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,ton}@cs.nmsu.edu

Stylianos Vasileiou and William Yeoh
Washington University in St. Louis
St. Louis, MO
{v.stylianos,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 πk and a query q such that πa entails q and πk does not entail q, where the notion of entailment is dependent on the logical theories underlying πa and πk, how to change πk - given πa and the support for q in πa - so that πk 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 πk with respect to a query q; (2) define the notion of an explanation for a query q from a program πa to a program πk 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 \text{not} represents (default) negation. Intuitively, a rule states that if all positive literals \(a_i\) are believed to be true and no negative literal \text{not } a_i \text{ is believed to be true, then} a_0 \text{ must be true. If} a_0 \text{ is omitted, the rule is called a constraint. If} n = 0, \text{it is called a fact. For a rule} r, \text{head}(r) \text{denotes } a_0; \text{pos}(r) \text{and neg}(r) \text{denote the set } \{a_1, \ldots, a_m\} \text{ and } \{a_{m+1}, \ldots, a_n\}, \text{respectively. atoms}(r) \text{denotes the set of all atoms in } r, \text{viz. } (\text{head}(r)) \cup \text{pos}(r) \cup \text{neg}(r); \text{and, } \text{atom}(\Pi) \text{denotes the set of all atoms of } \Pi. \text{heads}(\Pi) (\text{negs}(\Pi)) \text{denotes the set of atoms occurring in the head of rules of } \Pi \text{ (negative literals of } \Pi). \text{Let } \Pi \text{ be a program. } I \subseteq \text{atoms}(\Pi) \text{ is called an interpretation of } \Pi. \text{For an atom } a, a \text{ (resp. } a \text{ not }) \text{ is satisfied by } I, \text{denoted by } I \models a \text{ (resp. } I \models \neg a \text{). A set of literals } S \text{ is satisfied by } I \text{ if } I \models S \text{ if } I \text{ satisfies each literal in } S. \text{A rule } r \text{ is satisfied by } I \text{ if } I \not\models \text{body}(r) \text{ or } I \not\models \text{head}(r). I \text{ is a model of a program if it satisfies all its rules. An atom } a \text{ is supported by } I \text{ in } \Pi \text{ if there exists } r \in P \text{ such that } \text{head}(r) = a \text{ and } I \models \text{body}(r). \text{The reduc of } \Pi \text{ wrt. } I \text{ (denoted by } \Pi^I) \text{ is the program obtained from } \Pi \text{ by deleting (i) each rule } r \text{ such that } \neg \text{neg}(r) \cap I \neq \emptyset, \text{and (ii) all negative literals in the bodies of the remaining rules. } I \text{ is an answer set of } \Pi \text{ if } I \text{ is the least Herbrand model of } \Pi^I [14], \text{which is the least fixpoint of the operator } T_I \text{ defined by } T_I(I) = \{a | \exists r \in \Pi, \text{head}(r) = a, I \models \text{body}(r)\} \text{ and is denoted by } \text{lfp}(T_I\Pi). \text{Given an answer set } I \text{ of } \Pi \text{ and an atom } q, \text{ a justification for } q \text{ wrt. } I \text{ is a set of rules } S \subseteq \Pi \text{ such that } I \models \text{body}(r) \text{ for } r \in S \text{ and } q \in \text{lfp}(T_{S^I}). \text{A justification } S \text{ for } q \text{ wrt. } I \text{ is minimal if there exists no proper subset } S' \subset S \text{ such that } S' \text{ is also a justification for } q \text{ wrt. } I. \text{It is easy to see that if } S \text{ is a minimal justification for } q \text{ wrt. } I \text{ then } \text{negs}(S) \cap \text{heads}(S) = \emptyset \text{ and heads}(S) \text{ is an answer set of } S. \text{2 PLANING USING ASP}\text{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.} \text{A planning problem – as described using PDDL [6] – is a triple } (I, G, D), \text{ where } I \text{ and } G \text{ encode the initial state of the world and the goal, respectively; and } D \text{ (the domain) specifies the actions and their preconditions and effects. Given a problem } P = (I, G, D), \text{ answer set planning translates it into a program } \pi(P, n) \text{ to compute solutions of } P, \text{ where } n \text{ is constant indicating the maximal length of solutions that we are interested in (i.e., horizon). Program } \pi(P, n) \text{ consists of different groups of rules:} \begin{itemize} \item \textbf{Facts:} These atoms define object constants, types of objects, actions, the initial state, and the goal state. \item \textbf{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) \text{ to denote that } l \text{ is true at step } t \text{ for } 1 \leq t \leq n. \end{itemize}
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 XAIP problem, a planning problem \( P = (I, G, D) \) is given, which is identical to the robot model work in balancing both approaches [1, 2].

Extended Abstract

AAMAS 2020, May 9–13, Auckland, New Zealand

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

Input: Programs \( \pi_a, \pi_h, \text{atom } q \)

Output: An explanation \( \epsilon \) for \( q \)

1. if \( \pi_a \cup \{¬ q\} \) has no answer set then return nil
2. Let \( I \) be an answer set of \( \pi_a \cup \{¬ q\} \)
3. Compute \( \Pi(\pi_a, I) \)
4. Compute an answer set \( J \) of \( \Pi(\pi_a, I) \)
5. Compute \( \epsilon = \{\text{head}(r) ← \text{pos}(r), \text{neg}(r) \mid \text{head}(r) ← \text{pos}(r), \text{neg}(r), \text{ok}(r) \in \Pi(\pi_a, I), \text{ok}(r) \in J\} \)
6. return \( \epsilon \) or \( \{\pi_h \cup \{\epsilon \}_{\pi_h} \} \)

Algorithm 2: Computing Non-Trivial LP-Explanation

1. if \( \Pi(\pi_a, I) \backslash \{q \} \) has no model then
2. return \( \{q \} \) only trivial lp-explanation exists
3. Compute an answer set \( J \) of \( \Pi(\pi_a, I) \) \( \backslash \{q \} \)

Given a program \( \pi_a \) and an answer set \( I \) supporting \( q \) of \( \pi_a \), we define \( \Pi(\pi_a, I) \) be the program such that:

- \( \Pi(\pi_a, I) \) contains the constraint \( ¬ q \);
- for each \( x \in \pi_a \text{ s.t. } \text{head}(x) \in I \text{ and } \text{neg}(x) \cap I = \emptyset \):
  - \( \text{head}(x) ← \text{pos}(x), \text{neg}(x), \text{ok}(x) \) is a rule in \( \Pi(\pi_a, I) \);
  - \( \text{ok}(x) \) ← is a rule of \( \Pi(\pi_a, I) \).
- \#minimize \{1, X : \text{ok}(X)\} is a rule of \( \Pi(\pi_a, I) \).
- No other rule is in \( \Pi(\pi_a, I) \).

Algorithm 1 can be used for computing an lp-explanation. To compute a non-trivial lp-explanation, Line 4 is replaced by the three lines (Lines 1-3) in Algorithm 2.

The proposed notion of an lp-explanation can be used in explainable planning as follows. Let \( \pi(P_a, t) \) and \( \pi(P_h, t) \) be the two programs encoding the planning model of the robot and the human, respectively. Assume that \( \alpha = \{a_1, \ldots, a_{t-1}\} \) is a plan in \( \pi(P_a, t) \) and is not a plan in \( \pi(P_h, t) \). This implies that \( \pi_a = \pi(P_a, t) \cup \text{occurs}^*(\alpha) \vdash \text{goal} \) and \( \pi_h = \pi(P_h, t) \cup \text{occurs}^*(\alpha) \not\vdash \text{goal} \). As such, an lp-explanation for the atom goal from \( \pi_a \) to \( \pi_h \) could explain why \( \alpha \) is not a solution in the model of \( P_h \). Indeed, Algorithm 1 can be used to compute an lp-explanation for the atom goal from \( \pi_a \) to \( \pi_h \), i.e., an explanation for the MRP between the robot and the human. This can be used as a seed for computing complete explanations for the MRP.

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