Action Languages, Answer Sets, and Planning

Vladimir Lifschitz

Department of Computer Sciences, University of Texas at Austin,
Austin, TX 78712, USA

Summary. This is a discussion of some of the achievements and challenges related to representing actions and the design of planners from the perspective of logic programming. We talk about recent work on action languages and translating them into logic programming, on representing possible histories of an action domain by answer sets, on efficient implementations of the answer set semantics and their use for generating plans, and on causal logic and its relation to planning algorithms. Recent progress in these areas may lead to the creation of planners which are based on the ideas of logic programming and combine the use of expressive action description languages with efficient computational procedures.

1 Introduction

This is a discussion of some of the achievements and challenges related to representing actions and the design of planners from the perspective of logic programming.

Describing the effects of actions and generating plans have been at the center of Artificial Intelligence research since the late fifties. In his classical paper *Programs with Common Sense*, John McCarthy [21] discusses the problem of planning his way to the airport, in which an initial state, described by the formula $at(I, desk)$, needs to be turned into a goal state that satisfies $at(I, airport)$. The goal can be achieved by executing first the action $go(desk, car, walking)$, and then $go(home, airport, driving)$. The effect of go is described by the axiom

$$did(go(x, y, z)) \supset at(I, y). \tag{1}$$

Other axioms tell us under what conditions it is possible to walk or drive from x to y; the possibility of executing an action a is denoted by $can(a)$.

More recently, work on representing changes caused by actions has been extended to causal dependencies that do not involve actions, including the cases when the cause and the effect are not even separated by a time interval—they obtain in the same state. If I have a watch on my wrist then $at(I, airport)$ is a cause for $at(watch, airport)$. There is no action or passage of time involved in this causal dependency; it is "static," unlike the "dynamic" dependency between going from x to y and being at y. Combinations of dynamic and static causal dependencies is what allows actions to have indirect effects, or

K. R. Apt et al. (eds.), *The Logic Programming Paradigm*

"ramifications." When I go to the airport, $at(I, airport)$ is the direct effect of this action, and $at(watch, airport)$ is one of its many indirect effects.

Once an action domain is described by a formal theory, it may be possible to use a theorem prover for solving planning problems in this domain. The paper by Cordell Green entitled *Application of Theorem Proving to Problem Solving* [11] shows, for instance, how a resolution proof procedure can be used to solve the problem of getting a bunch of bananas hanging from the ceiling just beyond a monkey's reach. The answer literal method allows us to extract the value of the situation variable s from the proof of the formula

$$\exists s\, HAS(monkey, bananas, s)$$

found by this procedure. The value, which equals

$$\begin{array}{l} reach(monkey, bananas, \\ \quad climb(monkey, box, \\ \quad\quad move(monkey, box, under\text{-}bananas, s_0))), \end{array}$$

encodes a solution to the planning problem. Finding a plan by extracting a term from an existence proof is known as *deductive planning*.

The answer literal method was a precursor of logic programming, and it is not surprising that logic programming has always played an important role in research on describing the effects of actions and on planning.

One other reason why the study of causal dependencies is related to logic programming is that causation is noncontrapositive, just like rules in logic programs. We can say, for instance, that being underpaid usually causes a person to look for another job. But it is not true that the lack of interest in other jobs is usually a cause for being paid well. This is similar to the difference between the rules

$$p \leftarrow not\ q$$

and

$$q \leftarrow not\ p.$$

This formal similarity makes the if operator used in logic programming a better tool for formalizing causality than material implication with its contraposition property

$$\neg p \supset q \equiv \neg q \supset p.$$

Over the years, the early work referenced above has inspired many research projects and scores of papers and monographs. This note is not an attempt to survey all important ideas developed in this research. We concentrate here on a small fraction of these ideas, on just a few threads, and try to show how they interact with each other and what can be expected from them in the future.