Conditional Updates of Answer Set Programming and Its Application in Explainable Planning∗

Extended Abstract

Van Nguyen and Tran Cao Son
New Mexico State University
Las Cruces, NM
{vnguyen,ts@cs.nmsu.edu}

Stylianos Vasileiou and William Yeoh
Washington University in St. Louis
St. Louis, MO
{svasileio,wyeh}@wustl.edu

ABSTRACT
In explainable planning, the planning agent needs to explain its plan to a human user, especially when the plan appears infeasible or suboptimal for the user. A popular approach is called model reconciliation, where the agent reconciles the differences between its model and the model of the user such that its plan is also feasible and optimal to the user. This problem can be viewed as a more general problem as follows: Given two knowledge bases πa and πh and a query q such that πa entails q and πh does not entail q, where the notion of entailment is dependent on the logical theories underlying πa and πh, how to change πh so given πa and the support for q in πa so that πh does entail q. In this paper, we study this problem under the context of answer set programming. To achieve this goal, we (1) define the notion of a conditional update between two logic programs πa and πh with respect to a query q; (2) define the notion of an explanation for a query q from a program πa to a program πh using conditional updates; (3) develop algorithms for computing explanations; and (4) show how the notion of explanation based on conditional updates can be used in explainable planning.

KEYWORDS
Explainable Planning; Answer Set Programming

ACM Reference Format:

1 LOGIC PROGRAMMING
Answer set programming (ASP) [10, 11] is a declarative programming paradigm based on logic programming under the answer set semantics. A logic program Π is a set of rules of the form

\[ a_0 \leftarrow a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \]

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 not \(a_i\) is believed to be true, then \(a_0\) must be true. If \(a_0\) is omitted, the rule is called a constraint. If \(n = 0\), it is called a fact. For a rule \(r\), head\(r\) denotes \(\{a_0\}\); \(\text{pos}(r)\) and \(\text{neg}(r)\) denote the set \(\{a_1, \ldots, a_m\}\) and \(\{a_{m+1}, \ldots, a_n\}\), respectively. atoms\(r\) denotes the set of all atoms in \(r\), viz. \(\{\text{head}\(r\)\} \cup \text{pos}(r) \cup \text{neg}(r)\); and, atoms\(\Pi\) denotes the set of all atoms of \(\Pi\). heads\(\Pi\) denotes the set of all atoms occurring in the head of rules of \(\Pi\) (negative literals of \(\Pi\)).

Let \(\Pi\) be a program. I ⊆ atoms\(\Pi\) is called an interpretation of \(\Pi\). For an atom \(a\), a (resp. not \(a\)) is satisfied by \(I\), denoted by \(I \models a\) (resp. \(I \models \neg a\)), if \(a \in I\) (resp. \(a \notin I\)). A set of literals \(S\) is satisfied by \(I\) if \(I \models S\) if \(I\) satisfies each literal in \(S\). A rule \(r\) is satisfied by \(I\) if \(I \models \text{body}\(r\)\) or \(I \models \text{head}\(r\)\). I is a model of a program if it satisfies all its rules. An atom \(a\) is supported by \(I\) in \(\Pi\) if there exists \(r \in P\) such that head\(r\) = \(a\) and \(I \models \text{body}\(r\)\). The reduct of \(\Pi\) w.r.t. \(I\) (denoted by \(\Pi^I\)) is the program obtained from \(\Pi\) by deleting (i) each rule \(r\) such that \(\neg \text{neg}(r)\) \(\cap I = \emptyset\), and (ii) all negative literals in the bodies of the remaining rules. I is an answer set \([5]\) of \(\Pi\) if \(I = \Pi^I\) is the least Herbrand model of \(\Pi^I\) [14], which is the least fixpoint of the operator \(T_{\Pi}\) defined by \(T_{\Pi}(I) = \{a \mid \exists r \in \Pi, \text{head}\(r\) = a, I \models \text{body}\(r\)\}\) and is denoted by \(\text{lfp}(T_{\Pi})\).

Given an answer set \(I\) of \(\Pi\) and an atom \(q\), a justification for \(q\) wrt. \(\Pi\) is a set of rules \(S \subseteq I\) such that \(I \models \text{body}\(r\)\) for \(r \in S\) and \(q \in \text{lfp}(T_{\Pi^S})\). A justification \(S\) for \(q\) wrt. \(\Pi\) is minimal if there exists no proper subset \(S' \subset S\) such that \(S'\) is also a justification for \(q\) wrt. \(\Pi\). It is easy to see that if \(S\) is a minimal justification for \(q\) wrt. \(\Pi\) then \(\text{negs}(S) \cap \text{heads}(S) = \emptyset\) and heads\(S\) is an answer set of \(\Pi\).

2 PLANNING USING ASP
Answer set planning refers to answer set programming in planning [9]. It has been shown by Gebser et al. [4] that answer set planning, combined with good heuristics, can perform at the highest level of state-of-the-art planning systems.

A planning problem – as described using PDDL [6] – is a triple \((I, G, \text{D})\) where \(I\) and \(G\) encode the initial state of the world and the goal, respectively; and \(\text{D}\) (the domain) specifies the actions and their preconditions and effects. Given a problem \(P = (I, G, \text{D})\), answer set planning translates it into a program \(\pi(P, n)\) to compute solutions of \(P\), where \(n\) is constant indicating the maximal length of solutions that we are interested in (i.e., horizon). Program \(\pi(P, n)\) consists of different groups of rules:
- **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 conditions 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: The rules in this group generates action occurrences and ensure that only valid plans are generated.

3 EXPLAINABLE PLANNING

In explainable planning (XAIP) problems [7], the planning agent needs to find ways to ensure that its plans are understood and accepted by human users. As the model or knowledge base of the robot differs from that of the human users, a plan that may be optimal in the model of the robot may be suboptimal or, worse, infeasible in the model of the human user. Researchers have approached this problem from two perspectives. The first is by enforcing that the robot finds explicable plans (i.e., plans that are optimal or feasible in the model of the human user) [8, 15]. The second is for the robot to provide explanations to the human user and reconciling their two models such that the plan of the robot is also optimal in the reconciled model of the human user [3, 12, 13]. There is also recent work in balancing both approaches [1, 2].

In an XAI problem, a planning problem \( P = (I, G, D) \) is given, which is identical to the robot model \( P_a = (I_a, G_a, D_a) \). The human model of the planning problem \( P_h = (I_h, G_h, D_h) \) might be different from the model of the robot. The focus of this paper is in the model reconciliation process, i.e., to bring the human’s model closer to the robot’s model by means of explanations in the form of model updates. Given \( P_a \) and \( P_h \), a model reconciliation problem (MRP) is defined by a tuple \( (\pi^* , P_a, P_h) \), where \( \pi^* \) is a cost-minimal solution for \( P_a \). A solution for an MRP is a multi-model explanation \( \epsilon \), which creates a model \( P_h^\epsilon \) from \( P_a \) and \( P_h \) such that \( \pi^* \) is also a cost-minimal solution of \( P_h^\epsilon \) by inserting to \( P_h \) (or removing from \( P_h \)) some initial conditions, action preconditions, action effects, or goals. It is required that the changes in the model of the human must be consistent with the robot’s model.

4 EXPLANATIONS USING ASP

Let \( \pi_a \) be the program of the robot, \( \pi_h \) be the program of the human, and \( q \) be an atom of \( \pi_a \) such that \( \pi_a \models q \) and \( \pi_h \not\models q \). Assume that the robot wishes to explain to the human that \( q \), representing a plan, is true. The robot could do so by identifying an answer set \( I \) supporting \( q \) and explaining to the human by presenting a set of rules \( \lambda \subseteq \pi_a \), which might be a justification for \( q \) wrt. \( I \), such that an update of \( \pi_h \) by \( \lambda \) given \( I \) will allow the human to accept that \( q \) is entailed. In other words, the process of updating \( \pi_h \) by \( \lambda \) given \( I \) should result in a new program, denoted by \( \pi_h \otimes \lambda \) such that \( \pi_h \otimes \lambda \models q \). Therefore, we define the operator \( \otimes \) before we discuss the explanation process.

Definition 4.1 (Conditional Update). Let \( \pi_a \) and \( \pi_h \) be two programs. Further, let \( I \) be an answer set of \( \pi_a \) and \( \lambda \subseteq \pi_a \). The conditional update of \( \pi_a \) with respect to \( \lambda \) and \( I \) is the program \( \pi_h' \otimes \lambda \), denoted by \( \pi_h \otimes \lambda \), where \( \pi_h' \) is the collection of rules from \( \pi_h \setminus \lambda \) such that (i) head(r) \( \in I \) and neg(r) \( \cap \lambda = \emptyset \) or (ii) neg(r) \( \cap \) head(s) \( \neq \emptyset \).

Let \( \pi_a \) and \( \pi_h \) denote two arbitrary but fixed programs and \( q \in \text{atoms}(\pi_a) \) such that \( \pi_a \models q \) and \( \pi_h \not\models q \).

Definition 4.2 (Explanation). A subprogram \( \epsilon \subseteq \pi_a \) is a lp-explanation for \( q \) from \( \pi_a \) to \( \pi_h \) wrt. an answer set \( I \) of \( \pi_a \) (or an lp-explanation for \( q \) wrt. \( I \)) if \( \pi_h \otimes I \models q \) and \( \pi_h \not\models q \).

Algorithm 1: LP - Explanation(\( \pi_a, \pi_h, q \))

\begin{algorithm}
\begin{algorithmic}[1]
\Function {LP - Explanation} {\( \pi_a, \pi_h, q \)}
\Require Programs \( \pi_a, \pi_h \), atom \( q \)
\Ensure An explanation \( \epsilon \) for \( q \)
\If {\( \pi_a \cup \{\neg q\} \) has no answer set \( \text{then return nil} \)}
\State Let \( I \) be an answer set of \( \pi_a \cup \{\neg q\} \)
\State Compute \( I(\pi_a, I) \)
\State Compute an answer set \( J \) of \( I(\pi_a, I) \)
\State Compute \( \epsilon = \{\text{head}(r) \leftarrow \text{pos}(r), \text{neg}(r) \mid \text{head}(r) \leftarrow \text{pos}(r), \text{neg}(r), \text{ok}(r) \in I(\pi_a, I), \text{ok}(r) \in J\} \)
\State \Return \( \{\pi_h \setminus (\epsilon \setminus \pi_h, \pi_h \setminus \epsilon)\} \)
\EndIf
\EndFunction
\end{algorithmic}
\end{algorithm}

Algorithm 2: Computing Non-Trivial LP-Explanation

\begin{algorithm}
\begin{algorithmic}[1]
\Function {Computing Non-Trivial LP-Explanation} {\( \pi_a, \pi_h \)}
\Require Programs \( \pi_a, \pi_h \)
\Ensure A solution for an MRP
\If {\( \pi_a \setminus \{q\} \) has no model then}
\State Compute an answer set \( J \) of \( \pi_a \), \( \{q\} \)
\Else
\State Compute a non-trivial lp-explanation, Line 4 is replaced by the three lines (Lines 1-3) in Algorithm 2.
\EndIf
\Return \( \pi_h \otimes J \models q \)
\EndFunction
\end{algorithmic}
\end{algorithm}

5 CONCLUSIONS AND FUTURE WORK

In this abstract, we consider a general problem of updating a theory \( \pi_a \) so that the resulting theory \( \pi_h \) credulously entails an atom \( q \) given that \( q \) is entailed by a theory \( \pi_a \) using ASP by proposing the notion of conditional updates in logic programming and use it to define the notion of an explanation. We then show how it can be used to compute explanations for MRP problems. Future work includes experimentally evaluating this approach against the state of the art.
REFERENCES


