Transformations of Logic Programs Related to Causality and Planning

Esra Erdem and Vladimir Lifschitz

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA
{esra,vl}@cs.utexas.edu

Abstract. We prove two properties of logic programs under the answer set semantics that may be useful in connection with applications of logic programming to representing causality and to planning. One theorem is about the use of disjunctive rules to express that an atom is exogenous. The other provides an alternative way of expressing that a plan does not include concurrently executed actions.

1 Introduction

In this note we prove two properties of logic programs under the answer set semantics [3] that may be useful in connection with applications of logic programming to representing causality and to planning.

According to the first of the two theorems, replacing a disjunctive rule of the form

$$p ; \neg p \leftarrow \tag{1}$$

(where ; is the disjunction symbol and $\neg$ is classical negation) by the nondisjunctive rules

$$\begin{array}{l} p \leftarrow \mathit{not}\ \neg p \\ \neg p \leftarrow \mathit{not}\ p \end{array} \tag{2}$$

is an equivalent transformation, as far as consistent answer sets are concerned. Under some conditions, this fact follows from [1]; our Theorem 1 is more general.

Rules (1) and (2) are of interest in connection with the system of causal logic introduced by McCain and Turner [9]. This logic is based on the "principle of universal causation," according to which, in a causally possible world history, every fact that obtains has a cause. McCain and Turner point out that the principle of universal causation needs to be relaxed in applications. For instance, in reasoning about actions, the causes for the occurrences and the non-occurrences of actions are usually not given. Instead, an occurrence of an action a at time t is postulated to have a cause whenever a occurs at t. In the language of [9], this is expressed by $a_t \Rightarrow a_t$. Similarly, the non-occurrence of an action a at time t is caused whenever a does not occur at t: $\neg a_t \Rightarrow \neg a_t$. In [9], the authors say that occurrences of actions are "exogenous" to the causal theory. Values of fluents at time 0 are usually treated as exogenous also. Generally, the assumption that an atom p is exogenous is expressed by

$$p \Rightarrow p \tag{3}$$

$$\neg p \Rightarrow \neg p. \tag{4}$$

Proposition 6.1 from [8] shows that the language of causal logic used in these examples is closely related to logic programming. It shows, in particular, that causal rules (3) and (4) can be translated into logic programming as rules (2). From Theorem 1 proved below we learn that there is another way to express in logic programming that p is exogenous: disjunctive rule (1) can be used instead. This possibility played a role in our experiments with the use of system DLV [2] for planning, as discussed in the next section.[1]

The second theorem proved in this note has to do with expressing uniqueness assumptions in logic programming. In a language containing a binary predicate constant p and no function constants, the constraints

$$\leftarrow p(c,d), p(c',d') \qquad (c \neq c' \text{ or } d \neq d') \tag{5}$$

$(c, d, c', d'$ range over the object constants) eliminate the answer sets that contain more than one atom of the form $p(c,d)$. Essentially the same result can be achieved using the rules

$$\begin{array}{ll}
p_1(c) \leftarrow p(c,d), & \\
p_2(d) \leftarrow p(c,d), & \\
\leftarrow p_1(c), p_1(c') & \quad (c \neq c'), \\
\leftarrow p_2(d), p_2(d') & \quad (d \neq d'),
\end{array}$$

where p_1, p_2 are auxiliary predicates. Theorem 2 is a generalization of this fact. Like Theorem 1, it is related to applications of DLV and similar systems to planning. When the goal is to find a plan in which actions are executed sequentially, the logic programming representation of the problem has to contain a "no-concurrency constraint" similar to (5). The equivalent transformation provided by Theorem 2 may allow us to state this constraint in a way that provides some computational advantages. An example is given in the next section.

2 Planning in the Blocks World

We define here a logic programming formalization of the blocks world in which *on(B,L)* expresses that a block B is at a location L, where L can be a block or the table. There is an action of moving a block B onto a location L denoted by $move(B,L)$. Time is represented by an initial segment of nonnegative integers $0, \ldots, t_{max}$.

[1] For details of our experiments with the use of systems DLV and SMODELS [11] to solve planning problems in the blocks world, see `http://www.cs.utexas.edu/users/esra/experiments/experiments.html` .

The first rule of the program describes the effect of moving a block:

$$on(B, L, T_1) \leftarrow move(B, L, T) \qquad (T_1 = T + 1).$$

The commonsense law of inertia is postulated in the form

$$on(B, L, T_1) \leftarrow on(B, L, T), not\ \neg on(B, L, T_1) \qquad (T_1 = T + 1).$$

A block can be moved only when it's clear:

$$\leftarrow move(B, L, T), on(B_1, B, T).$$

No two blocks can be on the same block at the same time:

$$\leftarrow on(B_1, B, T), on(B_2, B, T) \qquad (B_1 \neq B_2).$$

Wherever a block is, it's not anywhere else:

$$\neg on(B, L_1, T) \leftarrow on(B, L, T) \qquad (L_1 \neq L).$$

Every block is part of a tower supported by the table:

$$\begin{array}{ll} supported(B, T) \leftarrow on(B, table, T), & \\ supported(B, T) \leftarrow on(B, B_1, T), supported(B_1, T) & (B \neq B_1), \\ \leftarrow not\ supported(B, T). & \end{array}$$

No actions are executed at time t_{max}:

$$\leftarrow move(B, L, t_{max}).$$

Both the initial values of fluents and the occurrences of actions are exogenous:

$$\begin{array}{l} on(B, L, 0) \leftarrow not\ \neg on(B, L, 0), \\ \neg on(B, L, 0) \leftarrow not\ on(B, L, 0), \\ move(B, L, T) \leftarrow not\ \neg move(B, L, T), \\ \neg move(B, L, T) \leftarrow not\ move(B, L, T). \end{array}$$

Actions cannot be executed concurrently:

$$\leftarrow move(B, L, T), move(B_1, L_1, T) \qquad (B \neq B_1 \text{ or } L \neq L_1).$$

Every possible "history" of the blocks world over the time interval $0, \ldots, t_{max}$ corresponds to an answer set for this program. Given this formalization, DLV can be used to solve planning problems in the blocks world.

The two theorems proved in this note allow us to modify the last two groups of rules. According to Theorem 1, we can express that the initial values and actions are exogenous by disjunctive rules:

$$on(B, L, 0); \neg on(B, L, 0) \leftarrow \ ,$$
$$move(B, L, T); \neg move(B, L, T) \leftarrow \ .$$

Theorem 2 shows that the no-concurrency constraints can be rewritten as follows:

$$\begin{array}{ll} move_1(B, T) \leftarrow move(B, L, T), & \\ move_2(L, T) \leftarrow move(B, L, T), & \\ \leftarrow move_1(B_1, T), move_1(B_2, T) & (B_1 \neq B_2), \\ \leftarrow move_2(L_1, T), move_2(L_2, T) & (L_1 \neq L_2). \end{array}$$

Since the other rules of the program make it impossible to move a block to two different places at the same time, the rules involving $move_2$ are actually redundant, and the no-concurrency constraint can be expressed by the rules in the first and the third lines. This representation of nonconcurrency in the blocks world is used in [10].

Niemelä's modification has a significant effect on the size of the program after grounding. For the original program, this size grows as n^4 with the number n of blocks; for the modified program, it grows as n^3.

As to the computation time of DLV, each of the two modifications led to substantial improvements for the November 21, 1998 version of the system. With the additional optimizations incorporated in the June 11, 1999 version of DLV, the efficiency is not affected by either transformation in an essential way.

In our recent experiments with SMODELS, the computation time was found to be significantly affected by the second transformation. In one of the planning experiments with 8 blocks, the computation time was reduced from 219 seconds to 27 seconds. For a planning problem involving 11 blocks, the corresponding numbers were 637 seconds and 113 seconds.

3 Programs

In this note, a *program* is a set of rules of the form

$$L_1; \ldots; L_k \leftarrow L_{k+1} \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \tag{6}$$

where $0 \leq k \leq m \leq n$, and each L_i is a literal. (A literal is a propositional atom possibly preceded by classical negation $\neg$; a literal possibly preceded by negation as failure is called a *rule element*.) If $k = 0$, the rule is called a *constraint*. We denote the set of literals in the language of a program Π by $lit(\Pi)$.

For the definition of the reduct Π^X of a program Π relative to a subset X of $lit(\Pi)$, and for other definitions related to the answer set semantics, see [3] or [6].

4 Theorem 1

Theorem 1. *For any program Π and any atom p, the programs*

$$\begin{array}{l} \Pi \\ p; \neg p \leftarrow \end{array} \tag{7}$$

and

$$\begin{array}{l} \Pi \\ p \leftarrow \mathit{not}\ \neg p \\ \neg p \leftarrow \mathit{not}\ p \end{array} \tag{8}$$

have the same consistent answer sets.

It is essential that the statement of the theorem is restricted to consistent answer sets. For instance, if Π consists of the rules

$$\begin{array}{l} p \leftarrow \neg p \\ \neg p \leftarrow p \end{array}$$

then the set of all literals is an answer set for (7), but not for (8).

Both the theorem and the proof below can be extended to programs with negation as failure allowed in the heads of rules [6].

In the following lemma, used in the proof of Theorem 1, we denote program (7) by Π_1 and program (8) by Π_2.

Lemma 1. *For any consistent set X of literals and any subset Y of X, Y is closed under Π_1^X iff Y is closed under Π_2^X.*

Proof. Let X be a consistent set of literals, and let Y be a subset of X. Since X is consistent, at least one of the literals p, $\neg p$ does not belong to X. Consider 3 cases.

Case 1: $p \notin X$ and $\neg p \notin X$. Then Π_1^X is

$$\begin{array}{l} \Pi^X \\ p; \neg p \leftarrow \end{array}$$

and Π_2^X is

$$\begin{array}{l} \Pi^X \\ p \leftarrow \\ \neg p \leftarrow \end{array}$$

Since $Y \subseteq X$, $p \notin Y$ and $\neg p \notin Y$. Consequently, Y is closed neither under Π_1^X nor under Π_2^X.

Case 2: $p \in X$ and $\neg p \notin X$. Then Π_1^X is

$$\begin{array}{l} \Pi^X \\ p; \neg p \leftarrow \end{array}$$

and Π_2^X is

$$
\begin{array}{l}
\Pi^X \\
p \leftarrow
\end{array}
$$

The “if part”: assume that Y is closed under Π_2^X. Then Y is closed under Π^X. In addition, p is in Y. Therefore, Y is also closed under Π_1^X. The “only if” part: assume that Y is closed under Π_1^X. Then Y is closed under Π^X. As $\neg p$ is not in X due to the case assumption, and $Y \subseteq X$, $\neg p$ is not in Y either. Then p is in Y. Therefore, Y is closed under Π_2^X also.

Case 3: $p \notin X$ and $\neg p \in X$. Similar to Case 2.

Proof of Theorem 1: Let X be a consistent set of literals. We want to show that X is an answer set for Π_1^X iff X is an answer set for Π_2^X. Recall that a consistent set X of literals is an answer set for a disjunctive program without negation as failure iff it is a minimal set closed under this program.

Notice first that X is closed under Π_1^X iff X is closed under Π_2^X, due to Lemma 1 for $Y = X$. It remains to check that

(a) X is minimal among the sets closed under Π_1^X
iff
(b) X is minimal among the sets closed under Π_2^X.

Assume (a), and consider any subset Y of X that is closed under Π_2^X. Due to Lemma 1, Y is closed under Π_1^X as well. By (a), it follows that $X = Y$. The proof in the other direction is similar.

5 Theorem 2

In the statement of Theorem 2, Π is a program, $C_1, \ldots, C_n$ $(n > 0)$ are sets, and p is a function such that for all $c_1 \in C_1, \ldots, c_n \in C_n$ its values $p(c_1, \ldots, c_n)$ are pairwise distinct atoms in the language of Π. The expressions $p_i(c)$, where $1 \leq i \leq n$ and $c \in C_i$, are assumed to be pairwise distinct atoms that do not belong to the language of Π.

Theorem 2. *If X is an answer set for the program Π_1 obtained from Π by adding the rules*

$$p_i(c_i) \leftarrow p(c_1, \ldots, c_n) \tag{9}$$

$$\leftarrow p_i(c), p_i(c') \tag{10}$$

$(1 \leq i \leq n,\ c_1 \in C_1, \ldots, c_n \in C_n,\ c, c' \in C_i)$ then

$$X \cap lit(\Pi) \tag{11}$$

is an answer set for the program Π_2 obtained from Π by adding the rules

$$\leftarrow p(c_1,\dots,c_n), p(c'_1,\dots,c'_n) \qquad (c_1,c'_1 \in C_1,\ \dots, c_n, c'_n \in C_n, \langle c_1,\dots,c_n\rangle \neq \langle c'_1,\dots,c'_n\rangle). \tag{12}$$

Moreover, every consistent answer set for Π_2 can be represented in form (11) for some consistent answer set X for Π_1.

The theorem asserts, in other words, that replacing constraints (12) with rules (9) and (10) may extend the answer sets by new literals $p_i(c)$, $\neg p_i(c)$, but the parts of the answer sets that belong to the language of Π remain the same.

To apply Theorem 2 to the blocks world example (Section 2), we use it to replace the no-concurrency constraints by the rules involving $move_1$ and $move_2$ consecutively for $T = 0$, $T = 1$, etc., each time with

- $n = 2$,
- C_1 equal to the set of block constants,
- C_2 equal to the set of location constants,
- $p(c_1, c_2)$ equal to $move(c_1, c_2, T)$.

The proof of Theorem 2 is based on two facts. One is the splitting set theorem (see [7] for terminology and notation). The other is a property of constraints that easily follows from the definition of an answer set. Recall that a set X of literals is said to *violate* a constraint

$$\leftarrow L_1,\dots,L_m, not\ L_{m+1},\dots, not\ L_n$$

if $L_1,\dots,L_m \in X,\ L_{m+1},\dots,L_n \notin X$. The fact about constraints that we need is that the effect of adding a set of constraints to a program is to eliminate the answer sets that violate at least one of these constraints.

For any subset Y of $lit(\Pi)$, we define:

$$Y^* = \{p_i(c_i) : c_1 \in C_1,\dots,c_n \in C_n,\ p(c_1,\dots,c_n) \in Y\}.$$

Clearly

$$Y^* \cap lit(\Pi) = \emptyset. \tag{13}$$

Lemma 2. *A consistent set X of literals is an answer set for the union of Π with rules (9) iff $X = Y \cup Y^*$ for some consistent answer set Y for Π.*

Proof. Take $U = lit(\Pi)$. This set splits the union of Π with rules (9), and Π is the bottom of this union relative to U. By the splitting set theorem, X is an answer set for the union program iff X can be represented as a union of an answer set Y for Π with an answer set for program $e_U(\Pi, Y)$. The latter consists of the rules

$$p_i(c_i) \leftarrow \tag{14}$$

for all $c_1,\dots,c_n$ such that

$$p(c_1,\dots,c_n) \in Y.$$

Consequently, its only answer set is Y^*.

Proof of Theorem 2: Let X be an answer set for Π_1. Then X is an answer set for the union of Π with rules (9). As any answer set for a program with constraints without negation as failure, X is consistent. By Lemma 2, X can be represented as $Y \cup Y^*$, where Y is an answer set for Π. It is clear that

$$X \cap lit(\Pi) = Y. \tag{15}$$

We will show that Y is an answer set for Π_2. As Y is an answer set for Π, it is sufficient to show that Y does not violate any of constraints (12). Assume that it does. That means that Y contains a pair of distinct atoms $p(c_1, \ldots, c_n)$ and $p(c'_1, \ldots, c'_n)$. Take an i such that $c_i \neq c'_i$. Both $p_i(c_i)$ and $p_i(c'_i)$ are in Y^*, and consequently in X. It follows that X violates (10), contrary to the assumption that it is an answer set for Π_1.

Now we will show that any consistent answer set Y for Π_2 can be represented as $X \cap lit(\Pi)$ for some consistent answer set X for Π_1. Take $X = Y \cup Y^*$. By (15), to complete the proof, we only need to show that X is a consistent answer set for Π_1. By Lemma 2, X is a consistent answer set for the union of Π with rules (9). Assume that X violates constraints (10). Then X contains a pair of atoms $p_i(c)$, $p_i(c')$ with $c \neq c'$. Since $p_i(c) \in X = Y \cup Y^*$ and $Y \subseteq lit(\Pi)$, it follows that $p_i(c) \in Y^*$. This means that $c = c_i$ for some atom $p(c_1, \ldots, c_n)$ in Y. Similarly, $c' = c'_i$ for some atom $p(c'_1, \ldots, c'_n)$ in Y. It follows that Y violates (12), contrary to the assumption that it is an answer set for Π_2.

6 Related Work

Gelfond *et al* [4] compare a disjunctive logic program Π with the nondisjunctive program Π' obtained by replacing each rule of form (6) by k rules:

$$\begin{array}{l}
L_1 \leftarrow L_{k+1}, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n, not\ L_2, \ldots, not\ L_k \\
\vdots \\
L_k \leftarrow L_{k+1}, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n, not\ L_1, \ldots, not\ L_{k-1}
\end{array}$$

Here each L_i is a literal, i.e., an atom possibly preceded by classical negation, and $0 \leq k \leq m \leq n$. They show that each answer set for Π' is also an answer set for Π. Ben-Eliyahu and Dechter [1] show that Π is equivalent to Π' if Π is "head-cycle free". Their proof is based on translating Π and Π' into propositional logic; for the two programs, the results happen to be the same.

Our Theorem 1 is different from that of [1] in two ways. First, it allows us to replace a single disjunctive rule (1)—and consequently any finite set of rules of this form—by nondisjunctive rules. This is not the same as completely eliminating all disjunctive rules from a program. Second, the disjunctive programs we consider are not required to be head-cycle free.

Consider, for instance, the program

$$p; q \leftarrow \tag{16}$$
$$p; \neg p \leftarrow \tag{17}$$
$$p \leftarrow q \tag{18}$$
$$q \leftarrow p. \tag{19}$$

By Theorem 1, we can replace rule (17) by nondisjunctive rules (2) and leave rule (16) as it is. The theorem from [1] would not allow us to justify this replacement. It is not applicable to program (16)–(19) because this program is not head-cycle free: its dependency graph has a cycle that goes through the atoms p and q that belong to the head of the same disjunctive rule.

In satisfiability planning, auxiliary atoms similar to $p_i(c)$ from our Theorem 2 are sometimes used to eliminate action symbols altogether [5].

Acknowledgments

We would like to thank Wolfgang Faber, Nicola Leone, Norman McCain, Ilkka Niemelä, Gerald Pfeifer, Patrik Simons and Hudson Turner for useful discussions related to representing the blocks world in logic programming and to the use of DLV and SMODELS. This work was partially supported by National Science Foundation under grant IIS-9732744.

References

1. Rachel Ben-Eliyahu and Rina Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1994.
2. Thomas Eiter, Nicola Leone, Christinel Mateis, Gerald Pfeifer, and Francesco Scarcello. A deductive system for non-monotonic reasoning. In *Proceedings of the 4th International Conference on Logic Programmimg and Nonmonotonic Reasoning*, pages 363–374. Springer-Verlag, 1997.
3. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
4. Michael Gelfond, Halina Przymusinska, Vladimir Lifschitz, and Miroslaw Truszczyński. Disjunctive defaults. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 230–237, 1991.
5. Henry Kautz and Bart Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of 13th National Conference on AI*, pages 1194–1201, 1996.
6. Vladimir Lifschitz. Foundations of logic programming. *Principles of Knowledge Representation*, pages 69–127, 1996.
7. Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In *Proceedings of the Eleventh International Conference on Logic Programming*, pages 23–37, 1994.
8. Norman McCain. *Causality in Commonsense Reasoning About Actions.* PhD thesis, The University of Texas at Austin, 1997.

9. Norman McCain and Hudson Turner. Causal theories of action and change. In *Proceedings of AAAI-97*, pages 460–465, 1997.
10. Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 1999. To appear.
11. Ilkka Niemelä and Patrik Simons. Efficient implementation of the well-founded and stable model semantics. In *Proceedings of Joint International Cenference and Symposium on Logic Programming*, pages 289–303, 1996.