regarding plan validity was resolved.

A McNemar's test determined that participants' predictions about their teammate's plan after receiving information from their virtual assistant were significantly more accurate than their predictions prior to receiving this information (95% compared to 5%, $p < 0.001$). These results are consistent with **H2** since participants more accurately predicted their teammate's plan *after* being presented with information derived from a discrepancy resolving plan, compared to their prediction before receiving the information.