On Model Reconciliation: How to Reconcile When Robot Does not Know Human's Model?

Ho Tuan Dung Tran Cao Son
Department of Computer Science, New Mexico State University, Las Cruces, USA
dungho@nmsu.edu stran@nmsu.edu

The Model Reconciliation Problem (MRP) was introduced to address issues in explainable AI planning. A solution to a MRP is an *explanation* for the differences between the models of the human and the planning agent (robot). Most approaches to solving MRPs assume that the robot, who needs to provide explanations, knows the human model. This assumption is not always realistic in several situations (e.g., the human might decide to update her model and the robot is unaware of the updates).

In this paper, we propose a dialog-based approach for computing explanations of MRPs under the assumptions that (*i*) the robot does not know the human model; (*ii*) the human and the robot share the set of predicates of the planning domain and their exchanges are about action descriptions and fluents' values; (*iii*) communication between the parties is perfect; and (*iv*) the parties are truthful. A solution of a MRP is computed through a dialog, defined as a sequence of rounds of exchanges, between the robot and the human. In each round, the robot sends a potential explanation, called *proposal*, to the human who replies with her evaluation of the proposal, called *response*. We develop algorithms for computing proposals by the robot and responses by the human and implement these algorithms in a system that combines imperative means with answer set programming using the multi-shot feature of `clingo`.

1 Introduction

Plan explanation is important for human and AI system (or, robot, as we will refer to the AI system in this paper) to work together in human-aware planning and scheduling. The *model reconciliation problem* (MRP), introduced by [1], is proposed as a way for the robot to explain its solutions to the human. Formally, a MRP is a tuple (π^*, M_R, M_H) where M_R is a planning problem of the robot and M_H is an approximation of M_R, representing the planning problem that the human considers, and π^* is an optimal plan for M_R. A solution of a MRP (π^*, M_R, M_H), called an *explanation*, is a pair $\varepsilon = (\varepsilon^+, \varepsilon^-)$ such that M_H^*, which is obtained from M_H by adding ε^+ to it and removing ε^- from it, will have π^* as one of its optimal plan.

Several approaches to solving a MRP have been proposed. [1] formalized the problem as a search problem and computed an explanation by searching through the space of potential explanations[1]. [9] solved the problem using answer set programming and were able to deal with situations when some actions are missing in the human model or the initial states in the two models are different. [14] identified an appropriate explanation by exploiting the existing hitting set duality between minimal correction sets and minimal unsatisfiable sets. All of these approaches consider the problem in the context of classical planning and assume that the robot knows the human model M_H. [13] argued that it is not always realistic to assume that the robot knows M_H and attempted to address this issue by investigating the *model-free*

[1] A expanded version of this paper is in [12]. As the key ideas are the same, we will simply refer to [1] to save space.

Y. Lierler, J.F. Morales, C. Dodaro, V. Dahl, M. Gebser, T. Tekle (Eds.):
International Conference on Logic Programming 2022 (ICLP'22).
EPTCS 364, 2022, pp. 27–48, doi:10.4204/EPTCS.364.4

model reconciliation problem in which the human model is not available in declarative form and not known to the planning agent. Their approach assumes that the robot and human model are represented by Markov Decision Processes (MDP) and the differences between the models are on the transition probabilities, the reward function and the discounting factor. It then applies reinforcement learning to solve the problem.

In this paper, we present an approach to solving MRPs when the robot *does not know* the human model. Different from the model-free model reconciliation problem considered by [13], we assume that the robot and human model are represented by logic programs. In our approach, an explanation is computed via a dialog between the robot and the human, which is defined as a sequence of rounds of information exchange. In each round, the robot presents an explanation (about its optimal plan) to the human, who will respond with her evaluation of the proposal. The response of the human will allow the robot to understand the human model. It will also decide whether another round of information exchange is necessary. In this paper, we assume that the communication between the robot and the human is perfect and the two parties are truthful. Similar to [1], we assume that the human model is an approximation of the robot model. These assumptions ensure that the human and the robot can *understand* each other and the termination of the dialog.

The main contributions of this paper are a framework for model reconciliation when the robot does not know the human model and its implementation. Specifically, we (*i*) define the notions of a proposal (from the robot) and a response (from the human); (*ii*) develop algorithms for computing proposals and responses; and (*iii*) implement the algorithms for computing an explanation via two dialogue controllers, one for the human and one for the robot.

2 Preliminaries

Answer set programming (ASP), introduced by [8] and [10], refers to the approach of problem solving using logic programming under the answer set semantics. A logic program Π is a set of rules of the form

$$a_0 \leftarrow a_1, \ldots, a_m, \mathit{not}\ a_{m+1}, \ldots, \mathit{not}\ a_n$$

where $0 \leq m \leq n$, each a_i is an atom of a propositional language, and *not* represents (default) negation. Intuitively, a rule states that if all positive literals a_i are believed to be true and no negative literal $\mathit{not}\, a_i$ is believed to be true, then a_0 must be true. Semantically, a logic program induces a set of answer sets, being distinguished models of the program determined by the answer set semantics; see the paper by [4] for details. Answer sets of a program can be computed using the solver *clingo* [3] or `dlv` [2].

Planning Using ASP. A planning problem—as described using PDDL by [5] is a triple (I,G,D), where I and G encode the initial state of the world and the goal, respectively; and D (the domain) specifies the actions and their preconditions and effects. Given a problem $M = (I,G,D)$, we can translate it into a program $\pi(M,n)$ whose answer sets correspond one-to-one to plans of maximal length n of M [7]. Standard encoding for computing solutions of planning problems using answer set solvers are available. In this paper, we will make use of an encoding similar to that presented by [9] that utilizes the translator from planning problems in PDDL to logic program facts[2] because it simplifies our experiments. Program $\pi(M,n)$ consists of different groups of rules:

[2] `https://github.com/potassco/plasp`

- **Facts:** These atoms define object constants, types of objects, actions, the initial state, and the goal state.
- **Reasoning About Effects of Actions:** Rules in this group make sure that an action can only be executed if all of its preconditions are true and all of the effects of the actions become true. We use $h(l,t)$ to denote that l is true at step t for $1 \leq t \leq n$.
- **Goal Enforcement and Action Generation:** To generate action occurrences, we use the choice rule "$1\{occurs(A,T) : action(action(A))\}1 \leftarrow not\ goal(T)$ and to enforce that the goal is satisfied at n, we use the rule $\leftarrow not\ goal(n)$ where $goal(i)$ denotes that the goal of the plan satisfies at step i.

By computing answer sets of $\pi(M,0),\ldots,\pi(M,k),\ldots$, we can compute optimal plan for M. It has been proved that if k is the smallest integer such that $\pi(M,k)$ has an answer set A then A contains an optimal plan of M.

Model Reconciliation Problem. Given two planning problems $M_r = (I_r, G_r, D_r)$ and $M_h = (I_h, G_h, D_h)$, a *model reconciliation problem* (MRP) is defined by a tuple $\langle \pi^*, M_r, M_h \rangle$, where π^* is a cost-minimal solution for M_r. As in other papers, we define the cost of a plan by its length. A solution for an MRP is a multi-model explanation $\varepsilon = (\varepsilon^+, \varepsilon^-)$, which creates a model M_h^* from M_h such that π^* is also a cost-minimal solution of M_h^* by inserting ε^+ to M_h and removing ε^- from M_h, where ε^+ (or ε^-) are sets containing some initial conditions, action preconditions/effects, or goals. It is required that the changes in the model of the human must be consistent with the robot's model.

For ease of reading, we will use the well-known Blocksworld domain as running example in this paper. The fluents in this domain are $clear(x)$, $on(x,y)$, $handempty$, $holding(x)$, and $onTable(x)$ and the actions are $unstack(x,y)$, $stack(x,y)$, $pickup(x)$, and $putdown(x)$ with their usual meanings. Let us consider the planning problem depicted in Figure 1: the initial state (left) and the goal state consisting of only the literal $on(b,a)$ (right). For space reason, we omit the details of the domain model of the robot and/or the human. It is easy to see that the optimal plan for the robot in this setting is $\langle unstack(a,b); putdown(a); pickup(b); stack(b,a) \rangle$. Assume that $stack(b,a)$ has only one precondition $clear(a)$, i.e., a is clear (which is true in the initial state). In that case, the optimal plan for the human is $\langle stack(b,a) \rangle$.

Figure 1: Achieving $on(b,a)$ given $on(a,b)$ and $on(b,table)$

3 Computing Solutions of MRPs via Dialogues

In this section, we will present our approach for computing a solution of MRPs when the robot does not know the model of planning model of the human (M_h) and vice versa. We assume that the two are honest and know that the human model might be incomplete or contain incorrect information. Given that the robot does not know the human model, the only way for the robot to explain to the human the differences between the two problems is to understand the human model through interactions.

3.1 Formalization: Explanations, Proposals, Responses, and Dialogues

Informally, the interaction gives the robot and the human the opportunities to inform each other about their models and, ultimately, leads to an agreement. We illustrate this interaction in the following ex-

ample, where, assuming that the human model does not contain the precondition $holding(x)$ for the action $stack(x,y)$, $clear(x)$ for $pickup(x)$, and $on(x,y)$ for $unstack(x,y)$, where x and y denote blocks, respectively.

Example 1 (Sample Dialogue) Consider the MRP given by the Blocksworld domain problem in Figure 1. Intuitively, a dialogue between the robot and the human can be as follows:

1. *Robot* (to Human): my optimal plan for the planning problem is $\pi^* = \langle unstack(a,b); putdown(a); pickup(b); stack(b,a)\rangle$;
2. *Human* (to Robot): no, my is $\langle stack(b,a)\rangle$ (this is because the $stack(b,a)$ action only requires $clear(a)$ as its precondition in the human model);
3. *Robot*: your plan is not valid; the $stack(b,a)$ action should have the precondition $holding(b)$ (this is because the robot realizes that the condition that prevents the action $stack(b,a)$ to occur in the initial state is $holding(b)$);
4. *Human*: ah, okay; in that case, my optimal plan is $\langle pickup(b); stack(b,a)\rangle$ (because the human model does not require $clear(b)$ as a precondition of $pickup(b)$);
5. *Robot*: your plan is still not valid; the $pickup(b)$ action should have the precondition $clear(b)$;
6. *Human*: I see. In that case, I agree with you. I'll modify the two actions in my model.

Observe that the robot model still differs from the human model in the action *unstack*. However, the dialogue stops because the optimal plan w.r.t. the human model also has length 4. In this example, the robot has no way to realize that there still exists differences between the two domains even though the human agrees with the robot. This is different in the next example.

Consider the problem depicted in Figure 2 with similar assumptions as in Example 1. It is easy to see that the sequence $\langle unstack(a,c); putdown(a); unstack(c,b); stack(c,b)\rangle$ is a plan achieving the goal $\{on(c,b)\}$ in the human model, after the human updates the actions *pickup* and *stack* with the correct preconditions as informed by the robot in Steps 3 and 5 in Example 1. Because this is not a plan with respect to the robot model, the robot will realize that the specification of the action *unstack* in the human model is incorrect, e.g., $unstack(a,c)$ is not executable in the initial state because $on(a,c)$ is a precondition of this action and it is not true in the initial state. It can be easily verified that after the robot informs the human that $unstack(x,y)$ requires $on(x,y)$ and the human updates his/her model with this information, the domain of the human is identical to the domain of the robot.

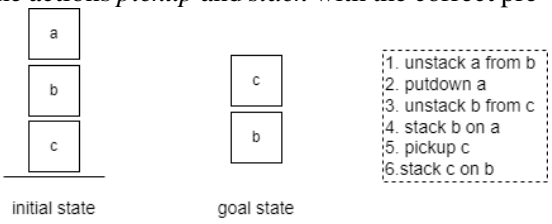

Figure 2: *Getting c on b*

Example 2 (Difference in Unstack Must be Removed)

We will now present our formalization for solving MRP that resembles the dialogue described above. We refer to a message sending from the robot to the human as a *proposal* and a message sending from the human to the robot as a *response*. For our definitions, we will need some notations. Let $M = (I,G,D)$ be a planning problem represented as a set of ASP facts as described in the previous section. Let $add(M) = \{add(x) \mid x \in M\}$ and $remove(M)$ be the set of atoms of the form $delete(x)$ where x is a possible action, a precondition, a postcondition, an initial state atom, or a goal atom. We define an explanation as follows.

Definition 1 *Given a planning problem M, $\varepsilon \subseteq add(M) \cup remove(M)$ is called an* explanation *with respect to M if there exists no x such that $add(x) \in \varepsilon$ and $remove(x) \in \varepsilon$.*

Intuitively, an explanation with respect to a planning problem M is given to someone else and consists of elements that should be added/removed with respect to the planning model of the receiver. Given a planning problem M and an explanation ε (w.r.t. M'), we denote with $M \otimes \varepsilon = M \setminus \{x \mid remove(x) \in \varepsilon\} \cup \{x \mid add(x) \in \varepsilon\}$.

Definition 2 *A* proposal *with respect to a planning problem M is a pair (π, ε) where π is an optimal solution of M and ε is an explanation with respect to M.*

Intuitively, each proposal consists of an optimal plan and an explanation[3]. In the dialogue between the robot and the human in Example 1, the first message of the robot to the human is $(\pi^*, \emptyset)$; the second one is the pair $(\pi^*, \{add(pre(stack(b,a), holding(b), value(holding(b), true)))\})$; etc.

Given a proposal from the robot, the human has different options to respond. We need an extra notation for this purpose. Let M be a planning problem. Let $\Omega(M)$ be the set of atoms of the form $why(a,t)$ or $why(a,f,v,t)$ where a, f, v, and t is an action, fluent, the truth value of f, and a integer ($\leq n$), respectively. Intuitively, $why(a,t)$ indicates that action a is not available; and $why(a,f,v,t)$ indicates that action a cannot be executed at step t because f has the value $\neg v$ at this time step. We assume that *goal* is not an action in M and use it as a special action that needs to occur for the goal to be true. The notion of a response is defined next.

Definition 3 *Given a proposal (π, ε), a* response *to (π, ε) with respect to a planning problem M is one of the following*

- $(\top, \top)$*, denoting* acceptable *and indicating that $M \otimes \varepsilon$ has π as one of its optimal plans;*
- $(\bot, \varepsilon')$*, denoting* inapplicable *and indicating that some information in ε is redundant for updating M and $\varepsilon' = (add(M) \cap \varepsilon) \cup \{remove(x) \mid remove(x) \in \varepsilon, x \notin M\}$; or*
- $(*, \omega)$*, denoting that π is* not executable *with respect to $M \otimes \varepsilon$ and $\omega \subseteq \Omega(M)$ is a set of atoms explaining why π is not executable in $M \otimes \varepsilon$; or*
- $(\pi', \top)$ *where π' is an optimal plan of $M \otimes \varepsilon$ and π' has smaller cost than π.*

Intuitively, a response is constructed given a proposal ε developed with respect to a planning problem M' (i.e., $M' = M_R$) and a planning problem M (i.e., $M = M_H$). It is acceptable if the updated planning problem $M \otimes \varepsilon$ has π as one of its optimal plans. It is inapplicable if ε is *contradictory* with M, i.e., ε contains some $add(x)$ (or $remove(x)$), indicating that x is missing (or redundant) in M but x is indeed in M (or not in M). The third option indicates that π is invalid in $M \otimes \varepsilon$. In this case, ω provides the robot with the reasons why the human believes that π is not executable. The fourth possible response from the human given a proposal (π, ε) indicates that the human still has a better plan than π after integrating ε into her model. The next example illustrates the third possibility.

Example 3 (Plan in Proposal is Not Executable) For the planning problem given Figure 2, assume that the human model differs from the robot model in that it does not contain the precondition *handempty* and the postcondition *clear* for the action *unstack*.

We can easily verify that the plan of the robot $\pi = \langle unstack(a,b), putdown(a), unstack(b,c), stack(b,a), pickup(c), stack(c,b)\rangle$ is not executable in the human model because the missing of the post-condition $clear(b)$ of the action $unstack(a,b)$ implies that $clear(b)$ is false after the execution of the sequence $\langle unstack(a,b), putdown(a)\rangle$ (by inertia). Therefore, $unstack(b,c)$ is not executable that implies that π is not executable in the human model. In this case, the human will inform the robot that the plan is not executable by responding with $(*, \{why(unstack(b,c), clear(b), true, 3)\})$.

[3]The first element of the proposal might be unnecessary under the assumption that the planning problem is fixed for the robot. Including the plan will provide the flexibility for the application of the algorithms developed in the next section, e.g., when the initial state or goal are different. We therefore include the plan as a part of an explanation.

Algorithm 1 Human Dialogue Controller

Input: Human problem M_h

1: $STATUS =$ Running
2: **while** $STATUS =$ Running **do**
3: Waiting for a proposal $p = (\pi, \varepsilon)$
4: **if** $p = \perp$ **then**
5: $STATUS = FAIL$
6: **end if**
7: $\varepsilon' =$ CheckApplicable(M_h, ε)
8: **if** $\varepsilon' \neq \emptyset$ **then**
9: Send$(\perp, \varepsilon')$
10: **else**
11: $\widehat{M_h} = M_h \otimes \varepsilon$
12: $\omega =$ ValidatePlan$(\widehat{M_h}, \pi)$
13: **if** $\omega \neq \emptyset$ **then**
14: Send$(*, \omega)$
15: **else**
16: $\pi_h =$ Solve$(\widehat{M_h}, \pi)$
17: **if** $length(\pi_h) < length(\pi)$ **then**
18: Send$(\pi_h, \top)$
19: **else**
20: Send$(\top, \top)$
21: $STATUS =$ Finish
22: **end if**
23: **end if**
24: **end if**
25: **end while**

Definition 4 (Dialogue) *Given an MRP (π^*, M_R, M_H), a dialogue between the robot and the human is a sequence of rounds $\langle r_1, \ldots, r_n \rangle$ where, for each $1 \leq i \leq n$, $r_i = (x_i, y_i)$ and x_i is a proposal with respect to M_r and y_i is a response to x_i with respect to M_H.*

$\langle r_1, \ldots, r_n \rangle$ is a successful *dialogue if $y_n = (\top, \top)$.*

3.2 Computing Explanations

We will now present the algorithms for the robot and human to work together in solving MRPs. We assume that the robot and the human are communicating via message passing.

3.2.1 Human Dialogue Controller.

Algorithm 1 presents the dialogue controller for the human. The human is actively waiting for a proposal from the robot (Line 3) and reacts to it following Definition 1 in computing her responses. The algorithm stops when (*i*) the robot sends $\perp$, indicating that it runs out of suggestion, which means that the problem has no solution, i.e., none of the proposals is acceptable to the human; or (*ii*) the human receives a proposal that is acceptable to her (Line 20). The steps in the algorithm are simple and self-explanatory

so we omit the details for brevity. We next discuss the procedures that are used in the algorithm.

- `CheckApplicable`(M,ε): this procedure computes $\varepsilon' = (add(M) \cap \varepsilon) \cup \{remove(x) \mid remove(x) \in \varepsilon, x \notin M\}$ (this code is simple and we therefore omitted it).
- `ValidatePlan`(M,π): this program checks for the executability condition of the plan π given the planning problem M. This is done using ASP and the code is included in Listing 1. In this program, $true(x)$ denotes that x is an element in M (e.g., $true(action(a))$ indicates an action in M). Given a plan π consisting of a set of atoms of the form $occurs(a,i)$, the plan is not executable if (*i*) an action in the plan does not exist in M (Lines 14); (*ii*) an action is not executable (Lines 15–16); or (*iii*) some goal is not satisfied at the end of the plan (Lines 17–18). Note that an action is not executable if one of its preconditions does not hold at the time the action is executed (Lines 10–13).

Listing 1: ValidatePlan with parameter M and π

```
1  maxTime(N+1) :- N = #max{T:occurs(_,T)}.
2  time(1..N-1) :- maxTime(N).
3  h(X,1) :- initialState(X,value(X,true)).
4  h(X,T+1) :- time(T), true(action(A)), occurs(A,T),
5   true(postcondition(action(A),
6    effect(unconditional),X,value(X,true))).
7  h(X, T+1) :- time(T),true(action(A)), occurs(A,T), h(X,T),
8    not true(postcondition(action(A),effect(unconditional),X,
9           value(X,false))).
10 not_executable(A,T,true,X) :- time(T), true(action(A)),
11  true(precondition(action(A),X,value(X,true))), not h(X,T).
12 not_executable(A,T,false,X) :- time(T), true(action(A)),
13  true(precondition(action(A),X,value(X,false))), h(X,T).
14 why(A,T) :- time(T), occurs(A,T), not true(action(A)).
15 why(A,X,V,T) :- time(T), true(action(A)), occurs(A,T),
16  not_executable(A,T,V,X).
17 why(goal,X,true,N):- maxTime(N),goal(X,value(X,true)),not h(X, N).
18 why(goal,X,false,N):- maxTime(N),goal(X,value(X,false)),h(X, N).
```

3.2.2 Robot Dialogue Controller.

The robot dialogue controller is detailed in Algorithm 2. In order to guide the search for an explanation, Algorithm 2 uses three variables, S, V, and U, to keep track of information sent to/received from the human. S is a priority queue of proposals that have not been considered where (π,ε) has higher priority than (π',ε') if $|\varepsilon| < |\varepsilon'|$. V is a set of proposals that have been considered. U is a set of elements that should not be contained in any proposal which are identified by responses of the form $(\bot,\varepsilon')$ from the human. Initially, $V = U = \emptyset$ and S is initialized with the trivial proposal $(\pi,\emptyset)$ where π is an optimal plan (Lines 2–3). Observe that π is given in the MRP (π^*, M_r, M_h). As such, we might not need to compute the plan π in Line 2.

The robot's main activity is described by a loop (Lines 4–24). In each iteration, it gets a potential proposal from the queue (Line 8), sends to the human (Line 9), waits for a response r (Line 10), and processes the response as follows.

- If $r = (\top,\top)$, by Def. 3 (Case 1), the human accepts the explanation ε and so the dialogue terminates successful (Line 12).

Algorithm 2 Robot Dialogue Controller

Input: Robot problem M_r

1: $S = \emptyset, U = \emptyset, V = \emptyset$, $STATUS =$ Running
2: $\pi =$ Solve(M_r)
3: enqueue$(S, (\pi, \emptyset))$
4: **while** $STATUS =$ Running **do**
5: **if** S is empty **then**
6: $STATUS = FAIL$; Send$(\perp)$
7: **end if**
8: $(\pi, \varepsilon) =$ dequeue(S), $V = V \cup \{(\pi, \varepsilon)\}$
9: Send$((\pi, \varepsilon))$
10: Waiting for a response $r = (x, y)$
11: **if** $x = \top$ **then**
12: $STATUS =$ Finish
13: **else if** $x = \perp$ **then**
14: $U = U \cup \varepsilon'$
15: **if** $(\pi, \varepsilon \setminus \varepsilon') \notin V$ **then**
16: enqueue$(S, (\pi, \varepsilon \setminus \varepsilon'))$
17: **end if**
18: **else if** $r = (*, \omega)$ **then**
19: $S = updateSR(M_r, \omega, \varepsilon, \pi, U, S, V)$
20: **else if** $r = (\pi', \top)$ **then**
21: $\omega =$ ValidatePlan(M_r, π')
22: $S = updateSH(M_r, \omega, \varepsilon, \pi, \pi', U, S, V)$
23: **end if**
24: **end while**

- If $r = (\perp, \varepsilon')$, by Def. 3 (Case 2), ε' should not be contained in the explanation; as such, U is updated (Line 14) and the explanation is dismissed, a potential explanation is then added to S and the dialogue goes into the next round (Lines 15–17).
- If $r = (*, \omega)$, by Def. 3 (Case 3), ω contains information why π is not executable in $M_h \otimes \varepsilon$. The robot will therefore expand ε with information from ω and U, update S using Algorithm 3 and the dialogue goes into the next round (Line 19).
- If $r = (\pi', \top)$, by Def. 3 (Case 4), the robot understands that the human has a plan π' that is better than π. In that case, the robot will identify the reason why π' is not an optimal plan in M_r (Line 21), update the queue S (Line 22) using Algorithm 4, and the dialogue moves to the next round.

We next discuss the two main procedures in Lines 19 and 22 of Algorithm 2.

Algorithm 3: The algorithm is used in Line 19. It computes possible expansions of a proposal (π, ε), whose explanation is not rejected by the human, given the responses from the human thus far, i.e., a set of facts indicating the reason why π is not executable in the human model (ω) and the set of rejected modifications from the human (U).

In this algorithm, C_1 is the set of actions that do not occur in the human model and therefore the atom $why(a, t)$ belong to ω (Line 2–4).

Algorithm 3 updateSR($M_r, \omega, \varepsilon, \pi, U, S, V$)

Input: M_r - planning problem, ω - the reason why π is not executable, π - robot plan, ε - robot's proposal, U - redundant information, S - current queue of proposals, V - set of examined proposals
Output: Updated queue S

1: $C_1 = C_2 = \emptyset$
2: **for** each $why(a,t)$ in ω **do**
3: $\quad C_1 = C_1 \cup \text{AddActions}(M_r, a)$
4: **end for**
5: **for** each $why(a,x,v,t)$ in ω **do**
6: $\quad$ **if** $(x,v) \in pre(M_r,a) \vee a = goal$ **then**
7: $\quad\quad C = \text{AddPosts}(M_r, \pi, x, v, t)$
8: $\quad\quad$ **if** $C \neq \emptyset$ **then**
9: $\quad\quad\quad C_2 = C_2 \cup C$
10: $\quad\quad$ **else**
11: $\quad\quad\quad C_2 = C_2 \cup \text{AddInits}(M_r, x, v) \cup \text{RemoveInit}(M_r, x, \neg v)$
12: $\quad\quad$ **end if**
13: $\quad$ **else**
14: $\quad\quad C_2 = C_2 \cup \text{RemovePre}(a, x, v)$
15: $\quad$ **end if**
16: **end for**
17: **for** each non empty subset C of $C_2 \setminus U$ **do**
18: $\quad$ **if** $(\pi, \varepsilon \cup C_1 \cup C) \notin V$ **then**
19: $\quad\quad$ `enqueue`$(S, (\pi, \varepsilon \cup C_1 \cup C))$
20: $\quad$ **end if**
21: **end for**
22: **return** S

C_2 is the set of missing postconditions that need to be added or redundant preconditions that need to be removed. When a precondition is in the human model and not a part of the robot model and it is required for the action to be executable then it needs to be removed. When a precondition is presented in both models and the action is not executable, it means that some action that is supposed to enable this precondition does not contain this postcondition in the human model. Therefore, a postcondition needs to be added.

The loop (Lines 5–16) is applied to atom of the form $why(a,x,v,t)$ in ω. For each element, C_2 is updated as follows. For simplicity of the presentation, we write (x,v) is a precondition (resp. a postcondition) of an action a and mean that *pre(action(a), x, value(x,v))* (resp. *post(action(a), effect(unconditional), x, value(x,v))* is an element of the domain model. $pre(M_r, a)$ is the collection of pairs of the form (x,v) that are preconditions of a.

- If $why(a,x,v,t) \in \omega$ then (x,v) is a precondition of a in the human model (computed by the code in Listing 1). $(x,v) \in pre(M_r,a)$ (Line 6, $a \neq goal$) indicates that (x,v) is also a precondition of a in the robot model. Therefore, to make π executable model in the human model, some action in π that occurs before a, whose index is less than t, must have a postcondition (x,v). This is computed using $AddPosts(M_r, \pi, x, v, t)$.
- If $why(goal,x,v,t) \in \omega$ (Line 6) then x is supposed to have the value v in the goal but it is not true after the execution of π. Therefore, to achieve the goal and make π executable in the human model,

some action in π must have a postcondition (x,v). This is computed using $AddPosts(M_r,\pi,x,v,t)$.
- If $why(a,x,v,t) \in \omega$ and $(x,v) \notin pre(M_r,a)$ and $a \neq goal$ then (x,v) is a precondition of a in the human model but it is not a precondition of a in the robot model. Therefore, the precondition should be removed from a (Line 9).

The updated S (Lines 17–21) includes all possible expansions of ε that have not been considered. Each expansion contains the set of missing actions in the human model (C_1) and some potential suggestions from C_2.

The function $AddPosts(M_r,\pi,x,v,t)$ collects actions that occurs before step t that have (x,v) as one of its postconditions and returns this set. The function $RemovePre(a,x,v)$ returns the precondition that should be removed from a. When some condition must be true and cannot be established by adding some postcondition or removing some precondition, it can be achieved by adding it to the initial state ($AddInits(M_r,x,v)$, Line 11) or removing its contradiction from the initial state ($RemoveInits(M_r,x,\neg v)$, Line 11). The codes for these procedures are fairly simple and are omitted for brevity.

Algorithm 4: This algorithm has similar functions as Algorithm 3 and is used in Line 22 of Algorithm 2. The key difference between the two algorithms lies in an additional input, the plan π', and therefore, ω. In Algorithm 4, π' is a plan executable in the human model and has cost smaller than the optimal plan in the robot domain. ω is the reason why π' is not executable in the robot model that is computed using the code in Listing 1 with M_r and π' as inputs. As in the previous algorithm, two sets of changes, C_1 and C_2, are computed and used to updated the queue of proposals.

C_1 is the set of actions that are not presented in the robot model but occur in the plan from the human model. So, C_1 is the set of actions that should be removed (Lines 2–4).

C_2 is the set of changes that prevent π to be executable in the human model given the robot's understanding about the human model. The difference between the construction in this algorithm with Algorithm 4 is that the robot does not really know what is present in the human model. Therefore, for each $why(a,x,v,t) \in \omega$, the robot must suggest one of the possibilities:
- adding the precondition (x,v) for a (Line 8 and Line 10) because this precondition is in the robot model; or
- adding the postcondition $(x,\neg v)$ for some action that occurs before a in π (for the case that (x,v) is already a precondition of a in the human model); or
- removing the postcondition (x,v) for some action that occurs before a in π; or
- removing some initial state information.

3.3 Properties of Algorithms

It is easy to see that if Algorithms 1–2 terminate with $STATUS = \text{Finish}$ then the two sides agree on a minimal explanation because of the priority in the queue S in Algorithm 1. It remains to be proven that under the assumption that M_R has a solution then the algorithms will terminate with $STATUS = \text{Finish}$. This can be achieved using the following observations:
- Algorithm 2 terminates with $STATUS = \text{FAIL}$ only if the queue $S = \emptyset$.
- Given that (π,ε) is a proposal sent by the robot to the human and the response from the human is either $(*,\omega')$ or $(\pi',\top)$ then Algorithm 3 or 4 adds at least one proposal (π,τ) to S such that $\varepsilon \subset \tau$ and for every $add(x) \in \tau$ (resp. $remove(x)$), $x \in M_H$ (resp. $x \notin M_H$);
- the response from the human for the first element of S, (π,ε), is of the form $(*,\omega')$ or $(\pi',\top)$.

Algorithm 4 updateSH($M_r, \omega, \varepsilon, \pi, \pi', U, S, V$)

Input: Similar to Algorithm 3 excepts π' - a plan from the human model
Output: Updated queue S

$C_1 = C_2 = \emptyset$
for each $why(a,t)$ in ω **do**
  $C_1 = C_1 \cup \text{RemoveAction}(a)$
end for
for each $why(a,x,v,t)$ in ω **do**
  $C = \text{AddPosts}(M_r, \pi', x, \neg v, t) \cup \text{RemovePost}(M_r, \pi', x, v, t)$
  if $C \neq \emptyset$ **then**
    $C_2 = C_2 \cup \text{AddPres}(M_r, a, x, v) \cup C$
  else
    $C_2 = C_2 \cup \text{AddPres}(M_r, a, x, v) \cup \text{RemoveInit}(M_r, x, v)$
  end if
end for
for each non empty subset C of $C_2 \setminus U$ **do**
  if $(\pi, \varepsilon \cup C_1 \cup C) \notin V$ **then**
    `enqueue`$(S, (\pi, \varepsilon \cup C_1 \cup C))$
  end if
end for
return S

4 Experimental Evaluation

We implement the algorithms described in the previous section and empirically validate the system using the benchmarks that have been used in comparing different systems that solve MRPs. Observe that because we solve MRPs under the assumption that the robot does not know the human model, a comparison on the efficiency between our system and others would not be a fair one. Nevertheless, a comparison between the output would be a validation of our approach.

We run our system using the same benchmarks used by [1]: the three planning domains *LOGISTICS, BLOCKSWORLD* and *ROVER* with the following setting. We use the problem instances as given, the robot model D_r, and create D_h from D_r by **(1)** removing one random precondition for every action in D_r; **(2)** removing one random postcondition for every action in D_r; **(3)** removing one random precondition and one random postcondition for every action in D_r; **(4)** removing all but one random precondition and one random postcondition from two actions in D_r; **(5)** removing some random actions that used in the robot plan; and **(6)** modifying initial states. Exceptions are made when an action has only one precondition/postcondition.

Table 4 tabulates the optimal plan lengths π, explanation lengths $|\varepsilon|$ and run-time in seconds. In our proposed framework, r, T, t is number of rounds, the total running time, and the running time excluding the planning time in both agents, respectively. For the two other framework, we record T_C - the total runtime of the framework C (the system by [1]) and T_N (the system by [9]). Observe that we do not experiment with the system by [14] as this system is similar to the system C (or N), just faster.

Examining the outputs confirms that our system is correct as it returns the same answers returned by the system N [9] or the system C [1]. It is interesting to observe that the number of rounds required for the robot to find a solution (r) is not proportional to the size of the explanation ($|\varepsilon|$) (see, e.g., Mod. 1.

Problem		$\|\pi\|$	Mod. 1						Mod. 2						Mod. 3					
			$\|\varepsilon\|$	r	T	t	T_C	T_N	$\|\varepsilon\|$	r	T	t	T_C	T_N	$\|\varepsilon\|$	r	T	t	T_C	T_N
BLOCKS-WORLD	*i*	16	3	4	155	2	6	88	4	6	111	3	0.3	34	7	6	245	6	44	37
	ii	20	3	3	58	1	43	81	3	4	27	3	0.5	21	7	6	33	5	31	27
	iii	18	3	4	348	2	3	290	4	5	394	4	0.5	222	7	5	239	5	30	223
	iv	20	3	3	360	2	13	221	3	2	301	3	0.7	280	8	6	4124	4	23	310
LOGIS-TICS	*v*	20	5	5	145	3	19	117	3	2	256	5	0.6	113	10	6	149	8	459	145
	vi	17	5	4	161	3	21	103	4	4	133	4	5	99	10	5	216	7	399	110
	vii	14	5	4	20	2	20	1397	4	4	22	4	3	1167	10	6	28	6	397	1173
	viii	24	5	4	828	2	23	18	4	4	1287	6	0.7	17	10	6	805	10	468	20
ROVER	*ix*	10	5	4	7	2	46	7	5	4	14	8	6	5	5	3	10	4	570	6
	x	8	2	2	2	1	2	3	6	2	6	3	7	3	4	5	5	3	374	4
	xi	11	6	3	15	5	49	11	6	4	26	16	11	15	8	6	30	13	569	40
	xii	8	4	3	10	5	3	2	4	4	16	8	4	11	5	4	24	12	61	17

Problem		$\|\pi\|$	Mod. 4						Mod. 5					Mod. 6				
			$\|\varepsilon\|$	r	T	t	T_C	T_N	$\|\varepsilon\|$	r	T	t	T_N	$\|\varepsilon\|$	r	T	t	T_N
BLOCKS-WORLD	*i*	16	11	7	606	12	452	81	9	1	69	5	47	2	1	3	1	1
	ii	20	9	6	27	6	437	93	20	3	29	10	43	4	3	5	3	1
	iii	18	8	5	107	9	652	277	18	3	218	12	45	5	3	7	2	3
	iv	20	8	6	344	8	473	213	26	2	418	8	46	5	2	41	19	99
LOGIS-TICS	*v*	20	4	3	160	2	16	137	4	3	135	8	56	5	2	237	57	112
	vi	17	4	4	234	3	15	103	9	3	117	10	57	4	3	221	97	124
	vii	14	4	3	19	2	14	1213	19	3	24	9	59	7	3	278	49	121
	viii	24	4	3	953	2	14	21	16	3	670	13	59	6	2	283	102	132
ROVER	*ix*	10	8	4	18	6	589	7	41	3	25	23	323	4	2	10	23	3
	x	8	9	5	15	6	1513	3	35	3	14	13	354	4	2	8	13	1
	xi	11	11	6	48	18	1516	14	21	3	61	53	1498	6	3	71	53	15
	xii	8	8	3	42	12	39	9	29	3	66	62	2537	10	4	56	62	18

Table 1: Varying Modifications and Domains

ROVER, *iv* vs. *xi*). As expected, our system is slower than other systems. As it can be seen, the major time spending in our system is for planning since several calls to the planner (Solve) need to be made during the course of a dialogue.

5 Conclusions, Discussion, and Future Work

We propose a dialogue-based framework for solving MRPs that differs from earlier approaches to solving MRPs [1, 9, 14] in that our framework does not assume that the robot, who needs to solve the MRPs, knows the human model. We develop the notions of a proposal/response from the robot/human and algorithms for computing proposals/responses as well as for controlling of the dialogue between the robot and the human. We implement the proposed algorithms in a distributed setting and validate the usability of our system using benchmarks from the literature. Our experiments show that the proposed system computes correct answers but is not as efficient as state-of-the-art MRP solvers. Improving the performance of the system by identifying better strategy for selecting proposal (e.g., aiming at a faster convergence of the dialogue) than the current brute-force strategy could be a worthy undertaking that we leave for the future.

To the best of our knowledge, the work that is most closely to our framework is proposed by [6]. In this work, a method for generating questions to refine the robot's understanding of the human model is proposed. The interaction between the agents is also dialog-based but the content of the dialog is different. In each interaction, the robot provides a tuple (I, G, π), the initial state, goal state and a plan, and the answer from the human is simply *yes*/*no*. As such, [6] focuses on identifying sequence of questions that help the robot to learn the human model. The work of [13] also removes the assumption that the robot knows the human model but employs reinforcement learning to solve the problem.

Finally, we note that the use of ASP as the representation language for planning in solving MRPs implies that the proposed framework can be used for computing solutions of the Model Reconciliation

in Logic Programs (MRLP) problems proposed by [11].

Acknowledgement

The authors have been partially supported by NSF grants 1914635, 1757207, and 1812628.

References

[1] Tathagata Chakraborti, Sarath Sreedharan, Yu Zhang & Subbarao Kambhampati (2017): *Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy*. In Carles Sierra, editor: *IJCAI*, ijcai.org, pp. 156–163, doi:10.24963/ijcai.2017/23.

[2] Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer & Francesco Scarcello (1998): *The KR System dlv: Progress Report, Comparisons and Benchmarks*. In Anthony G. Cohn, Lenhart K. Schubert & Stuart C. Shapiro, editors: *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998*, Morgan Kaufmann, pp. 406–417.

[3] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2014): *Clingo = ASP + Control: Preliminary Report*. *CoRR* abs/1405.3694, doi:10.48550/ARXIV.1405.3694. Available at `https://arxiv.org/abs/1405.3694`.

[4] M. Gelfond & V. Lifschitz (1991): *Classical negation in logic programs and disjunctive databases*. *New Generation Computing*, pp. 365–387, doi:10.1007/BF03037169.

[5] Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld & David Wilkins (1998): *PDDL – the planning domain definition language*. Technical Report TR-98-003, Yale CCVC.

[6] Sachin Grover, David E. Smith & Subbarao Kambhampati (2020): *Model Elicitation through Direct Questioning*. *CoRR* abs/2011.12262. arXiv:2011.12262.

[7] V. Lifschitz (2002): *Answer set programming and plan generation*. *Artificial Intelligence* 138(1–2), pp. 39–54, doi:10.1016/S0004-3702(02)00186-8.

[8] V. Marek & M. Truszczyński (1999): *Stable models and an alternative logic programming paradigm*. In: *The Logic Programming Paradigm: a 25-year Perspective*, pp. 375–398, doi:10.1007/978-3-642-60085-2_17.

[9] Van Nguyen, Stylianos Loukas Vasileiou, Tran Cao Son & William Yeoh (2020): *Explainable Planning Using Answer Set Programming*. In: *KRR*, pp. 662–666.

[10] I. Niemelä (1999): *Logic programming with stable model semantics as a constraint programming paradigm*. *Annals of Mathematics and Artificial Intelligence* 25(3,4), pp. 241–273, doi:10.1023/A:1018930122475.

[11] Tran Cao Son, Van Nguyen, Stylianos Loukas Vasileiou & William Yeoh (2021): *Model Reconciliation in Logic Programs*. In Wolfgang Faber, Gerhard Friedrich, Martin Gebser & Michael Morak, editors: *JELIA*, *Lecture Notes in Computer Science* 12678, Springer, pp. 393–406, doi:10.1007/978-3-030-75775-5_26.

[12] Sarath Sreedharan, Tathagata Chakraborti & Subbarao Kambhampati (2021): *Foundations of explanations as model reconciliation*. *Artif. Intell.* 301, p. 103558, doi:10.1016/j.artint.2021.103558.

[13] Sarath Sreedharan, Alberto Olmo Hernandez, Aditya Prasad Mishra & Subbarao Kambhampati (2019): *Model-Free Model Reconciliation*. In Sarit Kraus, editor: *IJCAI*, ijcai.org, pp. 587–594, doi:10.24963/ijcai.2019/83.

[14] Stylianos Loukas Vasileiou, Alessandro Previti & William Yeoh (2021): *On Exploiting Hitting Sets for Model Reconciliation*. In: *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, AAAI Press, pp. 6514–6521. Available at `https://ojs.aaai.org/index.php/AAAI/article/view/16807`.

6 Appendix 1: Algorithms

This appendix contains the descriptions of some algorithms that are referred to in the paper but are omitted due to lack of space. Algorithm 5 identifies potential postconditions of actions in the plan π that occur before step t and assigns the variable x the truth value v. Note that in our representation, actions and fluents are predicates. As such, if a postcondition is added to an action a then it is reasonable to add a similar postcondition to an action that is unifiable with a. Line 5 of Algorithm 5 ensures that this is done.

Algorithm 5 addPosts(M_r, π, x, v, t)

Input: M_r - planning problem, π - a plan, a - action, x - fluent, v - truth value of the fluent, t - a time step
Output: ε - set of elements to be added

1: $\varepsilon = \emptyset$
2: **for** $i = t-1$ to 1 **do**
3: $a = \pi[i]$
4: **for** each $p = postcondition(a', \text{effect}, x', v') \in M_r$ **do**
5: **if** $sameName(a, a')$ and $sameName(x, x')$ and $v = v'$ **then**
6: $\varepsilon = \varepsilon \cup \{add(p)\}$
7: **end if**
8: **end for**
9: **end for**
10: **return** ε

Algorithm 6 checks whether the variable x has the value v in the initial state of M_r and returns the set of elements that need to be added to the initial state (of the human).

Algorithm 6 addInit(M_r, x, v)

Input: M_r - planning problem, x - fluent, v - truth value of the fluent
Output: ε - initial state modification

1: $\varepsilon = \emptyset$
2: **if** $init(x, v) \in M_r$ **then**
3: $\varepsilon = \varepsilon \cup \{add(init(x, v))\}$
4: **end if**
5: **return** ε

Algorithm 7 checks whether the action a has (x, v) as one of its precondition in M_r and returns the set of preconditions need to be added to the human model. This algorithm also consider actions that are unifiable with a as in Algorithm 5.

Algorithm 8 identifies postconditions of the form (x, v) of actions in the plan π that occur before step t that are not in the robot model, and thus, have to be removed.

Algorithm 9 checks whether the variable x has the value v in the initial state of M_r and returns the set of elements that need to be removed from the initial state (of the human model).

Algorithm 10 returns the set of precondition of a that needs to be removed from the human model.

Algorithm 11 returns true if two terms are unifiable and false otherwise.

Algorithm 7 addPres(M_r, a, x, v)

Input: M_r - planning problem, a - an action, x - a fluent, v - truth value of the fluent
Output: ε - set of preconditions that should be added

$\varepsilon = \emptyset$
for each $p = precondition(a', x', v') \in M_r$ **do**
  if $sameName(a, a')$ and $sameName(x, x')$ and $v = v'$ **then**
    $\varepsilon = \varepsilon \cup \{add(p)\}$
  end if
end for
return ε

Algorithm 8 removePost(M_r, π, x, v, t)

Input: M_r - planning problem, π - a plan, x - a fluent, v - truth value of the fluent, t - a time step
Output: ε - set of postconditions to be removed

$\varepsilon = \emptyset$
for $i = t - 1$ to 1 **do**
  $a = \pi[i]$
  $p = postcondition(a, \text{effect}, x, v)$
  if $p \notin M_r$ **then**
    $\varepsilon = \varepsilon \cup \{remove(p)\}$
  end if
end for
return ε

7 Appendix 2: Blocksworld Domain - PDDL and ASP Encoding

Listing 2 is the PDDL encoding of the Blocksworld domain used in the paper and the experiment.

Listing 2: Robot Domain in PDDL

```
(define (domain BLOCKS)
  (:requirements :strips :typing)
  (:types block)
  (:predicates (on ?x-block ?y-block)
   (ontable ?x-block) (clear ?x-block)
   (handempty) (holding ?x-block)
  )
  (:action pickup
   :parameters (?x-block)
   :precondition(and (clear ?x)
     (ontable ?x) (handempty))
   :effect (and (not (ontable ?x))
        (not (clear ?x))
        (not (handempty)) (holding ?x)))
  (:action putdown
   :parameters (?x-block)
   :precondition (holding ?x)
   :effect  (and (not (holding ?x))
```

Algorithm 9 removeInit($M_r, x, v,$)

Input: M_r - planning problem, x - a fluent, v - truth value of the fluent
Output: The explanation ε

1: $\varepsilon = \emptyset$
2: **if** $i = init(x,v) \notin M_r$ **then**
3: $\quad \varepsilon = \varepsilon \cup \{remove(i)\}$
4: **end if**
5: **return** ε

Algorithm 10 removePre(a, x, v)

Input: a - an action, x - a fluent, v - truth value of the fluent
Output: ε – set of preconditions to be removed

1: **return** $\{remove(precondition(a,x,v))\}$

```
19      (clear ?x) (handempty) (ontable ?x)))
20    (:action stack
21     :parameters (?x-block ?y-block)
22     :precondition (and (holding ?x) (clear ?y))
23     :effect (and (not (holding ?x))
24          (not (clear ?y)) (clear ?x)
25          (handempty) (on ?x ?y)))
26    (:action unstack
27     :parameters (?x-block ?y-block)
28     :precondition (and (on ?x ?y)
29      (clear ?x) (handempty))
30     :effect (and (holding ?x) (clear ?y)
31      (not (clear ?x)) (not (handempty))
32          (not (on ?x ?y)))))
```

The next listing represents the set of facts encoding a planning problem in the Blocksworld domain for the robot in Figure 1.

Listing 3: The Robot Planning Problem as ASP facts

```
1  constant(c("b")).
2  constant(c("a")).

4  has(c("b"),type("block")).
5  has(c("a"),type("block")).

7  initialState(var("handempty"),value(var("handempty"),true)).
8  initialState(var(("clear",c("a"))),
9       value(var(("clear",c("a"))),true)).
10 initialState(var(("on",c("a"),c("b"))),
11      value(var(("on",c("a"),c("b"))),true)).
12 initialState(var(("ontable",c("b"))),
13      value(var(("ontable",c("b"))),true)).

15 action(action(("unstack",c("b"),c("b")))).
16 pre(action(("unstack",c("b"),c("b"))),var("handempty"),
```

Algorithm 11 sameName(t_1, t_2)

Input: t_1, t_2 are terms presenting actions or fluents
Output: true/false

1: $s_1 = getSignature(t_1)$
2: $s_2 = getSignature(t_2)$
3: **return** $s_1 = s_2$

```
17        value(var("handempty"),true)).
18  pre(action(("unstack",c("b"),c("b"))),var(("clear",c("b"))),
19        value(var(("clear",c("b"))),true)).
20  pre(action(("unstack",c("b"),c("b"))),var(("on",c("b"),c("b"))),
21        value(var(("on",c("b"),c("b"))),true)).
22  post(action(("unstack",c("b"),c("b"))),
23        effect(unconditional),var(("on",c("b"),c("b"))),
24        value(var(("on",c("b"),c("b"))),false)).
25  post(action(("unstack",c("b"),c("b"))),
26        effect(unconditional),var(("clear",c("b"))),
27        value(var(("clear",c("b"))),false)).
28  post(action(("unstack",c("b"),c("b"))),
29        effect(unconditional),var("handempty"),
30        value(var("handempty"),false)).
31  post(action(("unstack",c("b"),c("b"))),
32        effect(unconditional),var(("clear",c("b"))),
33        value(var(("clear",c("b"))),true)).
34  post(action(("unstack",c("b"),c("b"))),
35        effect(unconditional),var(("holding",c("b"))),
36        value(var(("holding",c("b"))),true)).


39  action(action(("unstack",c("a"),c("b")))).
40  pre(action(("unstack",c("a"),c("b"))),var("handempty"),
41        value(var("handempty"),true)).
42  pre(action(("unstack",c("a"),c("b"))),var(("clear",c("a"))),
43        value(var(("clear",c("a"))),true)).
44  pre(action(("unstack",c("a"),c("b"))),var(("on",c("a"),c("b"))),
45        value(var(("on",c("a"),c("b"))),true)).
46  post(action(("unstack",c("a"),c("b"))),
47        effect(unconditional),var(("on",c("a"),c("b"))),
48        value(var(("on",c("a"),c("b"))),false)).
49  post(action(("unstack",c("a"),c("b"))),
50        effect(unconditional),var("handempty"),
51        value(var("handempty"),false)).
52  post(action(("unstack",c("a"),c("b"))),
53        effect(unconditional),var(("clear",c("a"))),
54        value(var(("clear",c("a"))),false)).
55  post(action(("unstack",c("a"),c("b"))),
56        effect(unconditional),var(("clear",c("b"))),
57        value(var(("clear",c("b"))),true)).
58  post(action(("unstack",c("a"),c("b"))),
59        effect(unconditional),var(("holding",c("a"))),
```

```
60      value(var(("holding",c("a"))),true)).

62  action(action(("unstack",c("b"),c("a")))).
63  pre(action(("unstack",c("b"),c("a"))),var("handempty"),
64      value(var("handempty"),true)).
65  pre(action(("unstack",c("b"),c("a"))),var(("clear",c("b"))),
66      value(var(("clear",c("b"))),true)).
67  pre(action(("unstack",c("b"),c("a"))),var(("on",c("b"),c("a"))),
68      value(var(("on",c("b"),c("a"))),true)).
69  post(action(("unstack",c("b"),c("a"))),
70      effect(unconditional),var(("on",c("b"),c("a"))),
71      value(var(("on",c("b"),c("a"))),false)).
72  post(action(("unstack",c("b"),c("a"))),
73      effect(unconditional),var("handempty"),
74      value(var("handempty"),false)).
75  post(action(("unstack",c("b"),c("a"))),
76      effect(unconditional),var(("clear",c("a"))),
77      value(var(("clear",c("a"))),true)).
78  post(action(("unstack",c("b"),c("a"))),
79      effect(unconditional),var(("clear",c("b"))),
80      value(var(("clear",c("b"))),false)).
81  post(action(("unstack",c("b"),c("a"))),
82      effect(unconditional),var(("holding",c("b"))),
83      value(var(("holding",c("b"))),true)).

85  action(action(("unstack",c("a"),c("a")))).
86  pre(action(("unstack",c("a"),c("a"))),var("handempty"),
87      value(var("handempty"),true)).
88  pre(action(("unstack",c("a"),c("a"))),var(("clear",c("a"))),
89      value(var(("clear",c("a"))),true)).
90  pre(action(("unstack",c("a"),c("a"))),var(("on",c("a"),c("a"))),
91      value(var(("on",c("a"),c("a"))),true)).
92  post(action(("unstack",c("a"),c("a"))),
93      effect(unconditional),var(("on",c("a"),c("a"))),
94      value(var(("on",c("a"),c("a"))),false)).
95  post(action(("unstack",c("a"),c("a"))),
96      effect(unconditional),var("handempty"),
97      value(var("handempty"),false)).
98  post(action(("unstack",c("a"),c("a"))),
99      effect(unconditional),var(("clear",c("a"))),
100     value(var(("clear",c("a"))),false)).
101 post(action(("unstack",c("a"),c("a"))),
102     effect(unconditional),var(("clear",c("a"))),
103     value(var(("clear",c("a"))),true)).
104 post(action(("unstack",c("a"),c("a"))),
105     effect(unconditional),var(("holding",c("a"))),
106     value(var(("holding",c("a"))),true)).

108 action(action(("stack",c("b"),c("b")))).
109 pre(action(("stack",c("b"),c("b"))),var(("clear",c("b"))),
110     value(var(("clear",c("b"))),true)).
111 pre(action(("stack",c("b"),c("b"))),var(("holding",c("b"))),
```

```
112       value(var(("holding",c("b"))),true)).
113  post(action(("stack",c("b"),c("b"))),
114       effect(unconditional),var(("on",c("b"),c("b"))),
115       value(var(("on",c("b"),c("b"))),true)).
116  post(action(("stack",c("b"),c("b"))),
117       effect(unconditional),var("handempty"),
118       value(var("handempty"),true)).
119  post(action(("stack",c("b"),c("b"))),
120       effect(unconditional),var(("clear",c("b"))),
121       value(var(("clear",c("b"))),true)).
122  post(action(("stack",c("b"),c("b"))),
123       effect(unconditional),var(("clear",c("b"))),
124       value(var(("clear",c("b"))),false)).
125  post(action(("stack",c("b"),c("b"))),
126       effect(unconditional),var(("holding",c("b"))),
127       value(var(("holding",c("b"))),false)).

129  action(action(("stack",c("a"),c("b")))).
130  pre(action(("stack",c("a"),c("b"))),var(("clear",c("b"))),
131       value(var(("clear",c("b"))),true)).
132  pre(action(("stack",c("a"),c("b"))),var(("holding",c("a"))),
133       value(var(("holding",c("a"))),true)).
134  post(action(("stack",c("a"),c("b"))),
135       effect(unconditional),var(("on",c("a"),c("b"))),
136       value(var(("on",c("a"),c("b"))),true)).
137  post(action(("stack",c("a"),c("b"))),
138       effect(unconditional),var("handempty"),
139       value(var("handempty"),true)).
140  post(action(("stack",c("a"),c("b"))),
141       effect(unconditional),var(("clear",c("a"))),
142       value(var(("clear",c("a"))),true)).
143  post(action(("stack",c("a"),c("b"))),
144       effect(unconditional),var(("clear",c("b"))),
145       value(var(("clear",c("b"))),false)).
146  post(action(("stack",c("a"),c("b"))),
147       effect(unconditional),var(("holding",c("a"))),
148       value(var(("holding",c("a"))),false)).


151  action(action(("stack",c("b"),c("a")))).
152  pre(action(("stack",c("b"),c("a"))),var(("clear",c("a"))),
153       value(var(("clear",c("a"))),true)).
154  pre(action(("stack",c("b"),c("a"))),var(("holding",c("b"))),
155       value(var(("holding",c("b"))),true)).
156  post(action(("stack",c("b"),c("a"))),
157       effect(unconditional),var(("on",c("b"),c("a"))),
158       value(var(("on",c("b"),c("a"))),true)).
159  post(action(("stack",c("b"),c("a"))),
160       effect(unconditional),var("handempty"),
161       value(var("handempty"),true)).
162  post(action(("stack",c("b"),c("a"))),
163       effect(unconditional),var(("clear",c("a"))),
```

```
164       value(var(("clear",c("a"))),false)).
165  post(action(("stack",c("b"),c("a"))),
166       effect(unconditional),var(("clear",c("b"))),
167       value(var(("clear",c("b"))),true)).
168  post(action(("stack",c("b"),c("a"))),
169       effect(unconditional),var(("holding",c("b"))),
170       value(var(("holding",c("b"))),false)).


173  action(action(("stack",c("a"),c("a")))).
174  pre(action(("stack",c("a"),c("a"))),var(("clear",c("a"))),
175       value(var(("clear",c("a"))),true)).
176  pre(action(("stack",c("a"),c("a"))),var(("holding",c("a"))),
177       value(var(("holding",c("a"))),true)).
178  post(action(("stack",c("a"),c("a"))),
179       effect(unconditional),var(("on",c("a"),c("a"))),
180       value(var(("on",c("a"),c("a"))),true)).
181  post(action(("stack",c("a"),c("a"))),
182       effect(unconditional),var("handempty"),
183       value(var("handempty"),true)).
184  post(action(("stack",c("a"),c("a"))),
185       effect(unconditional),var(("clear",c("a"))),
186       value(var(("clear",c("a"))),true)).
187  post(action(("stack",c("a"),c("a"))),
188       effect(unconditional),var(("clear",c("a"))),
189       value(var(("clear",c("a"))),false)).
190  post(action(("stack",c("a"),c("a"))),
191       effect(unconditional),var(("holding",c("a"))),
192       value(var(("holding",c("a"))),false)).

194  action(action(("putdown",c("b")))).
195  pre(action(("putdown",c("b"))),var(("holding",c("b"))),
196       value(var(("holding",c("b"))),true)).
197  post(action(("putdown",c("b"))),
198       effect(unconditional),var(("ontable",c("b"))),
199       value(var(("ontable",c("b"))),true)).
200  post(action(("putdown",c("b"))),
201       effect(unconditional),var("handempty"),
202       value(var("handempty"),true)).
203  post(action(("putdown",c("b"))),
204       effect(unconditional),var(("clear",c("b"))),
205       value(var(("clear",c("b"))),true)).
206  post(action(("putdown",c("b"))),
207       effect(unconditional),var(("holding",c("b"))),
208       value(var(("holding",c("b"))),false)).

210  action(action(("putdown",c("a")))).
211  pre(action(("putdown",c("a"))),var(("holding",c("a"))),
212       value(var(("holding",c("a"))),true)).
213  post(action(("putdown",c("a"))),
214       effect(unconditional),var(("ontable",c("a"))),
215       value(var(("ontable",c("a"))),true)).
```

```
post(action(("putdown",c("a"))),
     effect(unconditional),var("handempty"),
     value(var("handempty"),true)).
post(action(("putdown",c("a"))),
     effect(unconditional),var(("clear",c("a"))),
     value(var(("clear",c("a"))),true)).
post(action(("putdown",c("a"))),
     effect(unconditional),var(("holding",c("a"))),
     value(var(("holding",c("a"))),false)).

action(action(("pickup",c("b")))).
pre(action(("pickup",c("b"))),var("handempty"),
     value(var("handempty"),true)).
pre(action(("pickup",c("b"))),var(("ontable",c("b"))),
     value(var(("ontable",c("b"))),true)).
pre(action(("pickup",c("b"))),var(("clear",c("b"))),
     value(var(("clear",c("b"))),true)).
post(action(("pickup",c("b")))),
     effect(unconditional),var(("holding",c("b")))),
     value(var(("holding",c("b"))),true)).
post(action(("pickup",c("b")))),
     effect(unconditional),var("handempty"),
     value(var("handempty"),false)).
post(action(("pickup",c("b")))),
     effect(unconditional),var(("clear",c("b")))),
     value(var(("clear",c("b"))),false)).
post(action(("pickup",c("b")))),
     effect(unconditional),var(("ontable",c("b")))),
     value(var(("ontable",c("b"))),false)).

action(action(("pickup",c("a")))).
pre(action(("pickup",c("a"))),var("handempty"),
     value(var("handempty"),true)).
pre(action(("pickup",c("a"))),var(("ontable",c("a")))),
     value(var(("ontable",c("a"))),true)).
pre(action(("pickup",c("a"))),var(("clear",c("a")))),
     value(var(("clear",c("a"))),true)).
post(action(("pickup",c("a")))),
     effect(unconditional),var(("holding",c("a")))),
     value(var(("holding",c("a"))),true)).
post(action(("pickup",c("a")))),
     effect(unconditional),var("handempty"),
     value(var("handempty"),false)).
post(action(("pickup",c("a")))),
     effect(unconditional),var(("clear",c("a")))),
     value(var(("clear",c("a"))),false)).
post(action(("pickup",c("a")))),
     effect(unconditional),var(("ontable",c("a")))),
     value(var(("ontable",c("a"))),false)).alse)).
post(action(("pickup",c("a")))),effect(unconditional),
var(("ontable",c("a")))),value(var(("ontable",c("a")))),false)).
```

```
268 goal(var(("ontable",c("a"))),value(var(("ontable",c("a"))),true)).
269 goal(var(("on",c("b"),c("a"))),value(var(("on",c("b"),c("a"))),true)).
```