Embedding Revision Programs in Logic Programming Situation Calculus

Chitta Baral
Department of Computer Science
University of Texas at El Paso
El Paso, Texas 79968, U.S.A.
chitta@cs.utep.edu

Abstract

Revision programs were introduced by Marek and Truszczynski to specify change in knowledge bases. In this paper we show how to embed revision programs in logic programs with situation calculus notation. We extend Marek and Truszczynski's approach to allow incomplete initial knowledge base and extend the rules of revision programs to depend both on the initial and the final knowledge base. We show how revision programs and its proposed extension can be incorporated in theories of actions and how our usage of situation calculus notation makes this easier and elegant.

1 Introduction

Revision Programs were introduced by Marek and Truszczynski [MT94c, MT94a, MT94b] to specify revision/change in knowledge bases (databases) and belief sets. Unlike earlier approaches in belief revision where "updates" [1] were represented by classical theories, revision programs are a collection of rules, similar to rules in logic programs and have a non-classical semantics of "$\leftarrow$".

Although revision programs may be considered to specify various kinds of "updates", in this paper we consider it as representing an update in the sense of [Win89, KM92]. *In other words we consider a revision program to specify the effect (possibly complex[2]) of an action.*

In [MT94a] Marek and Truszczynski show that logic programs (under the stable semantics) can be embedded in revision programs. In [Sac93, Sch90] it is shown that logic programming with stable model semantics is more expressive than first-order logic. *Hence, revision programs are more expressive than first-order logic in specifying an update.* We further discuss the expressiveness issue in the conclusion.

Marek and Truszczynski [MT94c, MT94a, MT94b] also extensively study several other properties of revision programs. They show [MT94a] that several notions of logic programming, including one-step-provability operator, stable and supported models, generalize to revision programming. They also find a game theoretic characterization related to revision programming.

Unfortunately, they do not provide a translation and/or implementation of revision programs in logic programming. In [MT94a] (page 10) they say

> "Theorem 3.1 states that logic programs can be reinterpreted as revision programs. The question whether there is a simple representation of revision programming in terms of logic programming remains open."

A translation of revision programs to logic programming would allow us to use the available logic programming interpreters such as SLG [WC93] to compute updated theories when updates are given as revision programs.

The main goal of this paper[3] is to fill this gap and provide an embedding of revision programs in logic programming. Our embedding uses the situation calculus notation and hence will allow us not only to reason about the updated knowledge base after one update but will allow us to reason when given a sequence of updates. Moreover, our embedding fits into the recent flurry of formalizations of reasoning about actions in logic programming with situation

[1]Several different "update" operators (update, revision, contraction, erasure, forget etc) [KM89, GM88] and the relation between them have been studied [KM92] and postulates have been suggested for some of these operators [GM88, KM92].

[2]While a non-complex effect of an action relates actions and fluents, complex effect of action also incorporates relationship between the fluents.

[3]This paper expands on some of the key sections of our earlier conference paper [Bar94] by adding proofs, by adding additional motivations and by describing the role of this paper with respect to the big picture of the research in reasoning about actions. To our knowledge [Bar94] was the first paper that answered the open question regarding if revision programs can be represented using logic programs.

calculus [BG93, GL92, Dun93, DDS93, HT93, Tur94]. Our embedding will allow easier assimilation of representing effect of actions by revision programs into these logic programming theories of actions.

We would now like to introduce the term *Logic Programming Situation Calculus* to refer to the language of logic programming where the situation calculus notation is used. This will help us to distinguish it from the term 'Logic Programming' that does not demand situation calculus notation, and the term 'Situation Calculus' [MH69] that originally was rooted in first-order logic. Logic programming situation calculus has been extensively used in logic programming theories of actions [BG93, GL92, Dun93, DDS93, HT93, Tur94] and we believe will be useful in other theories of change such as representing transactions [BK93].

We now further motivate revision programs through an example and discuss its syntax and semantics as given by Marek and Truszczynski .

2 Revision Programs

Example 1 Consider a knowledge base of employees in a firm. One of the departments D in that firm has three employees: John, Peter and Carl. During an organizational shake up the managers (based on their observations of the working habits and relations between their employees) in the firm decide to use the following update specification P with respect to the department D.

"John remaining in the department *causes* Peter to leave the department and Carl remaining in the department *causes* John to stay in the department".

The managers want that minimal change (only to the extent dictated by P) is made to D because of the action 'shake up'. I.e., they do not want to make unnecessary changes.

Let us first analyze the statements in P. The intended meaning of the first statement is different from the statement "either John leaves the department or Peter leaves the department". *The first statement in P can explicitly dictate the removal of Peter but can not dictate the explicit removal of John.*

If the update specification is specified in propositional theory or in first order logic in a straight forward manner they would be equivalent.

The *causality* in the statement "John remaining in the department *causes* Peter to leave the department" is treated differently from the first order implication "John remains in the department D implies Peter does not remain in the department D". Our intent is to give a higher priority to "John than to Peter". Non-classical treatment of implication also happens in several other places, such as logic programs, conditionals, counterfactuals, etc. For a more detailed discussion on causality and its role in reasoning about actions we refer the reader to [Lin95, MT95, Bar95]. □

Marek and Truszczynski [MT94a] introduce the notion of *revision programs* to be able to represent such specifications.

2.1 Syntax and Semantics of Revision Programs

We now describe the syntax and semantics of revision programs.

Definition 1 *[MT94c]* An *update rule* can be of the following two forms

$$in(p) \leftarrow in(q_1), \ldots, in(q_m), out(s_1), \ldots, out(s_n) \quad (1)$$

$$out(p) \leftarrow in(q_1), \ldots, in(q_m), out(s_1), \ldots, out(s_n) \quad (2)$$

where p, q_i's and s_j's are atoms.

A collection of update rules is called a revision program. □

Let U be a denumerable set. Its elements are referred to as atoms. A knowledge base is any subset of U. By $\neg U$ we mean the set $\{\neg a \ : \ a \in U\}$. Elements of $U \cup \neg U$ are called literals. A revision program uses a syntax similar to logic programs except that it has two special operators "in" and "out". For any atom a the intuitive meaning $in(a)$ is that the atom a is present in the updated knowledge base. Similarly the meaning of $out(a)$ is that the atom a is absent in the updated knowledge base. For any atom p in U, $in(p)$ an $out(p)$ are referred to as r-literals of U.

Intuitively, the rules (1) and (2 mean that "having $q_1, \ldots, q_m$ as $true$ and $s_1, \ldots, s_n$ as $false$ in a state *causes* p to be $true$ (respectively, $false$) in that state."

The statement "John remaining in the department *causes* Peter to leave the department" is written as the update rule:

$$out(peter) \leftarrow in(john)$$

Definition 2 *[MT94c]* Let P be a revision program. By $norm(P)$ we denote the definite program obtained from P by replacing each occurance of $in(a)$ by $holds(a, s_0)$ and each occurance of $out(b)$ by $holds(b', s_0)$. The *necessary change* for P is the pair (I, O) where $I = \{a \ : \ holds(a, s_0) \in$ least model of $norm(P)\}$ and $O = \{b \ : \ holds(b', s_0) \in$ least model of $norm(P)\}$. P with necessary change (I, O) is said to be *coherent* if $I \cap O = \emptyset$. □

Definition 3 *[MT94c]* Let P be a revision program and D_I and D_R be two knowledge bases.

P_{D_R} is the revision program obtained from P by eliminating from P every rule of the type 1 or 2 such that $q_i \notin D_R$ or $s_j \in D_R$.

$P_{D_R}|D_I$ is the revision program obtained from P_{D_R} by eliminating from the body of each rule in P_{D_R} $in(a)$ if $a \in D_I$ and $out(a)$ if $a \notin D_I$.

If $P_{D_R}|D_I$ with necessary change (I, O) is coherent and $D_R = D_I \cup I \setminus O$ then D_R is called *P-justified revision* of D_I, and we write $D_I \xrightarrow{P} D_R$. □

Intuitively, P_{D_R} is the set of rules obtained from P by removing all rules in P whose body is not satisfied by D_R; and $P_{D_R}|D_I$ is the set of rules obtained from P_{D_R} by removing all r-literals that satisfy D_I from the bodies of rules in P_{D_R}. Also, D_R is a *P-justified revision* of D_I if the change between D_R and D_I is dictated by P.

2.2 Examples

We now illustrate the semantics of revision programs through several examples.

Example 2 Let $D_I = \{f, g\}$ and P_1 be the revision program

$$\left.out(g) \leftarrow in(f)\ \right\}\ P_1$$

Let D_R be $\{f\}$.

P_{1D_R} is same as P_1 and $P_{1D_R}|D_I = \{out(g)\}$ and hence is coherent with the necessary change $(\emptyset, \{g\})$ and $D_R = D_I \cup \emptyset \setminus \{g\}$. Hence, $D_I \xrightarrow{P} D_R$. $\square$

Example 3 Let P_2 be

$$\left.\begin{array}{l} out(g) \leftarrow in(f) \\ out(f) \leftarrow in(g) \end{array}\right\} P_2$$

It is easy to see that P_2-justified revisions of D_I are $\{f\}$ and $\{g\}$. $\square$

Example 4 Let P_3 be

$$\left.\begin{array}{l} out(g) \leftarrow in(f) \\ out(f) \end{array}\right\} P_3$$

It is easy to see that P_3-justified revisions of D_I is $\{g\}$. $\square$

Example 5 Let P_4 be

$$\left.\begin{array}{l} in(f) \leftarrow in(f) \\ in(g) \leftarrow in(g) \end{array}\right\} P_4$$

It is easy to see that P_4-justified revisions of D_I is $\{f, g\}$. $\square$

3 Embedding Revision Programs in Extended Logic Programs

In this section we translate revision programs to extended logic programs [GL90] and show that the answer sets of the translated program correspond to the P-revisions.

The extended logic program $\Pi(P, D_I)$ where P is the revision program and D_I is the initial knowledge base, uses variables of three sorts: *situation* variables $S, S', \ldots$, *fluent* variables $F, F', \ldots$, and *action*[4] variables $A, A', \ldots$. We follow the convention of variables starting with capital letters and constants starting with small letters.

The program $\Pi(P, D_I)$ consists of the translations of the individual update rules and the initial knowledge base in P and certain other rules. We now present the translation $\Pi(P, D_I)$ where s is the situation corresponding to the initial knowledge base D_I, a corresponds to an action whose execution dictates the updating described by the revision program P and $res(a, s)$ is the situation corresponding to the knowledge base obtained by updating the initial knowledge base with the revision program P.

[4]The revision variables correspond to the action variables in situation calculus and in the translation of the language $\mathcal{A}$ to extended logic programs in [GL92]

Algorithm 1 *[Translating Revision Programs – with CWA about the initial database]*

Step 1. Initial Database – ID

If p is proposition in the initial database then $\Pi(P, D_I)$ contains

(1.1) $holds(p, s)$

and the rule

(1.2) $\neg holds(F, s) \leftarrow not\ holds(F, s)$

which encodes the CWA about the initial database. We denote the set of rules obtained in Step 1 by ID meaning the initial database.

Step 2. Inertia Rule – IR

(2.1) $holds(F, res(a, s)) \leftarrow holds(F, s), not\ ab(F, a, s)$

(2.2) $\neg holds(F, res(a, s)) \leftarrow \neg holds(F, s), not\ ab(F', a, s)$

These rules is motivated by the minimality consideration that only changes that happens to the initial knowledge base are the ones dictated by the revision program.

Step 3. Translating the update rules – TR = P + AB*

(a) Each update rule of the type (1)

is translated to the rules:

(3.a.1) $holds(p, res(a, s)) \leftarrow holds(q_1, res(a, s)), \ldots, holds(q_m, res(a, s)),$
$\neg holds(s_1, res(a, s)), \ldots, \neg holds(s_n, res(a, s))$

(3.a.2) $ab(p', a, s) \leftarrow holds(q_1, res(a, s)), \ldots, holds(q_m, res(a, s)),$
$\neg holds(s_1, res(a, s)), \ldots, \neg holds(s_n, res(a, s))$

(b) Each update rule of the type (2)

is translated to the rules:

(3.b.1) $\neg holds(p, res(a, s)) \leftarrow holds(q_1, res(a, s)), \ldots, holds(q_m, res(a, s)),$
$\neg holds(s_1, res(a, s)), \ldots, \neg holds(s_n, res(a, s))$

(3.b.2) $ab(p, a, s) \leftarrow holds(q_1, res(a, s)), \ldots, holds(q_m, res(a, s)),$
$\neg holds(s_1, res(a, s)), \ldots, \neg holds(s_n, res(a, s))$

The rules (3.a.2) and (3.b.2) block the inertia rules and avoid inconsistency. The set of rules obtained from (3.a.2) and (3.b.2) is denoted by AB and the set of rules obtained by (3.a.1) and (3.b.1) is denoted by P*. Note the similarity between P^* and $norm(P)$. The literals $\neg holds(p, s)$ in P^* is replaced by $holds(p', s)$ in $norm(P)$. □

Example 6 Consider D_I and P_2 from Example 3. The translation $\Pi(P_2, D_I)$ consists of the following rules:

$$\left.\begin{array}{l} holds(f, s) \\ holds(g, s) \\ \neg holds(g, res(a, s)) \leftarrow holds(f, res(a, s)) \\ ab(g, a, s) \leftarrow holds(f, res(a, s)) \\ \neg holds(f, res(a, s)) \leftarrow holds(g, res(a, s)) \\ ab(f, a, s) \leftarrow holds(g, res(a, s)) \\ 1.2, 2.1, 2.2 \end{array}\right\} \Pi(P_2, D_I)$$

$\square$

We now state the theorem about the correctness of our translation $\Pi(P, D_I)$.

Theorem 1 Let P be a revision program corresponding to a revision operator a and D_I be an initial database. Let $\Pi(P, D_I)$ be the translation to extended logic programs.

(i) $D_I \xrightarrow{P} D_R$ implies there exists a consistent answer set A of $\Pi(P, D_I)$ such that
(a) $f \in D_R$ iff $holds(f, res(a, s)) \in A$.
(b) $f \notin D_R$ iff $\neg holds(f, res(a, s)) \in A$.

(ii) If A is a consistent answer set of $\Pi(P, D_I)$ then $D_I \xrightarrow{P} D_R$, where $D_R = \{f : holds(f, res(a, s)) \in A\}$

Proof is given in the Appendix.

Example 7 The answer sets of $\Pi(P_2, D_I)$ are

$\{holds(f, s), holds(g, s), holds(f, res(a, s)), \neg holds(g, res(a, s))ab(g, a, s)\}$ and

$\{holds(f, s), holds(g, s), holds(g, res(a, s)), \neg holds(f, res(a, s))ab(f, a, s)\}$ $\square$

4 Rule based Revision of Incomplete Knowledge Bases

The approach in the last section and in [MT94c] assumes that the initial knowledge base is complete. i.e. there is CWA about the initial knowledge base. In this section we define P-justified revision of possibly incomplete knowledge bases with respect to revision specifications. We denote an incomplete knowledge base by a pair of disjoint sets $< D_I^+, D_I^- >$, where D_I^+ is the set of literals true in the knowledge base and D_I^- is the set of literals false in the knowledge base. We say a complete knowledge base D_I agrees with an incomplete knowledge base $< D_I^+, D_I^- >$, if $D_I^+ \subseteq D_I$ and $D_I^- \cap D_I = \emptyset$.

Definition 4 Let $< D_I^+, D_I^- >$ be an incomplete initial knowledge base and P be a revision program. We say $< D_I^+, D_I^- > \xrightarrow{P} D_R$, if there exists a complete database D_I that agrees with $< D_I^+, D_I^- >$ such that $D_I \xrightarrow{P} D_R$. $\square$

Given a revision program P, and an incomplete initial knowledge base $< D_I^+, D_I^- >$, we now present a translation to an extended logic program (denoted by $\Pi_{inc}(P, < D_I^+, D_I^- >)$ that computes the updated knowledge bases.

Algorithm 2 *[Translating Revision Specs – without CWA about the initial database]*

Step 1. Initial Database - IDC = C + ID

If p is in U then $\Pi_{inc}(P, < D_I^+, D_I^- >)$ contains

(1.0) $holds(p, s)$ *or* $\neg holds(p, s)$

If $p \in D_I^+$ then $\Pi_{inc}(P, < D_I^+, D_I^- >)$ contains

(1.1) $holds(p, s)$

If $q \in D_I^-$ then $\Pi_{inc}(P, < D_I^+, D_I^- >)$ contains

(1.2) $\neg holds(q, s)$

Step 2. and *Step 3.* are exactly as in Algorithm 1. $\square$

Theorem 2 Let P be a revision program corresponding to a revision operator a and $< D_I^+, D_I^- >$ be an initial database (possibly incomplete). Let $\Pi_{inc}(P, < D_I^+, D_I^- >)$ be the translation to extended logic programs.

(i) $< D_I^+, D_I^- > \xrightarrow{P} D_R$ implies there exists a consistent answer set A of $\Pi_{inc}(P, < D_I^+, D_I^- >)$ such that for an atom f

(a) $f \in D_R$ iff $holds(f, res(a, s)) \in A$.

(b) $\neg f \in D_R$ iff $\neg holds(f, res(a, s)) \in A$.

(ii) If A is a consistent answer set of $\Pi_{inc}(P, < D_I^+, D_I^- >)$ then $< D_I^+, D_I^- > \xrightarrow{P} D_R$, where $D_R = \{f \ : \ holds(f, res(a, s)) \in A\} \cup \{\neg f \ : \ \neg holds(f, res(a, s)) \in A\}$

Proof:(sketch)

It is easy to see that $\Pi_{inc}(P, < D_I^+, D_I^- >)$ can be split [LT94] to two parts, the bottom part consisting of rules from IDC (i.e. Step 1) and the top part consisting of rules from the rest of the steps.

It is easy to see that A_0 is an answer set [GL91] of IDC iff $D_I = \{f \ : \ holds(f, s) \in A_0$ agrees with $< D_I^+, D_I^- >$. The proof then directly follows from Theorem 1 and the splitting theorem [LT94]. $\square$

5 Specifying updates that depend on the previous state

The revision programs defined in the previous sections can only express the relationship between the elements of the revised knowledge base. Although it uses the implicit assumption that there is minimal change to the initial database, it can not explicitly state any relation between the initial knowledge bases and the revised knowledge base. For example we can not express the effect of an action which is described as "*shooting* at the turkey causes the turkey to be *not alive* and the gun to be *unloaded* if the gun was *loaded*", where we need to refer to the initial state and the updated state at the same time.

Consider the straight forward attempt at representing the effect of 'shoot' by the following revision program P_{shoot}:

$out(alive) \leftarrow in(loaded)$
$out(loaded) \leftarrow in(loaded)$

If $D_I = \{alive, loaded\}$, then with respect to the action "shoot" we expect $D_R = \{\}$. But, it is easy to see that $D_I \stackrel{P_{shoot}}{\longrightarrow} D_R$ is not true.

The reason the revision program P_{shoot} did not work is because while the $in(loaded)$ in the body of the rules in P_{shoot} referred to the initial state the $out(loaded)$ in the head referred to the updated state.

To distinguish between these two states we introduce literals of the form $was\_in(q)$ and $was\_out(t)$, which intuitively mean that q was initially true and t was initially false respectively. More formally, an *extended update rule* can be of the following two forms:

$$\begin{aligned} in(p) \leftarrow\ & in(q_1), \ldots, in(q_m), out(s_1), \ldots, out(s_n), \\ & was\_in(t_1), \ldots, was\_in(t_k), was\_out(u_1), \ldots, was\_out(u_l) \end{aligned} \tag{3}$$

$$\begin{aligned} out(p) \leftarrow\ & in(q_1), \ldots, in(q_m), out(s_1), \ldots, out(s_n), \\ & was\_in(t_1), \ldots, was\_in(t_k), was\_out(u_1), \ldots, was\_out(u_l) \end{aligned} \tag{4}$$

where p, q_i's, s_j's, t_i's and u_j's are atoms.

The effect of the action "shoot" is intuitively represented by the extended revision program:

$out(alive) \leftarrow was\_in(loaded)$
$out(loaded) \leftarrow was\_in(loaded)$

Definition 5 Let P be an extended revision program and D_I and D_R be two knowledge bases.

P^{D_I} is the revision program obtained from P by eliminating from P every rule of the type 3 or 4 such that $t_i \notin D_I$ or $u_j \in D_I$, and eliminating all *was_in* atoms from the resultant program. We then say $D_I \xrightarrow{P} D_R$ iff $D_I \xrightarrow{P^{D_I}} D_R$. □

As in the last section we translate the initial knowledge base and the extended revision program to an extended logic program so as to compute the revised knowledge bases. As in the previous section, our translation uses situation calculus notations. The translation of an initial knowledge base D_I and the revision program P denoted by $\Pi_{ext}(P, D_I)$ consists of the following:

Algorithm 3 *[Translating Extended Revision Programs]*
Our translation will be same as in Algorithm 1 except the translation of the update rules. The extended update rules are translated as follows:

(a) Each extended update rule of the type (3)

is translated to the rules

(3.a.1') $$\begin{aligned} holds(p, res(a,s)) \leftarrow\ & holds(q_1, res(a,s)), \ldots, holds(q_m, res(a,s)), \\ & \neg holds(s_1, res(a,s)), \ldots, \neg holds(s_n, res(a,s)), \\ & holds(t_1, s), \ldots, holds(t_k, s),\ \neg holds(u_1, s), \ldots, \neg holds(u_l, s) \end{aligned}$$

(3.a.2') $$\begin{aligned} ab(p', a, s) \leftarrow\ & holds(q_1, res(a,s)), \ldots, holds(q_m, res(a,s)), \\ & \neg holds(s_1, res(a,s)), \ldots, \neg holds(s_n, res(a,s)), \\ & holds(t_1, s), \ldots, holds(t_k, s),\ \neg holds(u_1, s), \ldots, \neg holds(u_l, s) \end{aligned}$$

(b) Each extended update rule of the type (4)

is translated to the rules

$$(3.b.1')\ \neg holds(p, res(a,s)) \leftarrow holds(q_1, res(a,s)), \ldots, holds(q_m, res(a,s)),$$
$$\neg holds(s_1, res(a,s)), \ldots, \neg holds(s_n, res(a,s)),$$
$$holds(t_1, s), \ldots, holds(t_k, s),\ \neg holds(u_1, s), \ldots, \neg holds(u_l, s)$$

$$(3.b.2')\ ab(p, a, s) \leftarrow holds(q_1, res(a,s)), \ldots, holds(q_m, res(a,s)),$$
$$\neg holds(s_1, res(a,s)), \ldots, \neg holds(s_n, res(a,s)),$$
$$holds(t_1, s), \ldots, holds(t_k, s),\ \neg holds(u_1, s), \ldots, \neg holds(u_l, s)$$

□

By $\Pi_{ext}(P)$ we only denote the translation of the extended update rules in P.

Theorem 3 Let P be an extended revision program corresponding to a revision operator r and D_I be an initial database (possibly incomplete). Let $\Pi_{ext}(P, D_I)$ be the translation to extended logic programs.

(i) $D_I \xrightarrow{P} D_R$ implies there exists a consistent answer set A of $\Pi_{ext}(P, D_I)$ such that for an atom f

(a) $f \in D_R$ iff $holds(f, res(a,s)) \in A$.

(b) $\neg f \in D_R$ iff $\neg holds(f, res(a,s)) \in A$.

(ii) If A is a consistent answer set of $\Pi_{ext}(P, D_I)$ then $D_I \xrightarrow{P} D_R$, where $D_R = \{f\ :\ holds(f, res(a,s)) \in A\} \cup \{\neg f\ :\ \neg holds(f, res(a,s)) \in A\}$ □

Proof: Using Lemma 1 it is easy to see that A is an answer set of $\Pi(P^{D_I}, D_I)$ iff A is an answer set of $\Pi_{ext}(P, D_I)$. Rest follows from Definition 5 and Theorem 1. □

6 Relation with Logic Programming Theories of Actions

Recently, many theories of actions have been developed [BG93, GL92, Dun93, DDS93, HT93, Tur94] together with either translations to logic programs or implementations using logic programs [PR93]. A large number of these works are based on the specification language $\mathcal{A}$ proposed by Gelfond and Lifschitz [GL92]. Gelfond and Lifschitz also provide a sound translation of $\mathcal{A}$ to extended logic programs. Since [GL92], $\mathcal{A}$ was extended to incorporate concurrent actions [BG93], actual situations[BGP95], constraints, etc. and sound and often complete translations of these theories to logic programming formalisms (disjunctive, abductive, equational etc.) [Dun93, DDS93, HT93, Tur94] were given. Since, translation to logic programs were an important part of these works we refer to them as 'Logic Programming theories of actions'.

Allmost all of these translations used the situation calculus notation. Hence, our translation of revision programs to logic programming situation calculus suggests easier integration of revision programs into these theories of actions. We now give a simple illustration of this.

Consider an e-propositions from language $\mathcal{A}$ of the form.

$$a\ causes\ f\ if\ p_1, \ldots, p_n$$

If that is the only e-proposition with A as the action, the effect of the action A can be represented by the following extended revision program (when we assume that $f, p_1, \ldots, p_n$ are positive atoms):

$$in(f) \leftarrow was\_in(p_1), \ldots, was\_in(p_n)$$

In general, the (conditional) effects of an action a can be represented by an extended revision program P_a. Moreover, certain constraints about the domain can also be represented by an extended revision program C. The effect of the action a on a knowledge base D_I taking into account the constraints can then be computed from the program $\Pi(P_a \cup C, D_I)$. The above suggests an extension of $\mathcal{A}$ where constraints are represented as revision programs. (For an extension of $\mathcal{A}$ where constraints are given as a first-order theory see [KL94].)

But there is another concern that needs to be addressed. *How do we reason with several actions?* We now elaborate on this.

Suppose our domain has several actions $a_1, \ldots, a_n$ with their effects (possibly complex) represented by revision programs $P_{a_1}, \ldots, P_{a_n}$. Let s_o be the initial state. For simplicity let us assume that we have complete information about s_0. Now the question is how do we reason about the effect of a sequence of actions.

A simple approach would be to follow Algorithm 1 and use Step 1 (ID) w.r.t. s_0, have IR, and use Step 3 w.r.t. each of the revision programs $P_{a_1}, \ldots, P_{a_n}$. *One problem is that if for some action a_i, P_{a_i} is defined such that it leads to inconsistency then we can not even reason about other actions.*

Without dwelling on the arguments about whether this is desirable or not we describe a method of avoiding it. We introduce a new predicate p_holds (which intuitively means 'possibly holds') and replace all occurances of $holds(F, res(a, s))$ and $\neg holds(F, res(a, s))$ by $p\_holds(F, res(a, S))$ and $p\_holds(F', res(a, S))$ respectively in the rules obtained using Step 2 and Step 3 of Algorithm 1. We then add the following three rules:

$undefined(A, S) \leftarrow ab(F, A, S), ab(F', A, S)$

$holds(F, res(A, S)) \leftarrow p\_holds(F, res(A, S)), not\ undefined(A, S)$

$\neg holds(F, res(A, S)) \leftarrow p\_holds(F', res(A, S)), not\ undefined(A, S)$

It is easy to see that Theorem 1 still holds about this modified translation. But at the same time we avoid having inconsistent answer sets. Now, to reason about the effect and executability of a sequence of actions we only need to replace the constants a and s by the variables A and S in the original program.

7 Conclusion

In this paper we showed how to embed Marek and Truszczynski's revision programs in logic programming situation calculus. We then considered knowledge bases that may be incomplete and presented a translation for computing revision for such a case. We also extended the language of revision programs to allow rules explicitly relating the initial and the updated knowledge base. Finally we discussed how our work impacts the current research in logic programming theories of actions.

Let us briefly backtrack to our discussion on the expressibility of revision programs and discuss the possibility that updates and first-order theories can be combined together to express more than can be done through first-order theories only.

The approach taken in revision programs can be be considered as an approach to compute $D_I \; o \; P$ where D_I is the initial knowledge base, P is a revision program that expresses the theory that we would like to update D_I with, and o is a simple update operator based on symmetric difference. Now suppose D_I is a set of parents and we would like P to express information about ancestors such that the updated knowledge base also has all the ancestor pairs. P can be described by the following schema:

$in(anc(X,Y)) \leftarrow in(par(X,Y))$

$in(anc(X,Y)) \leftarrow in(par(X,Z)), in(anc(Z,Y))$

It is well known that we can not express transitive closure by first-order theory. To overcome this a different approach (than using revision programs) is taken in [GMR92]. They consider P to be a first-order theory but change the update operator o. *An apparent drawback to this alternative approach is that different update operators will be needed for different purposes. The revision programming approach on the other hand stays with the same simple update operator but uses a more expressive language to express P.*

We would now like to make a bold conclusion regarding the elegance and expressibility of Logic Programming Situation Calculus. Logic programming situation calculus combines the elegance of situation calculus with the declarativeness and expressibility of logic programming. We believe languages such as revision programs and transaction logic programs [BK93] that deal with change in knowledge bases be considered only as specification languages (i.e not considered as a completely new language) and a translation of them to Logic Programming Situation Calculus be given. Such an approach will allow lifting of results in logic programming theory to those formalisms, will obviate development of specific implementation of these languages (as available logic programming implementations can then be used), and will make it easier to integrate with the recent work on logic programming theories of actions. Our embedding of revision programs in logic programming situation calculus contributes towards this goal. (For these reasons and since we are really 'updating' rather than 'revising' we will prefer the term 'update specification' to the term 'revision program' in the sequel.) Embedding transaction logic programs in logic programming is still open and will be one of our next goals.

8 Acknowledgment

I would like to thank Prof. W. Marek whose talk on "Revision Programs" in UT El Paso in Dec 93 triggered the ideas expanded on this paper. I would also like to thank Hudson

Turner for his valuable comments. I am thankful to Michael Gelfond and our students for a discussion on 'causality' that made the connections between the update rules and 'causality' clear in my mind. We acknowledge the support of NSF through its grants NSF-IRI-92-11-662 and NSF-IRI-95-01-577.

References

[Bar94] C. Baral. Rule based updates on simple knowledge bases. In *Proc. of AAAI94, Seattle*, pages 136–141, August 1994.

[Bar95] C. Baral. Reasoning about Actions : Non-deterministic effects, Constraints and Qualification. In *Proc. of IJCAI 95*, pages 2017–2023, 1995.

[BG93] C. Baral and M. Gelfond. Representing concurrent actions in extended logic programming. In *Proc. of 13th International Joint Conference on Artificial Intelligence, Chambery, France*, pages 866–871, 1993.

[BGP95] C. Baral, M. Gelfond, and A. Provetti. Representing Actions I: Laws, Observations and Hypothesis. In *Proc. of AAAI 95 Spring Symposium on Extending Theories of Action: Formal theory and practical applications*, 1995.

[BK93] A. Bonner and M. Kifer. Transaction logic rogramming. In D. S. Warren, editor, *Logic Programming: Proc. of the 10th International Conf.*, pages 257–279, 1993.

[DDS93] M. Denecker and D. De Schreye. Representing incomplete knowledge in abductive logic programming. In *Proc. of ILPS 93, Vancouver*, pages 147–164, 1993.

[Dun93] P. Dung. Representing actions in logic programming and its application in database updates. In D. S. Warren, editor, *Proc. of ICLP-93*, pages 222–238, 1993.

[GL90] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In D. Warren and Peter Szeredi, editors, *Logic Programming: Proc. of the Seventh Int'l Conf.*, pages 579–597, 1990.

[GL91] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, pages 365–387, 1991.

[GL92] M. Gelfond and V. Lifschitz. Representing actions in extended logic programs. In *Joint International Conference and Symposium on Logic Programming.*, pages 559–573, 1992.

[GM88] P. Gardenfors and D. Makinson. Revisions in knowledge systems using epistemic entrenchment. In *Proc. 2nd international conference on theoretical aspects of reasoning about knowledge*, pages 1413–1419, 1988.

[GMR92] G. Grahne, A. Mendelzon, and P. Revesz. Knowledge transformations. In *PODS*, pages 246–260, 1992.

[HT93] S Holldobler and M Thielscher. Actions and specificity. In D. Miller, editor, *Proc. of ICLP-93*, pages 164–180, 1993.

[KL94] G. Kartha and V. Lifschitz. Actions with indirect effects: Preliminary report. In *KR 94*, pages 341–350, 1994.

[KM89] H. Katsuno and A. Mendelzon. A unified view of propositional knowledge base updates. In *Proc. of IJCAI-89*, pages 1413–1419, 1989.

[KM92] H. Katsuno and A. Mendelzon. On the difference between updating a knowledge base and revising it. In *Proc. of KR 92*, pages 387–394, 1992.

[Lin95] F. Lin. Embracing causality in specifying the indirect effects of actions. In *Proc. of IJCAI 95*, pages 1985–1993, 95.

[LT94] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In Pascal Van Hentenryck, editor, *Proc. of the Eleventh Int'l Conf. on Logic Programming*, pages 23–38, 1994.

[MH69] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, Edinburgh, 1969.

[MT94a] W. Marek and M. Truszczyński. Revision programming. manuscript, 1994.

[MT94b] W. Marek and M. Truszczyński. Revision programming, database updates and integrity constraints. In *To appear in 5th International conference in Database theory, Prague*, 1994.

[MT94c] W. Marek and M. Truszczyński. Revision specifications by means of programs. manuscript, 1994.

[MT95] N. McCain and M. Turner. A causal theory of ramifications and qualifications. In *Proc. of IJCAI 95*, pages 1978–1984, 95.

[PR93] J. Pinto and R. Reiter. Temporal reasoning in logic programming: A case for the situation calculus. In *Proceedings of 10th International Conference in Logic Programming, Hungary*, pages 203–221, 1993.

[PT95] T. Przymusinski and H. Turner. Update by means of inference rules. In *Proc. of Int'l Conf.on Logic Programming and non-monotonic reasoning*, May-June 1995.

[Sac93] D. Saccá. Multiple stable models are needed to solve unique solution problems. In *Informal Proc. of the Second Compulog Net meeting on Knowledge Bases (CNKBS 93)*, 1993.

[Sch90] J. Schlipf. The expressive power of the logic programming semantics. In *Proc. of the Ninth Symp. on Principles of Database Systems*, pages 196–204, 1990.

[Tur94] H. Turner. Signed logic programs. In *Proc. of the 1994 International Symposium on Logic Programming*, pages 61–75, 1994.

[WC93] D. S. Warren and W. Chen. Query evaluation under well-founded semantics. In *Proc. of PODS 93*, 1993.

[Win89] M. Winslett. Reasoning about action using a possible models approach. In *Proc. of 7th national conference on AI*, pages 89–93, 1989.

Appendix – Proof of Theorem 1

To prove Theorem 1 we need the following lemma. The proof of the Lemma is straightforward.

Lemma 1 Let P be an extended logic program without *not*. Let r be a rule from P given as:
$$l_0 \leftarrow l_1, \ldots, l_m, l_{m+1}, \ldots, l_n$$

(i) Let A be a set of literals such that there exist l_i in the body of rule r such that $l_i \notin A$. Then, A is a minimal set of literals that satisfy P iff A is a minimal set of literals that satisfy $P \setminus \{r\}$.

(ii) If $\{l_{m+1} \leftarrow, \ldots, l_n \leftarrow\} \subseteq P \setminus \{r\}$ then, A is a minimal set of literals that satisfy P iff A is a minimal set of literals that satisfy $P \cup \{l_0 \leftarrow l_1, \ldots, l_m\} \setminus \{r\}$. □

Proof (of Theorem 1)[5]: (ii) Let A be a consistent answer set of $\Pi(P, D_I)$. From the construction of $\Pi(P, D_I)$ it is obvious that $D_I = \{f \ : \ holds(f, s) \in A\}$. Let us denote D'_I, D_R, D'_R, I and O by the following sets:

$D'_I = \{f \ : \ \neg holds(f, s) \in A\}$ $\quad$ $D_R = \{f \ : \ holds(f, res(a, s)) \in A\}$

$D'_R = \{f \ : \ \neg holds(f, res(a, s)) \in A\}$ $\quad$ $I = \{f \ : \ ab(f', a, s) \in A\}$

$O = \{f \ : \ ab(f, a, s) \in A\}$

Now we show that $D_I \xrightarrow{P} D_R$.

A is an answer set of $\Pi(P, D_I)$
$\Rightarrow$ A is the minimal set of literals that satisfies $(\Pi(P, D_I))^A$.
$\Rightarrow$ A is the minimal set of literals that satisfies $ID^A \cup IR^A \cup P^* \cup AB$ $\quad$ (*1)
Let us analyze each of these sets one by one.

$ID^A = \{holds(f, s) \leftarrow : \ f \in D_I\} \cup \{\neg holds(f, s) \leftarrow : \ f \notin D_I\}$. Since we can split [LT94] the program $(\Pi(P, D_I))^A$ into two parts, one consisting of ID^A and the other consisting of

[5]The proof given here is influenced by the proof in [PT95]. In that paper, Przymusinski and Turner independently give another (slightly different) way to embed revision programs. Note that our translation appeared almost an year earlier in [Bar94]. One of the main differences between theirs and our approach is that we use the situation calculus notation which makes integration with logic programming theories of action easier. We also extend revision programs. Our original proof (less elegant than the current one) can be accessed via http://cs.utep.edu/chitta/chitta.html

the rest, using the splitting theorem we obtain that D_I and D'_I partition U.
Hence, $ID^A = \{holds(f,s) \leftarrow : \ f \in D_I\} \cup \{\neg holds(f,s) \leftarrow : \ f \in D'_I\}$.

$$IR^A = \{holds(f, res(a,s) \leftarrow holds(f,S) \ : \ f \notin O\}$$
$$\cup\{\neg holds(f, res(a,s) \leftarrow \neg holds(f,S) \ : \ f \notin I\}$$

Let IR^A* denote the set of rules obtained by partially evaluating IR^A w.r.t. D_I. i.e.
$IR^A* = \{holds(f, res(a,s) \leftarrow : \ f \notin O \ and \ f \in D_I\}$
$\cup \ \{\neg holds(f, res(a,s) \leftarrow : \ f \notin I \ and \ f \in D'_I\}$

Using Lemma 1 (by part (i) we eliminate the rules whose body is not satisfied by A and then by part (ii) we eliminate the body of the remaining rules) we have that
(*1) $\Rightarrow$ A is the minimal set of literals that satisfies $ID^A \cup IR^A * \cup P^* \cup AB$. (*2)

Let us consider P^* and AB. Let $P^*{}_{D_R}|D_I$ denote the set of rules obtained from P^* by (a) first removing rules whose body contains $holds(q, res(a,s))$ when $q \notin D_R$, or $\neg holds(t, res(a,s))$ when $t \notin D'_R$, and then (b) removing literals of the form $holds(q, res(a,s))$ and $\neg holds(t, res(a,s))$ from the body of the remaining rules when $q \in D_I$ and $t \in D'_I$ respectively.

$AB_{D_R}|D_I$ is constructed exactly the same way as $P^*{}_{D_R}|D_I$.

It is easy to see that if $holds(q, res(a,s))$ belongs to the body of some rule in $P^*{}_{D_R}|D_I$ then $q \in D_R \setminus D_I$. Similarly, if $\neg holds(t, res(a,s))$ belongs to the body of some rule in $P^*{}_{D_R}|D_I$ then $t \in D'_R \setminus D'_I$.

Using Lemma 1 we have that (*2) $\Rightarrow$
A is the minimal set of literals that satisfies $ID^A \cup IR^A * \cup P^*{}_{D_R}|D_I \cup AB_{D_R}|D_I$. (*3)
Since the language of ID^A and IR^A* are non-intersecting and are different from the literals appearing in the body of $P^*{}_{D_R}|D_I \cup AB_{D_R}|D_I$,
(*3) $\Rightarrow$
$A = ID^A \cup IR^A * \cup$ the minimal set of literals that satisfies $P^*{}_{D_R}|D_I \cup AB_{D_R}|D_I$. (*4)

Let IO be the minimal set of literals that satisfies $P^*{}_{D_R}|D_I \cup AB_{D_R}|D_I$. It is easy to see that $holds(f, res(a,s)) \in IO$ iff $ab(f,a,s) \in IO$, and similarly, $\neg holds(f, res(a,s)) \in IO$ iff $ab(f',a,s) \in IO$. Notice that $P^*{}_{D_R}|D_I \cup AB_{D_R}|D_I$ can be split to $P^*{}_{D_R}|D_I$ and $AB_{D_R}|D_I$ and hence IO_h the set of literals made up of the predicate $holds$ in IO is the minimal set of literals that satisfies $P^*{}_{D_R}|D_I$.

Also, notice that $(P^*{}_{D_R}|D_I)+$, the program obtained by transforming $(P^*{}_{D_R}|D_I)$ by renaming (as in [GL90]) is the same as $norm(P_{D_R}|D_I)$. Since, A is consistent, IO, a subset of A must be consistent. Hence using results from [GL90][6] P is coherent with necessary change (I,O).

From (*4) $D_R = D_I \setminus O \cup I$, and hence $D_I \xrightarrow{P} D_R$. □

(i) (sketch)
Let $A = \{holds(f, res(r,s)) \ : f \in D_R\} \cup \ \{\neg holds(f, res(r,s)) \ : f \notin D_R\} \cup$
$\{holds(f,s) \ : f \in D_I\} \cup \ \{\neg holds(f,s) \ : f \notin D_I\} \cup$

[6]The exact proposition in [GL90] that we refer to is "A consistent set $S \subseteq Lit$ is an answer set of an extended logic program Π if and only if S^+ is a stable model of Π^+."

$\{ab(f,r,s) \ : f \in O\} \cup \{ab(f',r,s) \ : f \in I\}$

We need to show that A is an answer set of $\Pi(P, D_I)$. i.e. We need to show that A is the smallest set of literals that satisfies $(\Pi(P, D_I))^A$. This can be shown by exactly going backwards in the proof of part (ii). $\Box$