Rule Based Updates on Simple Knowledge Bases

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

Abstract

In this paper we consider updates that are specified as rules and consider simple knowledge bases consisting of ground atoms. We present a translation of the rule based update specifications to extended logic programs using situation calculus notation so as to compute the updated knowledge base. We show that the updated knowledge base that we compute satisfies the update specifications and yet is minimally different from the original database. We then expand our approach to incomplete knowledge bases.
We relate our approach to the standard revision and update operators, the formalization of actions and its effects using situation calculus and the formalization of database evolution using situation calculus.

Introduction

Most work on belief revision in the literature focus on updating theories by sentences in the theory itself. 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).

In this paper[1] we consider updates that are specified as rules (MT94b) (similar to rules in a logic program) and present methods to compute updated knowledge bases when knowledge bases consist of a set of ground atoms.

The following example illustrates the kind of updates that we consider.

Consider a knowledge base consisting of three employees: John, Peter and Carl; which represent a certain department D in an organisation. During an oraganisational shake up the department has to be updated based on the new knowledge that "If John remains in the department D then Peter has to leave the department D and if Carl remains in the department then John has to stay in the department".

[1]Supported by the grants NSF-IRI-92-11-662 and NSF-CDA 90-15-006.

It should be noted that the intended meaning (MT94a) of the first statement is different from the statement "either John leaves the department or Peter leaves the department". If the new knowledge is specified in propositional theory or in first order logic they would be equivalent. The "if" and "then" in the statement "If John remains in the department D then Peter has to leave the department D" are treated differently from the first order implication. Our intent is to give a higher priority to "John than to Peter". This is necessary because we might like to have the language that specifies updates to have properties (say like 'monotonicity') which are different from the ones held by the language of the database.

To specify such rules Marek and Truszczynski (MT94a) introduce the notion of *revision programs*. They define P-justified revision of simple knowledge bases by a revision program. In this paper we show how to compute P-justified revisions of knowledge bases using extended logic programs and situation calculus. Marek and Truszczynski 's definition of P-justified revision is only limited to the case when the CWA is assumed about the initial knowledge base. We extend the idea of P-justified revision to knowledge bases that could be incomplete and present an extended logic program that computes the revised knowledge base when the initial knowledge base may be incomplete.

We then consider update rules that explicitly relates the initial knowledge base to the revised knowledge base and show how revisions can be computed for such updates. Such rules are beyond the scope of Marek and Truszczynski 's revision programs.

Our approach of computing the revised knowledge base is similar to the formalization of database evolution (Rei92) but uses extended logic programs (GL91) instead of first order logic. In its use of situation calculus and extended logic programs our approach treats revision specifications as "actions" and a knowledge base as a "situation" and has similarity to the formalization of actions and their effects in (GL92). Our approach is different than the event calculus approach in (Ko92) and considers more complex revisions than discussed in it.

Revision Specifications

In this section we review the concept of revision specifications[2] and P-justified revision as defined in (MT94b).

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 specification 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 revised knowledge base. Similarly the meaning of $out(a)$ is that the atom a is absent in the revised knowledge base. For any atom p in U, $in(p)$ an $out(p)$ are referred to as r-literals of U.

The statement "If John remains in the department D then Peter has to leave the department D" is written as the revision rule:

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

We now formally define the revision specifications and P-justified revision.

Definition 1 *(MT94b)* A *revision 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 revision rules is called a revision specification. □

A knowledge base B is a *r-model* of (satisfies) an r-literal $in(p)$ ($out(p)$ respectively) if $p \in B$ ($p \notin B$, respectively). B is a r-model of the body of a rule if it satisfies each r-literal of the body. B is a r-model of a rule C if the following conditions hold: whenever B satisfies the body of C, then B satisfies the head of C. B is a r-model of a revision specification P if B satisfies each rule in P.

Definition 2 *(MT94b)* Let P be a revision specification. By $norm(P)$ we denote the definite program obtained from P by replacing each occurance of $in(a)$ by a and each occurance of $out(b)$ by b'. The *necessary change* for P is the pair (I, O) where $I = \{a \ : \ a \in$ least model of $norm(P)\}$ and $O = \{b \ : \ b' \in$ least model of $norm(P)\}$. P with necessary change (I, O) is said to be *coherent* if $I \cap O = \emptyset$. □

[2]Marek and Truszczynski used the term *revision programs* instead of revision specifications. We believe it to be more of a specification language (similar to the language $\mathcal{A}$ (GL92) for specifying effects of actions) that can be implemented in a logical language of choice, rather than a programming language.

Definition 3 *(MT94b)* Let P be a revision specification 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}.

Example 1 Let $D_I = \{a, b\}$ and P_1 be the revision specification

$$out(b) \leftarrow in(a) \ \} \ P_1$$

Let D_R be $\{a\}$.

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

Let P_2 be

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

It is easy to see that P_2-justified revisions of D_I are $\{a\}$ and $\{b\}$.

Let P_3 be

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

It is easy to see that P_3-justified revisions of D_I is $\{b\}$. Intuitively we can consider P_1 as the logic program $\{out\_b \leftarrow not\ out_a\}$ and P_3 as the logic program

$out\_b \leftarrow not\ out\_a$

$out\_a \leftarrow$

Let P_4 be

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

It is easy to see that P_4-justified revisions of D_I is $\{a, b\}$. □

Proposition 1 *(MT94b)* If a knowledge base D satisfies a revision specification P then D is a unique P-justified revision of D. □

Proposition 2 *(MT94b)* Let P be a revision specification and D_I be a knowledge base. If a knowledge base D_R is a P-justified revision of D_I, then D_R is a r-model of P. □

Proposition 3 *(MT94b)* Let P be a revision specification and D_I be a knowledge base. If D_R is a P-justified revision of D_I, then $D_R \div D_I$ is minimal in the family $\{D \div D_I \ : \ D$ is a r-model of $P\}$, where $\div$ denotes the symmetric difference. i.e. $A \div B = (A \setminus B) \cup (B \setminus A)$. □

It should be noted that the above proposition just says P-justified revisions are r-models of the revision specification that are minimally different from the original database. It does not say that r-models of the revision specification that are minimally different from the original database are P-justified revisions. In Example 1 both $\{a\}$ and $\{b\}$ are r-models of P_1 minimally different from D_I but only $\{a\}$ is a P_1-justified revision. This is similar to the logic program $a \leftarrow not\ b$ which has two minimal models $\{a\}$ and $\{b\}$, but has the only stable model $\{a\}$.

Translating Revision specifications to Extended Logic Programs

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

The extended logic program $\Pi(P \cup D_I)$ where P is the revision specification and D_I is the initial knowledge base, uses variables of three sorts: *situation* variables $s, s', \ldots$, *fluent* variables $f, f', \ldots$, and *revision*[3] variables $r, r', \ldots$.

The program $\Pi(P \cup D)$ consists of the translations of the individual revision rules and the initial knowledge base in P and certain other rules. We now present the translation $\Pi(P \cup D)$ where s is the situation corresponding to the initial knowledge base D_I, r correspond to the revision dictated by the revision specification P and $res(r, s)$ is the situation corresponding to the knowledge base obtained by revising the initial knowledge base with the revision specification P.

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

1. Initial Database

If p is proposition in the initial database then $\Pi(P \cup D)$ 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.

2. Inertia Rule

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

[3]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)

This rule is motivated by the minimality consideration that only changes that happens to the initial knowledge base are the ones dictated by the revision specification.

3. Translating the revision rules

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

is translated to the rule

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

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

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

is translated to the rules:

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

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

Since the inertia rule (2.1) is only for the positive facts we do not need a rule defining abnormality corresponding to (3.a.1), but we do need such a rule corresponding to (3.b.1) to block the inertia rule and avoid inconsistency.

4. Completing the revised database

To encode the CWA w.r.t. the revised database $\Pi(P \cup D_I)$ contains the rule

(4.1) $\neg holds(F, res(r, s)) \leftarrow not\ holds(F, res(r, s))$ □

Example 2 Consider D_I and P_2 from Example 1. the translation $\Pi(P_2 \cup D_I)$ consists of the following rules:

$$\left.\begin{array}{l} holds(a, s) \\ holds(b, s) \\ \neg holds(b, res(r_1, s)) \leftarrow holds(a, res(r_1, s)) \\ ab(b, r_1, s) \leftarrow holds(a, res(r_1, s)) \\ \neg holds(a, res(r_1, s)) \leftarrow holds(b, res(r_1, s)) \\ ab(a, r_1, s) \leftarrow holds(b, res(r_1, s) \\ 1.2 \\ 2.1 \\ 4.1 \end{array}\right\} \Pi(P_2 \cup D_I)$$

□

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

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

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

Proof:(sketch)
(i) Let $A = A_1 \cup A_2 \cup A_3 \cup A_4 \cup A_5 \cup A_6$ where,

$A_1 = \{holds(f, res(r,s)) \ : f \in D_I \setminus O\}$

$A_2 = \{holds(f, res(r,s)) \ : f \in I\}$

$A_3 = \{\neg holds(f, res(r,s)) \ : f \notin D_R\}$

$A_4 = \{holds(f,s) \ : f \in D_I\}$

$A_5 = \{\neg holds(f,s) \ : f \notin D_I\} \cup$

$A_6 = \{ab(f,r,s) \ : f \in O\}$

It is easy to see that A is consistent. To show A as an answer set of $\Pi(P \cup D_I)$ we observe that A_1, A_3, A_4, A_5 come from application of the rules (2.1), (4.1), (1.1) and (1.2) respectively and A_2 and A_6 come from combined application of the rules (3.a.1), (3.b.1) and (3.b.2).

Moreover, there is a one to one correspondence between P_{D_R} and R^A where R consists of the rules from (3.a.1) and (3.b.1). □

Example 3 The answer sets of $\Pi(P_2 \cup D_I)$ are

$\{holds(a,s), holds(b,s), holds(a, res(r_1,s)),$
$\neg holds(b, res(r_1,s)) ab(b, r_1, s)\}$ and

$\{holds(a,s), holds(b,s), holds(b, res(r_1,s)),$
$\neg holds(a, res(r_1,s)) ab(a, r_1, s)\}$ □

Rule based Revision of Incomplete Knowledge Bases

The approach in the last section and in (MT94b) 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 believe that it is more intuitive and understandable to define the P-justified revision through a translation to an extended logic program than directly in the style given in the previous section and hence do the former in this section.

Unless otherwise specified from now on by a knowledge base we mean a possibly incomplete knowledge base which is a subset of $U \cup \neg U$. As in the last section we translate the initial knowledge base and the revision specification 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 specification P denoted by $\Pi_{inc}(P \cup D_I)$ consists of the following:

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

1. Initial Database
If p is proposition in the initial knowledge base $\Pi_{inc}(P \cup D_I)$ contains

(1.1) $holds(p,s)$

If $\neg q$ is proposition in the initial knowledge base $\Pi_{inc}(P \cup D_I)$ contains

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

2. Inertia Rules

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

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

Since our initial knowledge base may be incomplete we need two different inertia rules.

3. Translating the revision rules

(a) Each revision rule of the type (1) is translated to the rules (3.a.1) and

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

(b) Each revision rule of the type (2) is translated to the rules (3.b.1) and (3.b.2). □

Definition 4 Let P be a revision specification and D_I be an initial knowledge base. If A is an answer set of $\Pi_{inc}(P \cup D_I)$ then the set $D_R = \{f : holds(f, res(r,s)) \in A\} \cup \{\neg f : \neg holds(f, res(r,s)) \in A\}$ is said to be a P-justified revision of D_I.

A knowledge base B is a *r-i-model* of (satisfies) an r-literal $in(p)$ ($out(p)$ respectively) if $p \in B$ ($\neg p \in B$, respectively). B is a r-i-model of the body of a rule if it satisfies each r-literal of the body. B is a r-i-model of a rule C if the following conditions hold: whenever B satisfies the body of C, then B satisfies the head of C. B is a r-i-model of a revision specification P if B satisfies each rule in P.

Proposition 4 Let P be a revision specification and D_I be a knowledge base. If D_R is a P-justified revision of D_I, then $D_R \div D_I$ is minimal in the family $\{D \div D_I : D$ is a r-i-model of $P\}$, where $\div$ denotes the symmetric difference. i.e. $A \div B = (A \setminus B) \cup (B \setminus A)$. □

Specifying revisions that depend on the previous state

The revision specifications 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 if we would like to say that "all assistant professors with 20 journal papers are to be promoted to associate professors" we can not express it using revision specifications.

In this section we extend revision specifications to allow us to specify such update descriptions.

An *extended revision rule* can be of the following two forms:

$$in(p) \leftarrow in(q_1), \ldots, in(q_m), out(s_1), \ldots, out(s_n), was\_in(t_1), \ldots, was\_in(t_k), was\_out(u_1), \ldots, was\_out(u_l) \quad (3)$$

$$out(p) \leftarrow in(q_1), \ldots, in(q_m), out(s_1), \ldots, out(s_n), was\_in(t_1), \ldots, was\_in(t_k), was\_out(u_1), \ldots, was\_out(u_l) \quad (4)$$

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

An extended revision specification is a collection of extended revision rules.

The statement "all assistant professors with 20 journal papers are to be promoted to associate professors" can be expressed using the following extended revision specification:

$$in(associate(X)) \leftarrow was\_in(assistant(X)), was\_in(haspaper(X, 20))$$

$$out(assistant(X)) \leftarrow was\_in(assistant(X)), was\_in(haspaper(X, 20))$$

Similar to the last section we define revisions with respect to an extended revision specification using extended logic programs and use situation calculus notation.

Algorithm 3 *[Translating Extended Revision Specifications]*

Our translation will be same as in Algorithm 2 except the translation of the revision rules. The extended revision rules are translated as follows:

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

is translated to the rules

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

(3.a.2') $ab(p', a, s) \leftarrow holds(q_1, res(r, s)), \ldots, holds(q_m, res(r, s)), \neg holds(s_1, res(r, s)), \ldots, \neg holds(s_n, res(r, s)), not\ \neg holds(t_1, s), \ldots, not\ \neg holds(t_k, s), not\ holds(u_1, s), \ldots, not\ holds(u_l, s)$

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

is translated to the rules

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

(3.b.2') $ab(p, a, s) \leftarrow holds(q_1, res(r, s)), \ldots, holds(q_m, res(r, s)), \neg holds(s_1, res(r, s)), \ldots, \neg holds(s_n, res(r, s)), not\ \neg holds(t_1, s), \ldots, not\ \neg holds(t_k, s), not\ holds(u_1, s), \ldots, not\ holds(u_l, s)$ □

The use of $not\ \neg holds(t_1, s)$ and $not\ holds(u_l, s)$ instead of $holds(t_1, s)$ and $\neg holds(u_l, s)$ in (3.a.2') and (3.b.2') is to be cautious when applying the inertia rules (GL92). For example if $D_I = \{assistant(john)\}$ and we have the extended revision specification

$$out(assistant(X)) \leftarrow was\_in(haspaper(X, 20))$$

for the update called "promote" we would not like to have $assistant(john)$ in $D_{promote}$ because we are not sure if $haspaper(john, 20)$ is $false$ in D_I. We would rather have $D_{promote}$ contain neither $assistant(john)$ nor $\neg assistant(john)$.

Definition 5 Let P be an extended revision specification and D_I be an initial knowledge base. If A is an answer set of $\Pi_{inc}(P \cup D_I)$ then the set $D_R = \{f : holds(f, res(r, s)) \in A\} \cup \{\neg f : \neg holds(f, res(r, s)) \in A\}$ is said to be a P-justified revision of D_I.

Relationship with standard update operators

In this section we discuss how rule based revision relates to standard revision and update operators.

When we consider a knowledge base to be a set of propositional facts (with CWA) it is easy to see that the concepts of update and revision (KM92) coincide. For such knowledge bases the following proposition relates the standard definition of updates with P-justified revision.

Definition 6 For any revision rule S, f_S is the propositional formula obtained by replacing each $out(a)$ in S by $\neg a$ and each $in(a)$ in S by a and treating $\leftarrow$ as the implication. For any revision specification P, F_P is the propositional formula obtained by the conjunction of all the f_S's, for all S's in P. □

Proposition 5 Let P be a revision specification and D_I be a knowledge base.

D_R is a model (in the propositional sense) of $D_I\ o\ F_P$ where o is the revision operator (KM92) iff $D_R \div D_I$ is minimal in the family $\{D \div D_I : D$ is a r-model of $P\}$ □

When we consider a knowledge base to be a set of literals then a knowledge base may have several models and update and revision (KM92) may be different depending upon the definition of closeness between models and between theories (knowledge bases).

Proposition 6 Let P be a revision specification and D_I be a knowledge base.

D_R is a model of $D_I\ o\ F_P$ where o is the revision operator (KM92) iff $D_R \div D_I$ is minimal in the family $\{D \div D_I : D$ is a r-i-model of $P\}$ □

From the above propositions it is clear that the P-justified revisions computed using the translations suggested in this paper do not compute all the models of the standard revisions (KM92). In Example 1 both $\{a\}$ and $\{b\}$ are r-models of P_1 minimally different from D_I and are also be the models of $D_I \circ F_{P_1}$ but only $\{a\}$ is a P_1-justified revision.

One possible way to obtain all the models would be to translate the revision specification P_1 to a first-order theory instead of an extended logic program and minimize the abnormality using circumscription. That has been the approach of Reiter (Rei92) to specify database evolution. On the other hand in certain cases we might need revisions to be specified as rules instead of a formula and also in certain cases extended logic programs may be preferred over circumscription as a computing formalism.

Relation with $\mathcal{A}$ and its extensions

$\mathcal{A}$ is a specification language for representing effects of actions suggested by Gelfond and Lifschitz in (GL92). The e-propositions in $\mathcal{A}$ which are of the form

$$A \text{ causes } F \text{ if } P_1, \ldots, P_n$$

corresponds to the extended revision rule

$$in(F) \leftarrow was\_in(P_1), \ldots, was\_in(P_n)$$

when the domain consists of only action A and $F, P_1, \ldots, P_n$ are positive atoms.

Revision specifications of the form (1) and (2) are similar to constraints in $\mathcal{AR}$ (KL94), an extension of $\mathcal{A}$. Although the constraints in $\mathcal{AR}$ allow for formulas we believe if rules of the form (1) and (2) are used instead it may be possible to state when a domain description in the language of $\mathcal{AR}$ will have models with unique transition functions.

Conclusion

In this paper we considered the language of *revision specifications* for specifying revision conditions as rules. We presented a translation to extended logic programs that uses situation calculus notation so as to compute the revised knowledge base given a knowledge base consisting of atoms and a revision specification. 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 specifications to allow rules explicitly relating the initial and the revised database. Finally we compared our approach with the standard revise and update operators and with the specification language $\mathcal{A}$.

We believe a more thorough study is necessary to further relate extended revision specifications to standard update operators and also to further relate with languages for reasoning about actions. In particular the impact of using rule based constraints instead of constraint formulas in $\mathcal{AR}$ needs to be studied.

Acknowledgement

I would like to thank Prof. Wiktor 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 the anonymous referees for their valuable comments.

References

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

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

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.

G. Kartha and V. Lifschitz. Actions with indirect effects. To appear in KR 94, 1994.

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

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.

R. Kowalski. Database Updates in the Event Calculus. In *The Journal of Logic Programming*, 12 (1992), 121-146

W. Marek and M. Truszczyński. Revision programming. manuscript, 1994.

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

R. Reiter. Formalizing database evolution in the situation calculus. In ICOT, editor, *Proc. of the International Conference on Fifth Generation Computer Systems*, pages 600–609, 1992.